

Cloud Computing

Gregor von Laszewski

Editor

laszewski@gmail.com

<https://cloudmesh-community.github.io/pub/vonlaszewski-cloud.epub>

November 23, 2019 - 05:21 AM

Created by Cloudmesh & Cyberaide Bookmanager, <https://github.com/cyberaide/bookmanager>

CLOUD COMPUTING

Gregor von Laszewski

(c) Gregor von Laszewski, 2018, 2019

CLOUD COMPUTING

[1 PREFACE](#)

[1.1 Learning Objectives](#) ☼

[1.2 ePub Readers](#) ☼

[1.3 Corrections](#) ☼

[1.4 Contributors](#) ☼

[1.5 Notation](#) ☼

[1.5.1 Figures](#)

[1.5.2 Hyperlinks in the document](#)

[1.5.3 Equations](#)

[1.5.4 Tables](#)

[1.6 Updates](#) ☼

[2 OVERVIEW](#) ☼

[3 DEFINITION OF CLOUD COMPUTING](#) ☼

[3.1 Defining the term Cloud Computing](#)

[3.2 History and Trends](#)

[3.3 Job as a Cloud/Data Engineer](#)

[3.4 You must be that TALLL](#)

[4 DATACENTER](#)

[4.1 Data Center](#) ☼

[4.1.1 Motivation: Data](#)

[4.1.1.1 How much data?](#)

[4.1.2 Cloud Data Centers](#)

[4.1.3 Data Center Infrastructure](#)

[4.1.4 Data Center Characteristics](#)

[4.1.5 Data Center Metrics](#)

[4.1.5.1 Data Center Energy Costs](#)

[4.1.5.2 Data Center Carbon Footprint](#)

[4.1.5.3 Data Center Operational Impact](#)

[4.1.5.4 Power Usage Effectiveness](#)

[4.1.5.5 Hot-Cold Aisle](#)

[4.1.5.5.1 Containment](#)

[4.1.5.5.1.1 Water Cooled Doors](#)

[4.1.5.6 Workload Monitoring](#)

[4.1.5.6.1 Workload of HPC in the Cloud](#)

[4.1.5.6.2 Scientific Impact Metric](#)

[4.1.5.6.3 Clouds and Virtual Machine Monitoring](#)

[4.1.5.6.4 Workload of Containers](#)

[4.1.6 Example Data Centers](#)

[4.1.6.1 AWS](#)

[4.1.6.2 Azure](#)

[4.1.6.3 Google](#)

[4.1.6.4 IBM](#)

[4.1.6.5 XSEDE](#)

[4.1.6.5.1 Comet](#)

[4.1.6.5.2 Jetstream](#)

[4.1.6.6 Chameleon Cloud](#)

[4.1.6.7 Indiana University](#)

[4.1.6.8 Shipping Containers](#)

[4.1.7 Server Consolidation](#)

[4.1.8 Data Center Improvements and Consolidation](#)

[4.1.9 Project Natick](#)

[4.1.10 Renewable Energy for Data Centers](#)

[4.1.11 Societal Shift Towards Renewables](#)

[4.1.12 Datacenter Risks and Issues](#)

[4.1.13 Exercises](#)

[5 ARCHITECTURE](#)

[5.1 Architectures](#) 🍀

[5.1.1 Evolution of Compute Architectures](#)

[5.1.1.1 Mainframe Computing](#)

[5.1.1.2 PC Computing](#)

[5.1.1.3 Intranet and Server Computing](#)

[5.1.1.4 Grid Computing Computing](#)

[5.1.1.5 Internet Computing](#)

[5.1.1.6 Cloud Computing](#)

[5.1.1.7 Mobile Computing](#)

[5.1.1.8 Internet of Things Computing](#)

[5.1.1.9 Edge Computing](#)

[5.1.1.10 Fog Computing](#)

[5.1.2 As a Service Architecture Model](#)

[5.1.3 Product or Functional Based Model](#)

[5.1.4 NIST Cloud Architecture](#)

[5.1.5 Cloud Security Alliance Reference Architecture](#)

[5.1.6 Multicloud Architectures](#)

[5.1.6.1 Cloudmesh Architecture](#)

[5.1.7 Resources](#)

[5.2 NIST Big Data Reference Architecture](#) 🍀

[5.2.1 Pathway to the NIST-BDRA](#)

[5.2.2 Big Data Characteristics and Definitions](#)

[5.2.3 Big Data and the Cloud](#)

[5.2.4 Big Data, Edge Computing and the Cloud](#)

[5.2.5 Reference Architecture](#)

[5.2.6 Framework Providers](#)

[5.2.7 Application Providers](#)

[5.2.8 Fabric](#)

[5.2.9 Interface definitions](#)

[5.3 The Y-Scheduling Architecture View](#) 🍀

[6 REST](#)

[6.1 Introduction to REST](#) 🍀

[6.1.0.1 Collection of Resources](#)

[6.1.0.2 Single Resource](#)

[6.1.0.3 REST Tool Classification](#)

[6.2 OPENAPI 3.0](#)

[6.2.1 REST Specifications](#) 🍀

[6.2.1.1 OPENAPI](#)

[6.2.1.1.1 Open API 3.0 Specification \(OAS 3.0\)](#)

[6.2.1.1.1.1 Definitions](#)

[6.2.1.2 RAML](#)

[6.2.1.3 API Blueprint](#)

[6.2.1.4 JsonAPI](#)

[6.2.1.5 Tinyspec](#)

[6.2.1.6 Tools](#)

[6.2.1.6.1 Connexion](#)

[6.2.2 OpenAPI 3.0 REST Service via Introspection](#) 🍀

[6.2.2.1 Verification](#)

[6.2.2.2 Swagger-UI](#)

[6.2.2.3 Mock service](#)

[6.2.2.4 Exercise](#)

[6.2.3 REST AI services Example](#) 🍀

[6.2.3.1 Service Endpoints/ Paths](#)

[6.2.3.1.1 Path *kmeans/upload*](#)

[6.2.3.1.2 Path *kmeans/fit*](#)

[6.2.3.1.3 Path *kmeans/predict*](#)

[6.2.3.2 Files](#)

[6.2.3.3 Running the example](#)

[6.2.3.4 Notes](#)

[6.3 Flask RESTful Services](#) 🍀

[6.4 Django REST Framework](#) 🍀

[6.5 Github REST Services](#) 🍀

[6.5.1 Issues](#)

[6.5.2 Exercise](#)

[6.6 OpenAPI REST Services with Swagger](#) 🍀

[6.6.1 Swagger Tools](#)

[6.6.2 Swagger Community Tools](#)

[6.6.2.1 Converting Json Examples to OpenAPI YAML Models](#)

[6.7 REST WITH EVE](#)

[6.7.1 Rest Services with Eve](#) 🍀

[6.7.1.1 Ubuntu install of MongoDB](#)

[6.7.1.2 macOS install of MongoDB](#)

[6.7.1.3 Windows 10 Installation of MongoDB](#)

[6.7.1.4 Database Location](#)

[6.7.1.5 Verification](#)

[6.7.1.6 Building a simple REST Service](#)

[6.7.1.7 Interacting with the REST service](#)

[6.7.1.8 Creating REST API Endpoints](#)

[6.7.1.9 REST API Output Formats and Request Processing](#)

[6.7.1.10 REST API Using a Client Application](#)

[6.7.1.11 Towards cmd5 extensions to manage eve and mongo](#) 🍀

[6.7.2 HATEOAS](#) 🍀

[6.7.2.1 Filtering](#)

[6.7.2.2 Pretty Printing](#)

[6.7.2.3 XML](#)

[6.7.3 Extensions to Eve](#) 🍀

[6.7.3.1 Object Management with Eve and Evegenie](#)

[6.7.3.1.1 Installation](#)

[6.7.3.1.2 Starting the service](#)

[6.7.3.1.3 Creating your own objects](#)

[6.8 OPENAPI 2.0](#)

[6.8.1 OpenAPI 2.0 Specification](#) 🍀

[6.8.1.1 The Virtual Cluster example API Definition](#)

[6.8.1.1.1 Terminology](#)

[6.8.1.1.2 Specification](#)

[6.8.1.2 References](#)

[6.8.2 OpenAPI REST Service via Introspection](#) 🍀

[6.8.2.1 Verification](#)

[6.8.2.2 Mock service](#)

[6.8.2.3 Exercise](#)

[6.8.3 OpenAPI REST Service via Codegen](#) 🍀

[6.8.3.1 Step 1: Define Your REST Service](#)

[6.8.3.2 Step 2: Server Side Stub Code Generation and Implementation](#)

[6.8.3.2.1 Setup the Codegen Environment](#)

[6.8.3.2.2 Generate Server Stub Code](#)

[6.8.3.2.3 Fill in the actual implementation](#)

[6.8.3.3 Step 3: Install and Run the REST Service:](#)

[6.8.3.3.1 Start a virtualenv:](#)

[6.8.3.3.2 Make sure you have the latest pip:](#)

[6.8.3.3.3 Install the requirements of the server side code:](#)

[6.8.3.3.4 Install the server side code package:](#)

[6.8.3.3.5 Run the service](#)

[6.8.3.3.6 Verify the service using a web browser:](#)

[6.8.3.4 Step 4: Generate Client Side Code and Verify](#)

[6.8.3.4.1 Client side code generation:](#)

[6.8.3.4.2 Install the client side code package:](#)

[6.8.3.4.3 Using the client API to interact with the REST service](#)

[6.8.3.5 Towards a Distributed Client Server](#)

[6.9 Exercises](#) 🍀

[7 GRAPHQL](#) 🍀

[7.1 Prerequisites](#)

[7.1.1 Install Graphene](#)

[7.1.2 Install Django](#)

[7.1.3 Install GraphiQL](#)

[7.2 GraphQL type system and schema](#)

[7.2.1 Type System](#)

[7.2.2 Scalar Types](#)

[7.2.3 Enumeration Types](#)

[7.2.4 Interfaces](#)

[7.2.5 Union Types](#)

[7.3 GraphQL Query](#)

[7.3.1 Fields](#)

[7.3.2 Arguments](#)

[7.3.3 Fragments](#)

[7.3.4 Variables](#)

[7.3.5 Directives](#)

[7.3.6 Mutations](#)

[7.3.7 Query Validation](#)

[7.4 GraphQL in Python](#)

[7.5 Developing your own GraphQL Server](#)

[7.5.1 GraphQL server implementation](#)

[7.5.2 GraphQL Server Querying](#)

[7.5.3 Mutation example](#)

[7.5.4 GraphQL Authentication](#)

[7.5.5 JSON Web Token Authentication](#)

[7.5.5.1 Using Authentication with Curl](#)

[7.5.5.2 Expiration of JWT tokens](#)

[7.5.6 GitHub API v4](#)

[7.6 Dynamic Queries with GraphQL](#)

[7.7 Advantages of Using GraphQL](#)

[7.8 Disadvantages of Using GraphQL](#)

[7.9 Conclusion](#)

[7.9.1 Resources](#)

[7.10 Excercises](#)

[8 HYPERVISOR](#)

[8.1 Virtualization](#) 🌸

[8.1.1 Virtual Machines](#)

[8.1.2 System Virtual Machines](#)

[8.1.3 Hosted Virtualization](#)

[8.1.4 Summary](#)

[8.1.5 Virtualization Approches](#)

[8.1.5.1 Full virtualization](#)

[8.1.5.2 Paravirtualization](#)

[8.1.6 Virtualization Technologies](#)

[8.1.6.1 Selected Hardware Virtualization Technologies](#)

[8.1.6.2 AMD-V and Intel-VT](#)

[8.1.6.3 I/O MMU virtualization \(AMD-Vi and Intel VT-d\)](#)

[8.1.6.4 Selected VM Virtualization Software and Tools](#)

[8.1.6.4.1 Libvirt](#)

[8.1.6.4.2 QEMU](#)

[8.1.6.4.3 KVM](#)

[8.1.6.4.3.1 KVM vs QEMU](#)

[8.1.6.4.4 Xen](#)

[8.1.6.4.5 Hyper-V](#)

[8.1.6.4.6 VMWare](#)

[8.1.6.5 Parallels](#)

[8.1.6.5.1 VirtualBox](#)

[8.1.6.5.2 Wine – Wine is not an emulator](#)

[8.1.6.5.3 Comparison of some technologies](#)

[8.1.6.6 Selected Storage Virtualization Software and Tools](#)

[8.1.6.7 Selected Network Virtualization Software and Tools](#)

[8.2 Virtual Machine Management with QEMU](#) ☛

[8.2.1 Install QEMU](#)

[8.2.2 Create a Virtual Hard Disk with QEMU](#)

[8.2.3 Install Ubuntu on the Virtual Hard Disk](#)

[8.2.4 Start Ubuntu with QEMU](#)

[8.2.5 Emulate Raspberry Pi with QEMU](#)

[8.2.6 Resources](#)

[8.3 Manage VM guests with virsh](#) ☛

[9 IAAS](#)

[9.1 Introduction](#) ☛

[9.2 Amazon Web Services](#) ☛

[9.2.1 AWS Products](#)

[9.2.1.1 Virtual Machine Infrastructure as a Services](#)

[9.2.1.2 Container Infrastructure as a Service](#)

[9.2.1.3 Serverless Compute using AWS Lambda](#)

[9.2.1.4 Serverless Compute using AWS Lambda](#)

[9.2.1.5 Storage](#)

[9.2.1.6 Databases](#)

[9.2.2 Locations](#)

[9.2.3 Creating an account](#)

[9.2.4 AWS Command Line Interface](#)

[9.2.4.1 Introduction](#)

[9.2.4.2 Prerequisites](#)

[9.2.4.2.1 Install CLI](#)

[9.2.4.2.2 Configure CLI](#)

[9.2.5 AWS Admin Access](#)

[9.2.5.1 Introduction](#)

[9.2.5.2 Prerequisites](#)

[9.2.5.3 Setting up admin access using AWS CLI](#)

[9.2.5.3.1 Create an admin security group](#)

[9.2.5.3.2 Assign a security policy to the created group granting full admin access](#)

[9.2.6 Understanding the free tier](#)

[9.2.7 Important Notes](#)

[9.2.8 Introduction to the AWS console](#)

[9.2.8.1 Starting a VM](#)

[9.2.8.1.1 Setting up key pair](#)

[9.2.8.2 Stopping a VM](#)

[9.2.9 Access from the Command Line](#)

[9.2.10 Access from Python](#)

[9.2.11 Boto](#)

[9.2.12 libcloud](#)

[9.3 Microsoft Azure](#) 

[9.3.1 Products](#)

[9.3.1.1 Virtual Machine Infrastructure as a Services](#)

[9.3.1.2 Container Infrastructure as a Service](#)

[9.3.1.3 Databases](#)

[9.3.1.4 Networking](#)

[9.3.2 Registration](#)

[9.3.3 Introduction to the Azure Portal](#)

[9.3.4 Creating a VM](#)

[9.3.5 Create a Ubuntu Server 18.04 LTS Virtual Machine in Azure](#)

[9.3.6 Remote access the Virtual Machine](#)

[9.3.7 Starting a VM](#)

[9.3.8 Stopping the VM](#)

[9.3.9 Exercises](#)

[9.4 What is IBM Watson and why is it important? 🌸](#)

[9.4.1 How can we use Watson?](#)

[9.4.2 Creating an account](#)

[9.4.3 Understanding the free tier](#)

[9.5 Google IaaS Cloud Services 🌸](#)

[9.5.1 Cloud Computing Services and Products](#)

[9.5.1.1 Overview](#)

[9.5.1.2 AI and Machine Learning](#)

[9.5.1.3 API management](#)

[9.5.1.4 Compute](#)

[9.5.1.5 Data Analytics](#)

[9.5.1.6 Databases](#)

[9.5.1.7 Developer Tools](#)

[9.5.1.8 Internet of Things](#)

[9.5.1.9 Management Tools](#)

[9.5.1.10 Media and Migration](#)

[9.5.2 Migration](#)

[9.5.2.1 Networking](#)

[9.5.2.2 Security](#)

[9.5.2.3 Storage](#)

[9.5.2.4 Google IaaS Example](#)

[9.5.2.5 Google Cloud Console Overview](#)

[9.5.2.6 Use GCP Resources](#)

[9.5.2.7 Project navigation](#)

[9.5.2.8 Navigate Google Cloud Services](#)

[9.5.2.9 Section pinning](#)

[9.5.2.10 View activity across your GCP resources](#)

[9.5.2.11 Search across Cloud Console](#)

[9.5.2.12 Get support anytime](#)

[9.5.2.13 Manage users and permissions](#)

[9.5.2.14 Access the command line from your browser](#)

[9.5.3 Create a VM Example](#)

[9.5.3.1 Create a virtual machine instance](#)

[9.5.3.2 VM instances page](#)

[9.5.3.3 Connect to your instance](#)

[9.5.3.4 Run a simple web server](#)

[9.5.3.5 Visit your application](#)

[9.5.3.6 Cleanup](#)

[9.6 OpenStack](#)

[9.6.1 Introduction](#)

[9.6.2 OpenStack Architecture](#)

[9.6.3 Components](#)

[9.6.4 Core Services](#)

[9.6.4.1 Nova - Compute](#)

[9.6.4.2 Glance - Image Services](#)

[9.6.4.3 Swift - Object Storage](#)

[9.6.4.4 Cinder - Block Storage](#)

[9.6.4.5 Neutron - Networking](#)

[9.6.4.6 Horizon - Dashboard](#)

[9.6.4.7 Keystone - Identity Service](#)

[9.6.4.8 Ceilometer - Telemetry](#)

[9.6.4.9 Heat - Orchestration](#)

[9.6.5 Access from Python and Scripts](#)

[9.6.5.1 Libcloud](#)

[9.6.5.2 DevStack](#)

[9.7 Python Libcloud](#)

[9.7.1 Service categories](#)

[9.7.1.0.1 Compute](#)

[9.7.1.0.2 Key Pair Management](#)

[9.7.1.0.3 Block Storage](#)

[9.7.2 Installation](#)

[9.7.3 Quick Example](#)

[9.7.4 Managing your cloud credentials](#)

[9.7.5 Working with cloud services](#)

[9.7.5.1 Authenticating with cloud providers](#)

[9.7.5.1.1 Amazon AWS](#)

[9.7.5.1.2 Azure](#)

[9.7.5.1.2.1 Azure Classic Driver](#)

[9.7.5.1.2.2 Azure New Driver](#)

[9.7.5.1.3 OpenStack](#)

[9.7.5.1.4 Google](#)

[9.7.5.2 Invoking services](#)

[9.7.5.2.1 Creating Nodes](#)

[9.7.5.2.2 Listing Nodes](#)

[9.7.5.2.3 Starting Nodes](#)

[9.7.5.2.4 Stopping Nodes](#)

[9.7.6 Cloudmesh Community Program to Manage Clouds](#)

[9.7.7 Amazon Simple Storage Service S3 via libcloud](#) 

[9.7.7.1 Access key](#)

[9.7.7.2 Create a new bucket on AWS S3](#)

[9.7.7.3 List Containers](#)

[9.7.7.4 List container objects](#)

[9.7.7.5 Upload a file](#)

[9.7.7.6 References](#)

[9.8 AWS Boto](#)  

[9.8.1 Boto versions](#)

[9.8.2 Boto Installation](#)

[9.8.3 Access key](#)

[9.8.4 Boto configuration](#)

[9.8.5 Boto configuration with cloudmesh](#)

[9.8.6 EC2 interface of Boto](#)

[9.8.6.0.1 Create connection](#)

[9.8.7 List EC2 instances](#)

[9.8.7.0.1 Launch a new instance](#)

[9.8.7.0.2 Check running instances](#)

[9.8.7.0.3 Stop instance](#)

[9.8.7.0.4 Terminate instance](#)

[9.8.7.1 Reboot instances](#)

[9.8.8 Amazon S3 interface of Boto](#)

[9.8.8.0.1 Create connection](#)

[9.8.8.0.2 Create new bucket in S3](#)

[9.8.8.0.3 Upload data](#)

[9.8.8.0.4 List all buckets](#)

[9.8.8.0.5 List all objects in a bucket](#)

[9.8.8.0.6 Delete object](#)

[9.8.8.0.7 Delete bucket](#)

[9.8.9 References](#)

[9.8.10 Exercises](#)

[10 MAPREDUCE](#)

[10.1 Introduction to Mapreduce](#) 

[10.1.1 MapReduce Algorithm](#)

[10.1.1.1 MapReduce Example: Word Count](#)

[10.1.2 Hadoop MapReduce and Hadoop Spark](#)

[10.1.2.1 Apache Spark](#)

[10.1.2.2 Hadoop MapReduce](#)

[10.1.2.3 Key Differences](#)

[10.1.3 References](#)

[10.2 HADOOP](#)

[10.2.1 Hadoop](#) 🍀

[10.2.1.1 Hadoop and MapReduce](#)

[10.2.1.2 Hadoop EcoSystem](#)

[10.2.1.3 Hadoop Components](#)

[10.2.1.4 Hadoop and the Yarn Resource Manager](#)

[10.2.1.5 PageRank](#)

[10.2.2 Installation of Hadoop](#) 🍀

[10.2.2.1 Releases](#)

[10.2.2.2 Prerequisites](#)

[10.2.2.3 User and User Group Creation](#)

[10.2.2.4 Configuring SSH](#)

[10.2.2.5 Installation of Java](#)

[10.2.2.6 Installation of Hadoop](#)

[10.2.2.7 Hadoop Environment Variables](#)

[10.2.3 Hadoop Distributed File System \(Hadoop HDFS\)](#) 🍀

[10.2.3.1 Introduction](#)

[10.2.3.2 Features](#)

[10.2.3.3 HDFS Components](#)

[10.2.3.3.1 NameNode and DataNodes](#)

[10.2.3.4 Usage](#)

[10.2.3.4.1 Java Client API](#)

[10.2.3.4.2 FS Shell](#)

[10.2.3.5 References](#)

[10.2.3.6 Exercises](#)

[10.2.4 Apache HBase](#) 🍀

[10.2.4.1 Introduction](#)

[10.2.4.2 Features](#)

[10.2.4.3 Configuration](#)

[10.2.4.4 Usage](#)

[10.2.4.4.1 Connect to HBase.](#)

- [10.2.4.4.2 Create a table](#)
- [10.2.4.4.3 Describe a table](#)
- [10.2.4.4.4 HBase MapReduce job](#)

[10.2.4.5 References](#)

[10.2.5 Hadoop Virtual Cluster Installation Using Cloudmesh](#)

[10.2.5.1 Cloudmesh Cluster Installation](#)

- [10.2.5.1.1 Create Cluster](#)
- [10.2.5.1.2 Check Created Cluster](#)
- [10.2.5.1.3 Delete Cluster](#)

[10.2.5.2 Hadoop Cluster Installation](#)

- [10.2.5.2.1 Create Hadoop Cluster](#)
- [10.2.5.2.2 Delete Hadoop Cluster](#)

[10.2.5.3 Advanced Topics with Hadoop](#)

- [10.2.5.3.1 Hadoop Virtual Cluster with Spark and/or Pig](#)
- [10.2.5.3.2 Word Count Example on Spark](#)

[10.3 SPARK](#)

[10.3.1 Spark Lectures](#)

- [10.3.1.1 Motivation for Spark](#)
- [10.3.1.2 Spark RDD Operations](#)
- [10.3.1.3 Spark DAG](#)
- [10.3.1.4 Spark vs. other Frameworks](#)

[10.3.2 Installation of Spark](#)

- [10.3.2.1 Prerequisites](#)
- [10.3.2.2 Installation of Java](#)
- [10.3.2.3 Install Spark with Hadoop](#)
- [10.3.2.4 Spark Environment Variables](#)
- [10.3.2.5 Test Spark Installation](#)
- [10.3.2.6 Install Spark With Custom Hadoop](#)
- [10.3.2.7 Configuring Hadoop](#)
- [10.3.2.8 Test Spark Installation](#)

[10.3.3 Spark Streaming](#)

- [10.3.3.1 Streaming Concepts](#)
- [10.3.3.2 Simple Streaming Example](#)
- [10.3.3.3 Spark Streaming For Twitter Data](#)
 - [10.3.3.3.1 Step 1](#)
 - [10.3.3.3.2 Step 2](#)
 - [10.3.3.3.3 Step 3](#)

[10.3.3.3.4 Step 4](#)

[10.3.3.3.5 step 5](#)

[10.3.3.3.6 step 6](#)

[10.3.4 User Defined Functions in Spark](#) 🌸

[10.3.4.1 Resources](#)

[10.3.4.2 Instructions for Spark installation](#)

[10.3.4.2.1 Linux](#)

[10.3.4.3 Windows](#)

[10.3.4.4 MacOS](#)

[10.3.4.5 Instructions for creating Spark User Defined Functions](#)

[10.3.4.5.1 Example: Temperature conversion](#)

[10.3.4.5.1.1 Description about data set](#)

[10.3.4.5.1.2 How to write a python program with UDF](#)

[10.3.4.5.1.3 How to execute a python spark script](#)

[10.3.4.5.1.4 Filtering and sorting](#)

[10.3.4.6 Instructions to install and run the example using docker](#)

[10.4 HADOOP ECOSYSTEM](#)

[10.4.1 ELASTIC MAP REDUCE](#)

[10.4.1.1 AWS Elastic Map Reduce \(AWS EMR\)](#) 🌸

[10.4.1.1.1 Introduction](#)

[10.4.1.1.2 Why EMR?](#)

[10.4.1.1.3 Understanding Clusters and Nodes](#)

[10.4.1.1.4 Prerequisites](#)

[10.4.1.1.5 Creating EMR Cluster Using CLI](#)

[10.4.1.1.5.1 Create Security Roles](#)

[10.4.1.1.5.2 Setting up authentication](#)

[10.4.1.1.5.3 Determine the applicable subnet](#)

[10.4.1.1.5.4 Create the EMR cluster](#)

[10.4.1.1.5.5 Check the status of your cluster](#)

[10.4.1.1.5.6 Terminate your cluster](#)

[10.4.1.1.6 Creating EMR Cluster Using AWS Web Console](#)

[10.4.1.1.6.1 Set up authentication](#)

[10.4.1.1.6.2 Create the EMR cluster](#)

[10.4.1.1.6.3 View status and terminate EMR cluster](#)

[10.4.1.1.6.4 Submit Work to a Cluster](#)

[10.4.1.1.6.5 Processing Data](#)

[10.4.1.1.7 AWS Storage](#)

[10.4.1.1.8 Create EMR in AWS](#)

[10.4.1.1.8.1 Create the buckets](#)

[10.4.1.1.8.2 Create Key Pairs](#)

[10.4.1.1.9 Create Step Execution – Hadoop Job](#)

[10.4.1.1.10 Create a Hive Cluster](#)

[10.4.1.1.10.1 Create a Hive Cluster - Screen shots](#)

[10.4.1.1.11 Create a Spark Cluster](#)

[10.4.1.1.11.1 Create a Spark Cluster - Screenshots](#)

[10.4.1.1.12 Run an example Spark job on an EMR cluster](#)

[10.4.1.1.12.1 Spark Job Description](#)

[10.4.1.1.12.2 Creating the S3 bucket](#)

[10.4.1.1.12.3 Copy files to S3](#)

[10.4.1.1.12.4 Execute the Spark job on a running cluster](#)

[10.4.1.1.12.5 Execute the Spark job while creating clusters](#)

[10.4.1.1.12.6 View the results of the Spark job](#)

[10.4.1.1.13 Conclusion](#)

[10.4.2 TWISTER](#)

[10.4.2.1 Twister2](#) 

[10.4.2.1.1 Introduction](#)

[10.4.2.1.2 Twister2 API's](#)

[10.4.2.1.2.1 TSet API](#)

[10.4.2.1.2.2 Task API](#)

[10.4.2.1.3 Operator API](#)

[10.4.2.1.3.1 Resources](#)

[10.4.2.2 Twister2 Installation](#) 

[10.4.2.2.1 Prerequisites](#)

[10.4.2.2.1.1 Maven Installation](#)

[10.4.2.2.1.2 OpenMPI Installation](#)

[10.4.2.2.1.3 Install Extras](#)

[10.4.2.2.1.4 Compiling Twister2](#)

[10.4.2.2.1.5 Twister2 Distribution](#)

[10.4.2.3 Twister2 Examples](#) 

[10.4.2.3.1 Submitting a Job](#)

[10.4.2.3.2 Batch WordCount Example](#)

[10.4.3 HADOOP RDMA](#) 

[10.4.3.1 Launching a Virtual Hadoop Cluster on Bare-metal InfiniBand Nodes with SR-IOV on Chameleon](#)

[10.4.3.2 Launching Virtual Machines Manually](#)

[10.4.3.3 Extra Initialization when Launching Virtual Machines](#)

[10.4.3.4 Important Note for Tearing Down Virtual Machines and Deleting Network Ports](#)

[11 CONTAINER](#)

[11.1 Introduction to Containers](#) 🍀

[11.1.1 Motivation - Microservices](#)

[11.1.2 Motivation - Serverless Computing](#)

[11.1.3 Docker](#)

[11.1.4 Docker and Kubernetes](#)

[11.2 DOCKER](#)

[11.2.1 Introduction to Docker](#) 🍀

[11.2.1.1 Docker Engine](#)

[11.2.1.2 Docker Architecture](#)

[11.2.1.3 Docker Survey](#)

[11.2.2 Running Docker Locally](#) 🍀

[11.2.2.1 Installation for OSX](#)

[11.2.2.2 Installation for Ubuntu](#)

[11.2.2.3 Installation for Windows 10](#)

[11.2.2.4 Testing the Install](#)

[11.2.3 Dockerfile](#) 🍀

[11.2.3.1 Specification](#)

[11.2.3.2 References](#)

[11.2.4 Docker Hub](#) 🍀

[11.2.4.1 Create Docker ID and Log In](#)

[11.2.4.2 Searching for Docker Images](#)

[11.2.4.3 Pulling Images](#)

[11.2.4.4 Create Repositories](#)

[11.2.4.5 Pushing Images](#)

[11.2.4.6 Resources](#)

[11.2.5 Docker Compose](#) 🍀

[11.2.5.1 Introduction](#)

[11.2.5.2 Installation](#)

[11.2.5.2.1 Install on MacOS](#)

[11.2.5.2.2 Install on Linux](#)

[11.2.5.2.3 Install on Windows 10](#)

[11.2.5.2.3.1 System Requirements](#)

[11.2.5.2.4 Test the installation](#)

[11.2.5.3 Docker Compose File Directives](#)

[11.2.5.3.1 Configuration](#)

[11.2.5.3.1.1 `build`](#)

[11.2.5.3.1.2 `context`](#)

[11.2.5.3.1.3 `ARGS`](#)

[11.2.5.3.1.4 `command`](#)

[11.2.5.3.1.5 `depends_on`](#)

[11.2.5.3.1.6 `image`](#)

[11.2.5.3.1.7 `ports`](#)

[11.2.5.3.1.8 `volumes`](#)

[11.2.5.4 Usages](#)

[11.2.5.4.1 Build A Service depending on MongoDB](#)

[11.3 DOCKER PAAS](#)

[11.3.1 Docker Clusters](#) 🍀

[11.3.2 Docker Swarm](#) 🍀

[11.3.2.1 Terminology](#)

[11.3.2.2 Creating a Docker Swarm Cluster](#)

[11.3.2.3 Create a Swarm Cluster with VirtualBox](#)

[11.3.2.4 Initialize the Swarm Manager Node and Add Worker Nodes](#)

[11.3.2.5 Deploy the application on the swarm manager](#)

[11.3.3 Docker and Docker Swarm on FutureSystems](#) 🍀

[11.3.3.1 Getting Access](#)

[11.3.3.2 Creating a service and deploy to the swarm cluster](#)

[11.3.3.3 Create your own service](#)

[11.3.3.4 Publish an image privately within the swarm cluster](#)

[11.3.3.5 Exercises](#)

[11.3.4 Hadoop with Docker](#) 🍀

[11.3.4.1 Building Hadoop using Docker](#)

[11.3.4.2 Hadoop Configuration Files](#)

[11.3.4.3 Virtual Memory Limit](#)

[11.3.4.4 hdfs Safemode leave command](#)

[11.3.4.5 Examples](#)

[11.3.4.5.1 Statistical Example with Hadoop](#)

[11.3.4.5.1.1 Base Location](#)

[11.3.4.5.1.2 Input Files](#)

[11.3.4.5.1.3 Compilation](#)

- [11.3.4.5.1.4 Archiving Class Files](#)
- [11.3.4.5.1.5 HDFS for Input/Output](#)
- [11.3.4.5.1.6 Run Program with a Single Input File](#)
- [11.3.4.5.1.7 Result for Single Input File](#)
- [11.3.4.5.1.8 Run Program with Multiple Input Files](#)
- [11.3.4.5.1.9 Result for Multiple Files](#)

[11.3.4.5.2 Conclusion](#)

[11.3.4.6 Refernces](#)

[11.3.5 Docker Pagerank](#) 🍀

[11.3.5.1 Use the automated script](#)

[11.3.5.2 Compile and run by hand](#)

[11.3.6 Apache Spark with Docker](#) 🍀

[11.3.6.1 Pull Image from Docker Repository](#)

[11.3.6.2 Running the Image](#)

[11.3.6.2.1 Running interactively](#)

[11.3.6.2.2 Running in the background](#)

[11.3.6.3 Run Spark](#)

[11.3.6.3.1 Run Spark in Yarn-Client Mode](#)

[11.3.6.3.2 Run Spark in Yarn-Cluster Mode](#)

[11.3.6.4 Observe Task Execution from Running Logs of SparkPi](#)

[11.3.6.5 Write a Word-Count Application with Spark RDD](#)

[11.3.6.5.1 Launch Spark Interactive Shell](#)

[11.3.6.5.2 Program in Scala](#)

[11.3.6.5.3 Launch PySpark Interactive Shell](#)

[11.3.6.5.4 Program in Python](#)

[11.3.6.6 Docker Spark Examples](#)

[11.3.6.6.1 K-Means Example](#)

[11.3.6.6.2 Join Example](#)

[11.3.6.6.3 Word Count](#)

[11.3.6.7 Interactive Examples](#)

[11.3.6.7.1 Stop Docker Container](#)

[11.3.6.7.2 Start Docker Container Again](#)

[11.3.6.7.3 Remove Docker Container](#)

[11.4 KUBERNETES](#)

[11.4.1 Introduction to Kubernetes](#) 🍀

[11.4.1.1 What are containers?](#)

[11.4.1.2 Terminology](#)

- [11.4.1.3 Kubernetes Architecture](#)
- [11.4.1.4 Minikube](#)
 - [11.4.1.4.1 Install minikube](#)
 - [11.4.1.4.2 Start a cluster using Minikube](#)
 - [11.4.1.4.3 Create a deployment](#)
 - [11.4.1.4.4 Expose the servi](#)
 - [11.4.1.4.5 Check running status](#)
 - [11.4.1.4.6 Call service api](#)
 - [11.4.1.4.7 Take a look from Dashboard](#)
 - [11.4.1.4.8 Delete the service and deployment](#)
 - [11.4.1.4.9 Stop the cluster](#)
- [11.4.1.5 Interactive Tutorial Online](#)
- [11.4.2 Using Kubernetes on FutureSystems !\[\]\(f19758e515f41afbde8285185935a25a_img.jpg\)](#)
 - [11.4.2.1 Getting Access](#)
 - [11.4.2.2 Example Use](#)
 - [11.4.2.3 Exercises](#)
- [11.5 Running Singularity Containers on Comet !\[\]\(4ceafee3b4c4c24977a8940650ef21a7_img.jpg\)](#)
 - [11.5.1 Background](#)
 - [11.5.2 Tutorial Contents](#)
 - [11.5.3 Why Singularity?](#)
 - [11.5.4 Hands-On Tutorials](#)
 - [11.5.5 Downloading & Installing Singularity](#)
 - [11.5.5.1 Download & Unpack Singularity](#)
 - [11.5.5.2 Configure & Build Singularity](#)
 - [11.5.5.3 Install & Test Singularity](#)
 - [11.5.6 Building Singularity Containers](#)
 - [11.5.6.1 Upgrading Singularity](#)
 - [11.5.7 Create an Empty Container](#)
 - [11.5.8 Import Into a Singularity Container](#)
 - [11.5.9 Shell Into a Singularity Container](#)
 - [11.5.10 Write Into a Singularity Container](#)
 - [11.5.11 Bootstrapping a Singularity Container](#)
 - [11.5.12 Running Singularity Containers on Comet](#)
 - [11.5.12.1 Transfer the Container to Comet](#)
 - [11.5.12.2 Run the Container on Comet](#)
 - [11.5.12.3 Allocate Resources to Run the Container](#)
 - [11.5.12.4 Integrate the Container with Slurm](#)

[11.5.12.5 Use Existing Comet Containers](#)

[11.5.13 Using Tensorflow With Singularity](#)

[11.5.14 Run the job](#)

[11.5.15 Resources](#) 🍀

[11.5.15.1 Tutorialspoint](#)

[11.6 Exercises](#) 🍀

[12 SERVERLESS](#)

[12.1 FaaS](#) 🍀

[12.1.1 Introduction](#)

[12.1.2 Serverless Computing](#)

[12.1.3 Faas provider](#)

[12.1.4 Resources](#)

[12.1.5 Usage Examples](#)

[12.2 AWS Lambda](#) 🍀

[12.2.1 AWS Lambda Features](#)

[12.2.2 Understanding Function limitations](#)

[12.2.2.1 Execution Time](#)

[12.2.2.2 Function size](#)

[12.2.3 Understanding the free Tier](#)

[12.2.4 Writing your fist Lambda function](#)

[12.2.5 AWS Lambda Usecases](#)

[12.2.6 AWS Lambda Example](#)

[12.3 Apache OpenWhisk](#) 🍀

[12.3.1 OpenWhisk Workflow](#)

[12.3.1.1 The Action and Nginx](#)

[12.3.1.2 Controller: The System's Interface](#)

[12.3.1.3 CouchDB](#)

[12.3.1.4 Load Balancer](#)

[12.3.1.5 Kafka](#)

[12.3.1.6 Invoker](#)

[12.3.1.7 CouchDB again](#)

[12.3.2 Setting Up OpenWhisk Locally](#)

[12.3.2.1 Debugging quick-start](#)

[12.3.3 Hello World in OpenWhisk](#)

[12.3.4 Creating a custom action](#)

[12.4 Kubeless](#) 🍀

[12.4.1 Introduction](#)

[12.4.2 Programing model](#)

[12.4.3 System Architecture](#)

[12.5 Microsoft Azure Function](#)  

[12.6 Google Cloud Functions](#) 

[12.6.1 Google Cloud Function Example](#)

[12.7 OpenFaaS](#) 

[12.7.1 OpenFaas Components and Architecture](#)

[12.7.1.1 API Gateway](#)

[12.7.1.2 Function Watchdog](#)

[12.7.1.3 OpenFaas CLI](#)

[12.7.1.4 Monitoring](#)

[12.7.2 OpenFaas in Action](#)

[12.7.2.1 Prerequistics](#)

[12.7.2.2 Single Node Cluster](#)

[12.7.2.3 Deploy OpenFaas](#)

[12.7.2.4 To Run OpenFaas](#)

[12.7.3 OpenFaaS Function with Python](#)

[12.8 OpenLamda](#) 

[12.8.1 Suggested Materials](#)

[12.8.2 Development](#)

[12.8.3 OpenLambda](#)

[12.8.4 Getting Started](#)

[12.8.4.1 Install Dependencies](#)

[12.8.4.2 Start a Test Cluster](#)

[12.8.5 Administration](#)

[12.8.5.1 Writing Handlers](#)

[12.8.5.2 Cluster Directory](#)

[12.8.6 Configuration](#)

[12.8.7 Architecture](#)

[13 MESSAGING](#)

[13.1 MQTT](#) 

[13.1.1 Introduction](#)

[13.1.2 Publish Subscribe Model](#)

[13.1.2.1 Topics](#)

[13.1.2.2 Callbacks](#)

[13.1.2.3 Quality of Service](#)

[13.1.3 Secure MQTT Services](#)

- [13.1.3.1 Using TLS/SSL](#)
 - [13.1.3.2 Using OAuth](#)
 - [13.1.4 Integration with Other Services](#)
 - [13.1.5 MQTT in Production](#)
 - [13.1.6 Installation](#)
 - [13.1.6.1 MacOS install](#)
 - [13.1.6.2 MacOS Advanced Service install](#)
 - [13.1.6.3 Ubuntu install](#)
 - [13.1.6.4 Raspberry Pi Setup](#)
 - [13.1.6.4.1 Broker](#)
 - [13.1.6.4.2 Client](#)
 - [13.1.7 Server Usecase](#)
 - [13.1.8 IoT Use Case with a Raspberry PI](#)
 - [13.1.8.1 Requirements and Setup](#)
 - [13.1.8.2 Results](#)
 - [13.1.9 Conclusion](#)
 - [13.1.10 Exercises](#)
- [13.2 Python Apache Avro](#) 🍀
 - [13.2.1 Download, Unzip and Install](#)
 - [13.2.2 Defining a schema](#)
 - [13.2.3 Serializing](#)
 - [13.2.4 Deserializing](#)
 - [13.2.5 Resources](#)
- [14 GO](#)
 - [14.1 Introduction to Go for Cloud Computing](#) 🍀
 - [14.1.1 Organization of the chapter](#)
 - [14.1.2 References](#)
 - [14.2 Installation](#) 🍀
 - [14.3 Editors Supporting Go](#) 🍀
 - [14.4 Go Language](#) 🍀
 - [14.4.1 Concurrency in Go](#)
 - [14.4.1.1 GoRoutines \(execution\)](#)
 - [14.4.1.2 Channels \(communication\)](#)
 - [14.4.1.3 Select \(coordination\)](#)
 - [14.5 Libraries](#) 🍀
 - [14.6 Go CMD](#) 🍀
 - [14.6.1 CMD](#)

[14.6.2 DocOpts](#)

[14.7 Go REST](#) 🍀

[14.7.1 Gorilla](#)

[14.7.2 REST, RESTful](#)

[14.7.3 Router](#)

[14.7.4 Full code](#)

[14.8 Open API](#) 🍀

[14.8.1 Install from Homebrew](#)

[14.8.2 serve specification UI](#)

[14.8.3 validate a specification](#)

[14.8.4 Generate a Go OpenAPI server](#)

[14.8.5 generate a Go OpenAPI client](#)

[14.8.6 generate a spec from the source](#)

[14.8.7 generate a data model](#)

[14.8.8 other editors](#)

[14.9 Create an Echo service using Swagger and Go](#)

[14.9.1 Dependencies](#)

[14.9.2 Initialize a Golang project](#)

[14.9.3 Define APIs and generate code in Go](#)

[14.9.4 Implement the functionality](#)

[14.9.5 Run and test the server](#)

[14.9.6 References](#)

[14.10 Go Cloud](#) 🍀

[14.10.1 Golang Openstack Client](#)

[14.10.2 OpenStack from Go](#)

[14.10.2.1 GohperCloud](#)

[14.10.2.1.1 Authentication](#)

[14.10.2.1.2 Virtual machines](#)

[14.10.2.1.3 Resources](#)

[14.11 Go Links](#) 🍀

[14.11.1 Introductory Material](#)

[14.11.2 The GO Language](#)

[14.11.3 How popular is Go?](#)

[14.11.4 OpenAPI and Go](#)

[14.12 Exercises](#) 🍀

[15 REFERENCES](#)

1 PREFACE

Sat Nov 23 05:21:29 EST 2019 

1.1 LEARNING OBJECTIVES



Learning Objectives

- Learn about how we distribute material as ePub's.
- Learn how to create an ePub with our material from source.
- Introduce elementary notations we use in the ePub's.
- See who contributed to the ePub's.

1.2 EPUB READERS

This document is distributed in ePub format. Every OS has a suitable ePub reader to view the document. Such readers can also be integrated into a Web browser so that when you click on an ePub it is automatically opened in your browser. As we use ePubs the document can be scaled based on the user's preference. If you ever see a content that does not fit on a page we recommend you zoom out to make sure you can see the entire content.

We have made good experiences with the following readers:

- **macOSX:** [Books](#), which is a build in ebook reader
- **Windows 10:** [Microsoft edge](#), but it must be the newest version, as older versions have bugs. Alternatively, use [calibre](#)
- **Linux:** [calibre](#)

If you have an iPad or Tablet with enough memory, you may also be able to use them.

Sometimes you may want to adjust the zoom of your reader to increase or decrease it. Please adjust your zoom to a level that is comfortable for you. On macOS with a larger monitor, we found that zooming out multiple times results

in very good rendering allowing you to see the source code without horizontal scrolling.

1.3 CORRECTIONS

The material collected in this document is managed in

- <https://github.com/cloudmesh-community/book/chapters>

In case you see an error or like to make a contribution of your own section or chapter, you can do so in github via pull requests.

The easiest way to fix an error is to read the ePub and click on the cloud  symbol in a heading where you see the error. This will bring you to an editable document in github. You can directly fix the error in the web browser and create there a pull request. Naturally, you need to be signed into github before you can edit and create a pull request.

As a result contributors and authors will be integrated automatically next time we compile the material. Thus even if you corrected a single spelling error, you will be acknowledged.

1.4 CONTRIBUTORS

Contributors are sorted by the first letter of their combined Firstname and Lastname and if not available by their github ID. Please, note that the authors are identified through git logs in addition to some contributors added by hand. The git repository from which this document is derived contains more than the documents included in this document. Thus not everyone in this list may have directly contributed to this document. However if you find someone missing that has contributed (they may not have used this particular git) please let us know. We will add you. The contributors that we are aware of include:

Anand Sriramulu, Ankita Rajendra Alshi, Anthony Duer, Arnav, Averill Cate, Jr, Bertolt Sobolik, Bo Feng, Brad Pope, Brijesh, Dave DeMeulenaere, De'Angelo Rutledge, Eliyah Ben Zayin, Eric Bower, Fugang Wang, Geoffrey C. Fox, Gerald Manipon, Gregor von

Laszewski, Hyungro Lee, Ian Sims, IzoldaIU, Javier Diaz, Jeevan Reddy Rachepalli, Jonathan Branam, Juliette Zerick, Keith Hickman, Keli Fine, Kenneth Jones, Mallik Challa, Mani Kagita, Miao Jiang, Mihir Shanishchara, Min Chen, Murali Cheruvu, Orly Esteban, Pulasthi Supun, Pulasthi Supun Wickramasinghe, Pulkit Maloo, Qianqian Tang, Ravinder Lambadi, Richa Rastogi, Ritesh Tandon, Saber Sheybani, Sachith Withana, Sandeep Kumar Khandelwal, Sheri Sanders, Shivani Katukota, Silvia Karim, Swarnima H. Sowani, Tharak Vangalapat, Tim Whitson, Tyler Balson, Vafa Andalibi, Vibhatha Abeykoon, Vineet Barshikar, Yu Luo, ahilgenkamp, aralshi, azebrowski, bfeng, brandonfischer99, btpope, garbeandy, harshadpitkar, himanshu3jul, hrbahramian, isims1, janumudvari, joshish-iu, juaco77, karankotz, keithhickman08, kkp, mallik3006, manjunathsivan, niranda perera, qianqian tang, rajni-cs, rirasto, sahancha, shilpasingh21, swsachith, toshreyanjain, trawat87, tvangalapat, varunjoshi01, vineetb-gh, xianghang mi, zhengyili4321

1.5 NOTATION

The material here uses the following notation. This is especially helpful, if you contribute content, so we keep the content consistent.

if you like to see the details on how to create them in the markdown documents, you will have to look at the file source while clicking on the cloud in the heading of the Notation section ([Section 1.5](#)). This will bring you to the markdown tex, but you will still have to look at the [raw content](#) to see the details.

 OR  `![[Github]](images/github.png)`

If you click on the  or  in a heading, you can go directly to the > document in github that contains the next content. This is > convenient to fix errors or make additions to the content. The cloud will be automatically added upon inclusion of a new markdown file that includes in its first line a section header.

\$

Content in bash is marked with verbatim text and a dollar sign

```
$ This is a bash text
```

[1]

References are indicated with a number and are included in the > reference chapter [1]. Use it in markdown with > `[@las14cloudmeshmultiple]`. References must be added to the `references.bib` file in BibTeX format.

🚫 or ❌

Chapters marked with this emoji are not yet complete or have some issue that we know about. These chapters need to be fixed. If you like to help us fixing this section, please let us know. Use it in markdown with `:o2:` or if you like to use the image with `![[No]](images/no.png)`.

📺 [REST 36:02](#)

Example for a video with the `![[Video]](images/video.png)` emoji. Use it in markdown with `![[Video]](images/video.png) REST 36:02(https://youtu.be/xjFuA6q5N_U)`

📺 [Slides 10](#)

Example for slides with the `![[Presentation]](images/presentation.png)` emoji. These slides may or may not include audio.

📺 [Slides 10](#)

Slides without any audio. They may be faster to download. Use it in markdown with `![[Presentation]](images/presentation.png) Slides 10(TBD)`.



A set of learning objectives with the `![[Learning]](images/learning.png)` emoji.



A section is release when it is marked with this emoji in the syllabus.
Use it in markdown with `![[OK]](images/ok.png)`.



Indicates opportunities for contributions. Use it in markdown with `![[Question]](images/question.png)`.



Indicates sections that are worked on by contributors. Use it in markdown with `![[Construction]](images/construction.png)`.



Sections marked by the contributor with this emoji `![[Smiley]](images/smiley.png)` when they are ready to be reviewed.



Sections that need modifications are indicated with this emoji `![[Comment]](images/comment.png)`.



A warning that we need to look at in more detail `![[Warning]](images/warning.png)`



Notes are indicated with a bulb `![[Idea]](images/idea.png)`

Other emojis

Other emojis can be found at <https://gist.github.com/rxaviers/7360908>. However, note that emojis may not be viewable in other formats or on all platforms. We know that some emojis do not show in calibre, but they do show in macOS

iBooks and MS Edge

This is the list of emojis that can be converted to PDF. So if you like a PDF, please limit your emojis to

:cloud: ☁️ :o2: 🇦🇪 :relaxed: 😌 :sunny: ☀️ :baseball: ⚾️ :spades: ♠️ :hearts: ♥️ :clubs: ♣️ :diamonds: ♦️
:hotsprings: 🌋 :warning: ⚠️ :parking: 🅑 :a: 🅐 :b: 🅑 :recycle: ♻️ :copyright: © :registered: ® :tm: ™
:bangbang: !! :interrobang: !? :scissors: ✂️ :phone: 📞

1.5.1 Figures

Figures have a caption and can be referred to in the ePub simple with a number. We show such a reference pointer while referring to [Figure 1](#).



Figure 1: Figure example

Figures must be written in the md as

```
![[Figure example](images/code.png){#fig:code-example width=1in}
```

Note that the text must be in one line and must not be broken up even if it is longer than 80 characters. You can refer to them with `@fig:code-example`. Please note in order for numbering to work figure references must include the `#fig:` followed by a unique identifier. Please note that identifiers must be really unique and that identifiers such as `#fig:cloud` or similar simple identifiers are a poor choice and will likely not work. To check, please list all lines with an identifier such as.

```
$ grep -R "#fig:" chapters
```

and see if your identifier is truly unique.

1.5.2 Hyperlinks in the document

To create hyperlinks in the document other than images, we need to use proper markdown syntax in the source. This is achieved with a reference for example in

sections headers. Let us discuss the reference header for this section, e.g. Notation. We have augmented the section header as follows:

```
# Notation {#sec:notation}
```

Now we can use the reference in the text as follows:

```
In @sec:notation we explain ...
```

It will be rendered as: In [Section 1.5](#) we explain ...

1.5.3 Equations

Equations can be written as

```
$$a^2+b^2=c^2$$ {#eq:pythagoras}
```

and used in text:

$$a^2 + b^2 = c^2 \quad (1)$$

It will render as: As we see in [Equation 1](#).

The equation number is optional. Inline equations just use one dollar sign and do not need an equation number:

```
This is the Pythagoras theorem: $a^2+b^2=c^2$
```

Which renders as:

This is the Pythagoras theorem: $a^2 + b^2 = c^2$.

1.5.4 Tables

Tables can be placed in text as follows:

```
: Sample Data Table {#tbl:sample-table}
```

```
x   y   z
---  ---
1   2   3
4   5  42
```

As usual make sure the label is unique. When compiling it will result in an error if labels are not unique. Additionally there are several md table generators

available on the internet and make creating table more efficient.

1.6 UPDATES

As all documents are managed in github, the list of updates is documented in the commit history at

- <https://github.com/cloudmesh-community/book/commits/master>

In case you do a lecture withus we recommend that you download a new version oof the ePub every week. This way you are typically staying up to date. You can check the commit history and identify if the version of the ePub is older than the committed version, if so we recommend that you download a new version.

 *We typically will not make announcements to the class as the GitHub commit history is sufficient and you are responsible to monitor it as part of your class activities.*

2 OVERVIEW



Learning Objectives

- Gain an overview what currently is in this book
 - Review the high level goals
 - Be aware that this book is not complete and is worked on as we speak
 - Be aware to check out the book on a weekly basis to stay up to date
 - Be aware that additional material is distributed in separate books such as Linux, Python, and Writing in Markdown.
 - Be aware that books you may purchase may already be outdated by the time you order them.
-
-

In this book we provide a number of chapters that will allow you to easily get knowledge in cloud computing on theoretical and practical levels.

Although the following was originally covered in this book, we decided to split out its contents as to make the core cloud engineering book smaller. In case you take one of our classes using the book, we expect that you pick up the material covered also by these additional books. Please be aware that some of the class material is based on Python and Linux. You will need no knowledge of them as you can pick it up while reading this book.

- [Cloud Computing](#)
- [Linux for Cloud Computing](#)
- [Python for Cloud Computing](#)
- [Scientific Writing with Markdown](#)

The book is organized as follows:

Definition of Cloud Computing

We will start with the definition of what cloud computing is and motivate why it is important to not only know technologies such as AI or ML or

Databases. We present you with evidence that Clouds are absolutely relevant to today's technologies. We see furthermore a trend to utilize AI and ML services on in the cloud. Technologies such as virtual machine and containers and Function as a Service are essential to the repertoire of a modern Cloud or Data engineer. There is more than ML ... ☺

Data Center

This chapter will explain you why we need cloud data centers, how a cloud data center look likes and which environmental impact such data centers have.

Architecture

This chapter will introduce you to the basic architectural features and designs of cloud computing. We will discuss architectures for IaaS, and contrast it to other architectures. We will discuss the NIST definition of the cloud and the Cloud Security Alliance Reference Architecture. We will discuss the multi-cloud architecture introduced by cloudmesh as well as the Big Data Reference Architecture.

REST

This chapter will introduce you to a way on how to define services in the cloud that you can easily access via language independent client APIs. It will introduce you to the fundamental concepts of REST. We will more importantly introduce you to OpenAPI that allows you to specify REST services via a specification document so you can create APIs and clients form the document automatically. We will showcase you how to do that with `flask`.

We will showcase you on a very popular service such as GitHub how to easily interface with REST services in Python.

GraphQL

In this chapter we will introduce you to GraphQL which allows you to access data through a query language. It allows clients to easily formulate queries that retrieve desired data. Restrictions to the queries can be

formulated to download what is needed. Other features include a type system. Github has added in addition to its REST service also a GraphQL interface. You will have the opportunity to explore GraphQL while interfacing with GitHub.

Hypervisors

Virtualization is one of the important technologies that started the cloud revolution. It provides the basic underlying principles for the development and adoption of clouds. The concept, although old and already used in the early days of computing, has recently been exploited to lead to better utilization of servers as part of data centers, but also the local desktops.

In this chapter we introduce you to the basic concepts and distinguish the various forms of virtualization.

We list virtualization frameworks such as Libvirt, Qemu, KVM, Xen, and Hyper-V. Dependent on your hardware you will be encouraged to experiment with one or more of them.

IaaS

In the IaaS chapter we will be reviewing many of the services offered by providers such as AWS, Azure, Google, and OpenStack that is used by some academic clouds such as chameleon cloud.

In addition we will introduce you to elementary command line tools and programs to access this infrastructure.

In this section we will also provide you with information about multicloud management with cloudmesh which makes it extremely easy to switch between and use services from multiple clouds.

Important to note is that the appendix contains very useful information that augments this section. This includes a more detailed list of services for some IaaS providers as well as information on how to use chameleon cloud which has been adapted by us for this chapter.

Map/Reduce

In this chapter we discuss about the background of Mapreduce along with Hadoop and it's core components. We will also introduce Spark to you in this section to Spark. you in this section.

You will be presented on how you can use the systems on a single resource so you can explore them more easily, but we will also let you know how to install them on a cluster in principal.

We conclude this section with some important Map/Reduce frameworks used as part of the larger Map/Reduce ecosystem such as AWS Elastic Map/Reduce (AWS EMR). This also includes a discussion about Twister2 which is a version of Map/Reduce that could perform even faster then Spark.

🕒 In fact we have here two sections that need to be delineated a bit better which we hope we can do with your help.

Container

In the container chapter we will introduce you to the basic concepts of a container and delineate it from virtual machines as we have introduced you earlier. We will start the chapter with an introduction to Docker and than introduce you how to manage clusters capable of running many containers with the help of docker swarm and kubernetes. To showcase you its use on other PaaS and applications we even show you how to run Hadoop with docker as well as how to conduct a pagerank analysis. Kubernetes will be discussed in its own section.

As many academic datacenters do run queuing system, we will also showcase Singularity allowing you to use containers within a batch queuing system.

🕒 you will help us improving this section if you elect to conduct a project on comet.

We conclude the section with letting you know how to run Tensorflow via singularity,

Serverless Computing

Recently a new paradigm in cloud computing has been introduced. Instead of using virtual machines or containers functions with limited resource requirements are specified that can then be executed on function capable execution services hosted by cloud providers.

We will introduce you to this concept and showcase you some examples of FaaS services and frameworks.

Messaging Services

Many devices in the cloud need to communicate with each other. In this chapter we look into how we can provide alternatives to REST services that provide messaging capabilities. We will focus on MQTT which is often used to connect cloud edge devices between each other and the cloud.

GO

Go is a programming language used by Google and has been most famously used to implement Kubernetes. In this chapter we introduce you to the elementary features of Go and also take a closer look on how we can define REST services, use OpenAPI, and interface with clouds.

Cloud AI Services

As part of the class we will be exploring AI services that are hosted in cloud and offered as service. If interested you will be able to use them in your projects. As part of this class you will also be developing AI services and those can be hosted in the cloud and reused by others. While using cross-platform specifications, clients for Java, Python, Scala, Go, and other programming languages will be automatically created for you. This will allow others to reuse your services.

3 DEFINITION OF CLOUD COMPUTING



Learning Objectives

- Compare different definitions of cloud computing.
- Review the History of cloud computing.
- Identify trends.
- The current hot job is *data engineer* which is sought after more than data scientists (a new trend). You have chosen the right course ☺
- Be TALLL to be successful in cloud computing.

Videos:

-  [Definition of Cloud Computing 2019](#)
-

3.1 DEFINING THE TERM CLOUD COMPUTING

In this presentation we review three definitions of cloud computing. This includes the definitions by

- NIST
- Wikipedia
- Gartner

3.2 HISTORY AND TRENDS

We review some of the historical aspects that lead to cloud computing and especially look into more recent trends. These trends motivate that we need to look at enhancements to the traditional Service Model that include Infrastructure-, Platform- and Software- as a Service. These enhancements especially are targeting Function-, and Container as a Service.

3.3 JOB AS A CLOUD/DATA ENGINEER

We look at some job related trends that especially focus on the newest hot job description called **Data Engineer**. It is motivated that current job offerings as data engineer is 13% versus 1% for data scientists. As this class is targeted towards bringing the engineering component towards the data scientists, computer scientists, and application developer, This class is ideally suited for increasing your marketability.

3.4 YOU MUST BE THAT TALLL

We close this class with Gregor's TALLL principle to succeed in Cloud Computing:

You must be that TALLL to survive in Cloud Computing and Big Data

This principle includes the following characteristics

Trend Awareness (TA)

We need to be aware not only what is currently a trend, but what will be future trends

Longevity Planning (L)

We need to be able to reproduce our services and results (e.g. can we reproduce them still in six month).

Leap Detection (L)

We need to be able to deal with technology Leaps

Learning Willingness (L)

We need to constantly learn to keep up as technology changes every 6 month

Naturally this principal can be applied to other disciplines.

4 DATACENTER

4.1 DATA CENTER



Learning Objectives

- What is a data center.
 - What are import metrics.
 - What is the difference between a Cloud data center and a traditional datacenter.
 - What are examples of Cloud data centers.
-
-

4.1.1 Motivation: Data

Before we go into more details of a data center we like to motivate why we need them. Here we start with looking at the amount of data that recently got created and provide one of many motivational aspects. Not all data will or should be stored in data centers. However a significant amount of data will be in such centers.

4.1.1.1 How much data?

One of the issues we have is to comprehend how much data is created. It's hard to imagine and put into a perspective how much total data is created over a year, a month, a week, a day or even just an hour. Instead to easily visualize the amount of data produced we often find graphics easier to comprehend that shows how much data was generated in a minute. Such depictions usually include examples of data generated as a part of popular cloud services or the internet in general.

One such popular depiction is *Data Never Sleeps* (see [Figure 3](#)). It has been produced a number of times over the years and is now at version 7.0 released in 2019. If you identify a newer version, please let us know.

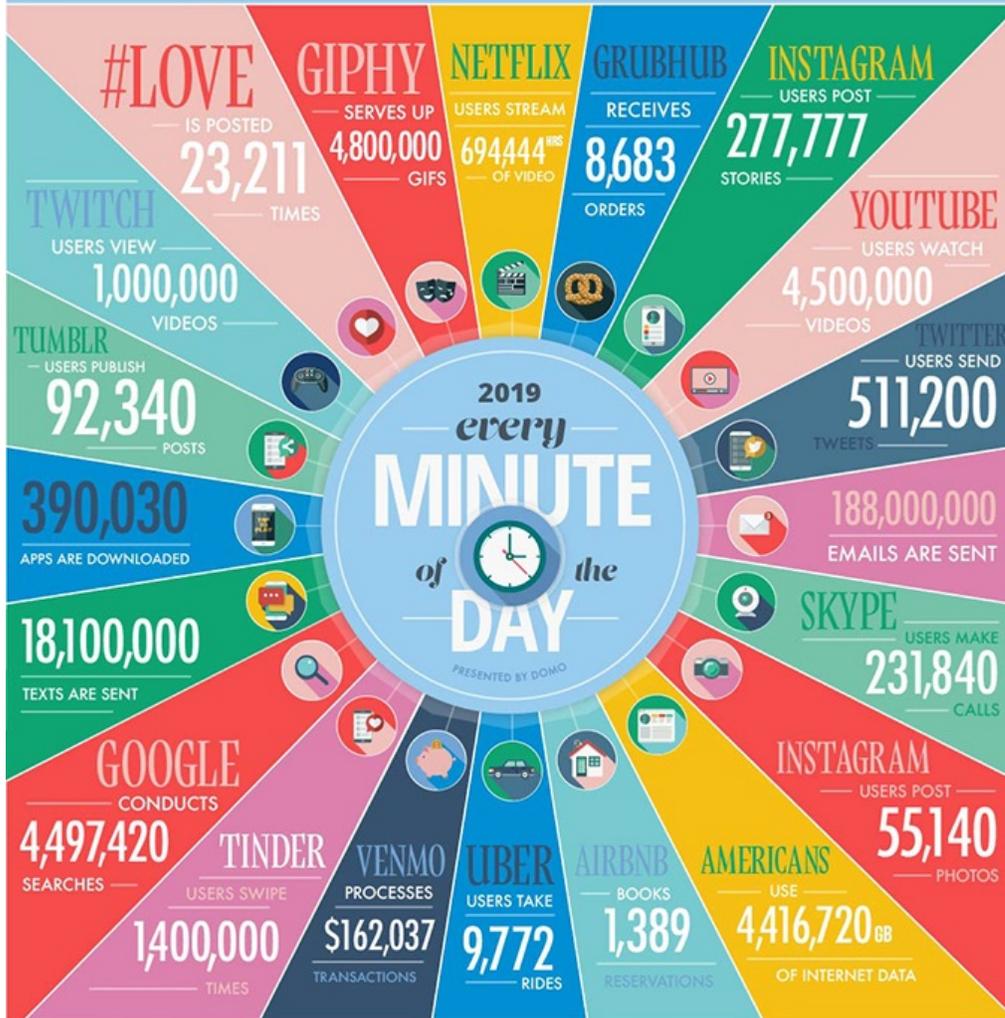
Observations for 2019: It is worth while to study this image in detail and identify some of the data that you can relate to of service you use. It is also a possible indication to study other services that are mentioned. For the data for 2019 we observe that a staggering ~4.5Mil google searches are executed every minute which is slightly lower than the number of videos watched on youtube. 18Mil text messages are send every minute. Naturally the numbers are averages over time.



DATA NEVER SLEEPS 7.0

How much data is generated *every minute*?

There's no way around it: big data just keeps getting bigger. The numbers are staggering, and they're not slowing down. By 2020, there will be 40x more bytes of data than there are stars in the observable universe. In our 7th edition of Data Never Sleeps, we bring you the latest stats on how much data is being created in every digital minute — and the numbers are staggering.



The world's internet population is growing significantly year-over-year. As of January 2019, the internet reaches 56.1% of the world's population and now represents 4.39 billion people — a 9% increase from January 2018.



GLOBAL INTERNET POPULATION GROWTH 2012-2018 (IN BILLIONS)

The ability to make data-driven decisions is crucial to any business. With each click, swipe, share, and like, a world of valuable information is created. Domo puts the power to make those decisions right into the palm of your hand by connecting your data and your people at any moment, on any device, so they can make the kind of decisions that make an impact.

Learn more at domo.com

SOURCES: STATISTA, INTERNET LIVE STATS, EXPANDED RAMBLINGS, NATIONAL ASSOCIATION OF CITY TRANSPORTATION OFFICIALS, WIRED



Figure 2: Data Never Sleeps [\[2\]](#)

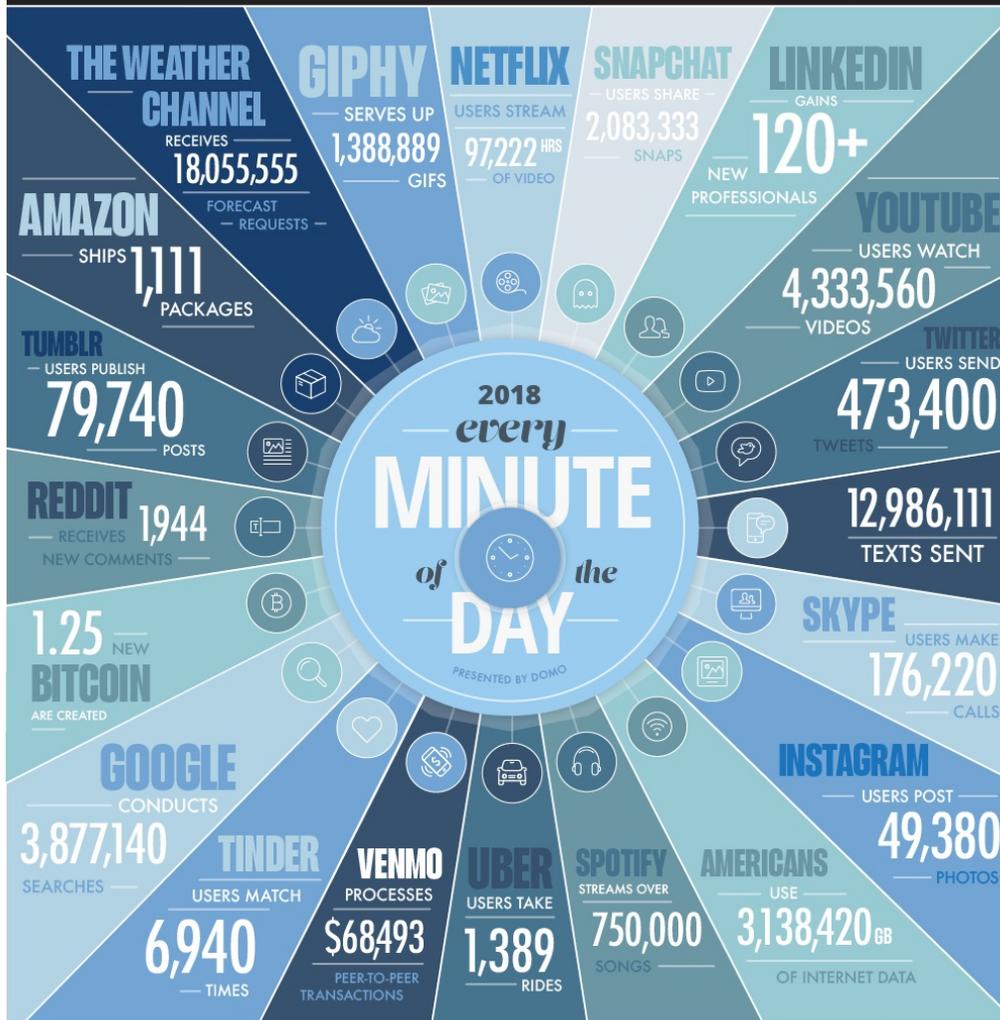
In contrast in 2017 we observed: A 3.8Mil google searches are executed every minute. Surprisingly the weather channel receives over 18Mil forecast requests which is even higher than the 12Mil text messages send every minute. Youtube certainly serving a significant number of users by 4.3Mil videos watched every minute.



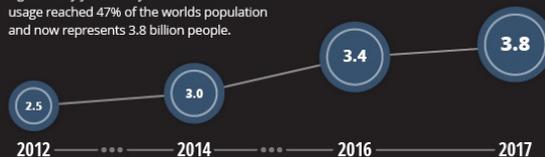
DATA NEVER SLEEPS 6.0

How much data is generated *every minute*?

There's no way around it: big data just keeps getting bigger. The numbers are staggering, but they're not slowing down. By 2020, it's estimated that for every person on earth, 1.7 MB of data will be created every second. In our 6th edition of Data Never Sleeps, we once again take a look at how much data is being created all around us every single minute of the day—and we have a feeling things are just getting started.



The world's internet population is growing significantly year-over-year. In 2017, internet usage reached 47% of the world's population and now represents 3.8 billion people.



GLOBAL INTERNET POPULATION GROWTH 2012-2017 (IN BILLIONS)

The ability to make data-driven decisions is crucial to any business. With each click, swipe, share, and like, a world of valuable information is created. Domo puts the power to make those decisions right into the palm of your hand by connecting your data and your people at any moment, on any device, so they can make the kind of decisions that make an impact.

Learn more at domo.com

SOURCES: STATISTA, LINKEDIN, INTERNET LIVE STATS, EXPANDED RAMBLINGS, SLASH FILM, RIAA, BUSINESS OF APPS, INTERNATIONAL TELECOMMUNICATIONS UNION, INTERNATIONAL DATA CORPORATION



Figure 3: Data Never Sleeps [3]

A different source publishes what is happening on the internet in a minute, but we have been able to locate a version from 2018 (see [Figure 4](#)). While some data seems the same, others are slightly different. For example this graph has a lower count for Google searches, while the number of text messages send is significantly higher in contrast to [Figure 3](#).

2018 *This Is What Happens In An Internet Minute*



Figure 4: Internet Minute 2018 [4]

While reviewing the image from last year from the same author, we find not only increases, but also declines. Looking at facebook showcases a loss of 73000 logins per minute. This loss is substantial. We can see that facebook services are replaced by other services that are more popular with the younger generation who tend to pick up new services quickly (see [Figure 5](#)).

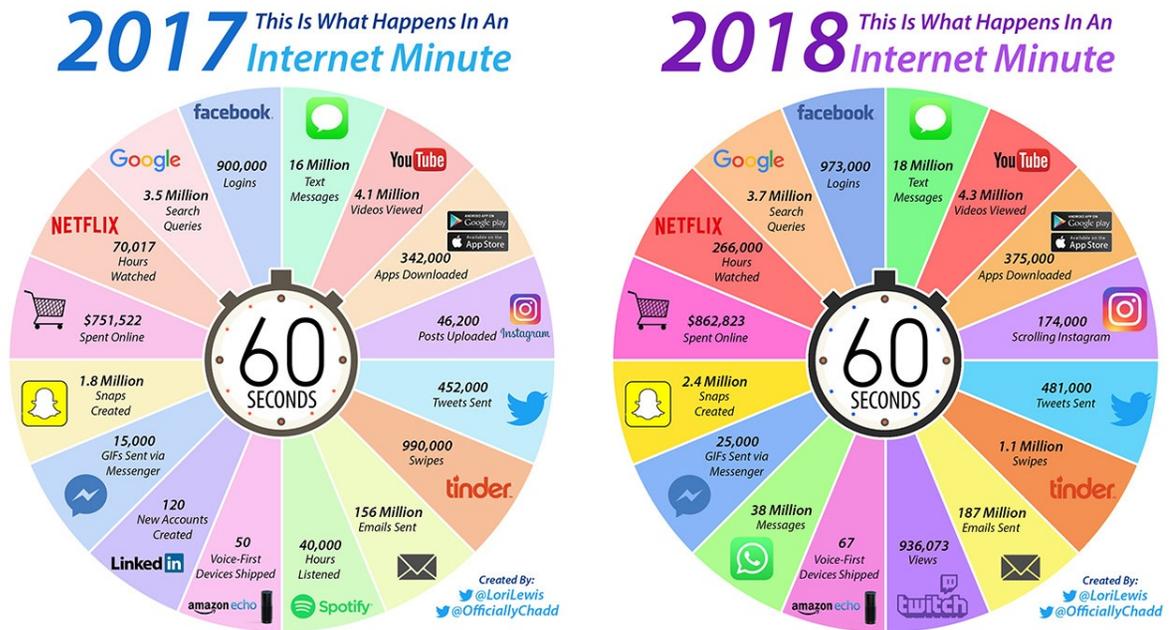


Figure 5: Internet Minute 2017-2018 [4]

It is also interesting to compare such trends over a longer period of time (see [Figure 6](#), [Figure 7](#)). An example is provided by looking at Google searches

- <http://www.internetlivestats.com/google-search-statistics/>.

and visualized in [Figure 6](#).

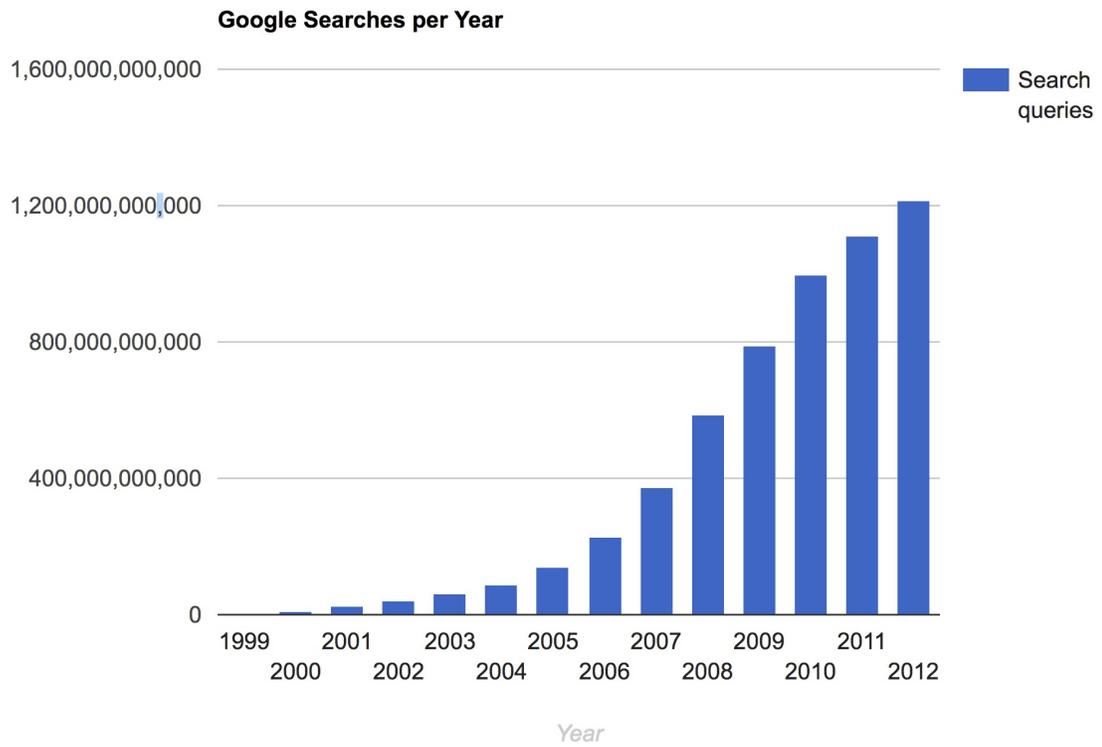


Figure 6: Google searches over time

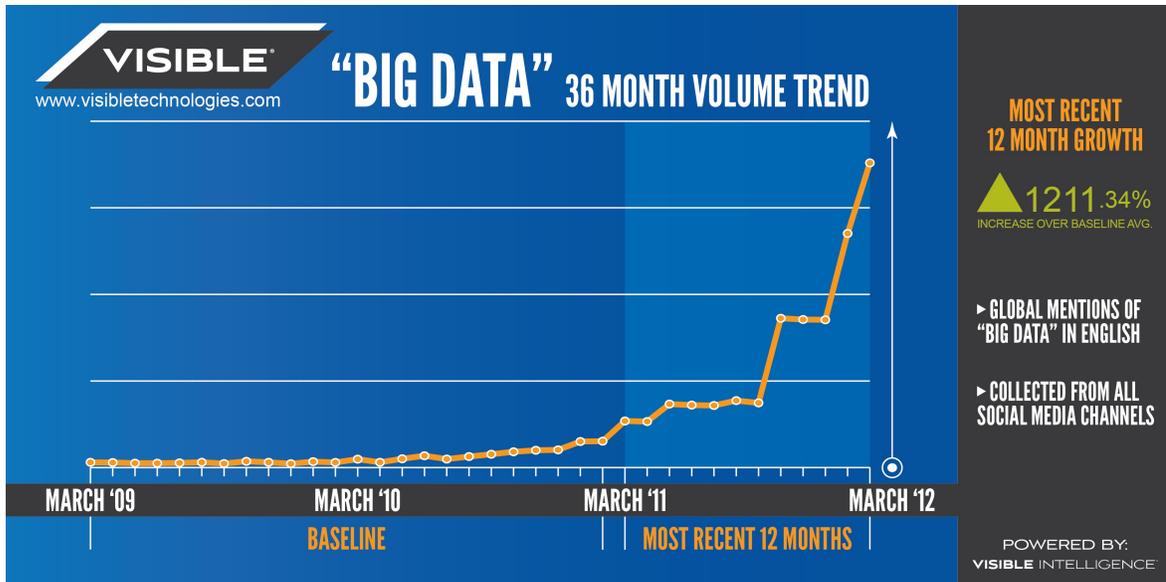


Figure 7: Big data trend. 2012 [5]

When looking at the trends, many predict an exponential growth in data. This trend is continuing.

4.1.2 Cloud Data Centers

A *data center* is a facility that hosts the information technology related to servers and data serving a large number of customers. data centers evolved from the need to originally have large rooms as the original computers filled in the early days of the compute revolution filled rooms. Once multiple computers were added to such facilities super computer centers created for research purposes. With the introduction of the internet and offering services such as Web hosting large business oriented server rooms were created. The need for increased facilities was even accelerated by the development of virtualization and servers being rented to customers in shared facilities. As the need of web hosting still is important but has been taken over by cloud data centers, the terms internet data center, and cloud data center are no longer used to distinguish it. Instead we use today just the term *data center*. There may be still an important difference between research data centers offered in academia and industry that focus on providing computationally potent clusters focus on numerical computation. Such data centers are typically centered around the governance around a smaller number of users that are either part of an organization or a virtual organization. However, we see that even in the research community data centers not only host

supercomputers, but also Web server infrastructure and these days even private clouds that support the organizational users. In case of the latter we speak about supporting the *long tail about science*.

The latter is driven by the 80%-20% rule. E.g. 20% of the users use 80% of the compute power. This means that the top 20% of scientists are served by the leadership class super computers in the nation, while the rest are either served by other servers, cloud offerings through research and public clouds.

4.1.3 Data Center Infrastructure

Due to the data and the server needs in the cloud and in research such data centers may look very different. Some focus on large scale computational resources, some on commodity hardware offered to the community. The size of them is also very different. While a supercomputing center as part of a university was one of the largest such data centers two decades ago, they dwarf the centers now deployed by industry to serve the long tail of customers.

In general a data center will have the following components:

- Facility: the entire data center will be hosted in a building. The building may have specific requirements related to security, environmental concerns, or even the integration into the local community with for example providing heat to surrounding residences.
- Support infrastructure: This building will include a significant number of support infrastructure that addresses for example continuous power supply, air conditioning, and security For this reason you find in such centers
 - Uninterruptible Power Sources (UPS)
 - Environmental Control Units
 - Physical Security Systems
- Information Technology Equipment: Naturally the facility will host the IT equipment including the following:
 - Servers
 - Network Services

- Disks
- Data Backup Services
- Operations staff: The facility will need to be staffed with the various groups that support such data centers. It includes
 - IT Staff
 - Security and Facility Staff
 - Support Infrastructure Staff

With regards to the number of people serving such a facility it is obvious that through automation is quite low. According to [6] proper data center staffing is a key to a reliable operation (see [Figure 8](#)).

According to [Figure 8](#) operational sustainability contains three elements of operational sustainability, namely management and operations, building characteristics, and site location [6].

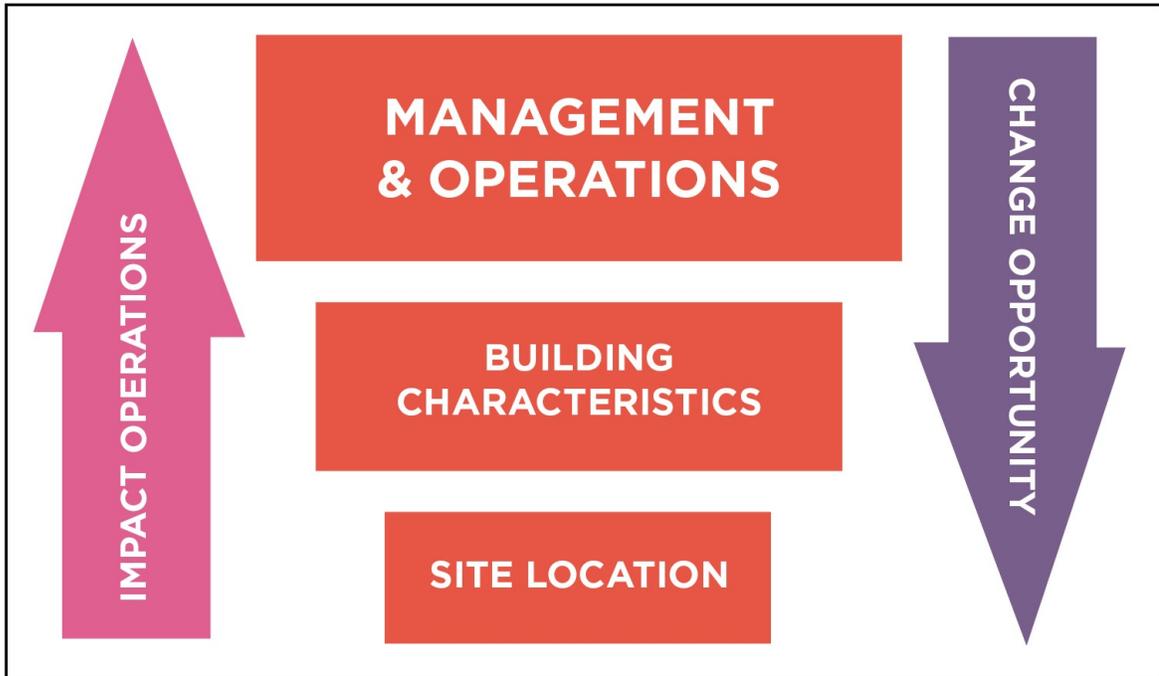


Figure 8: Datacenter Staff Impact [6]

Another interesting observation is the root cause of incidents in a data center. Everyone has probably experienced some outage, so it is important to identify where they come from in order to prevent them. As we see in [Figure 9](#) not every error is caused by an operational issue. External, installation, design, and manufacturer issues are together the largest issue for datacenter incidents (see [Figure 9](#)). Figure Outage. According to the Uptime Institute Abnormal Incident Reports (AIRs) database, the root cause of 39% of data center incidents falls into the operational area [6].

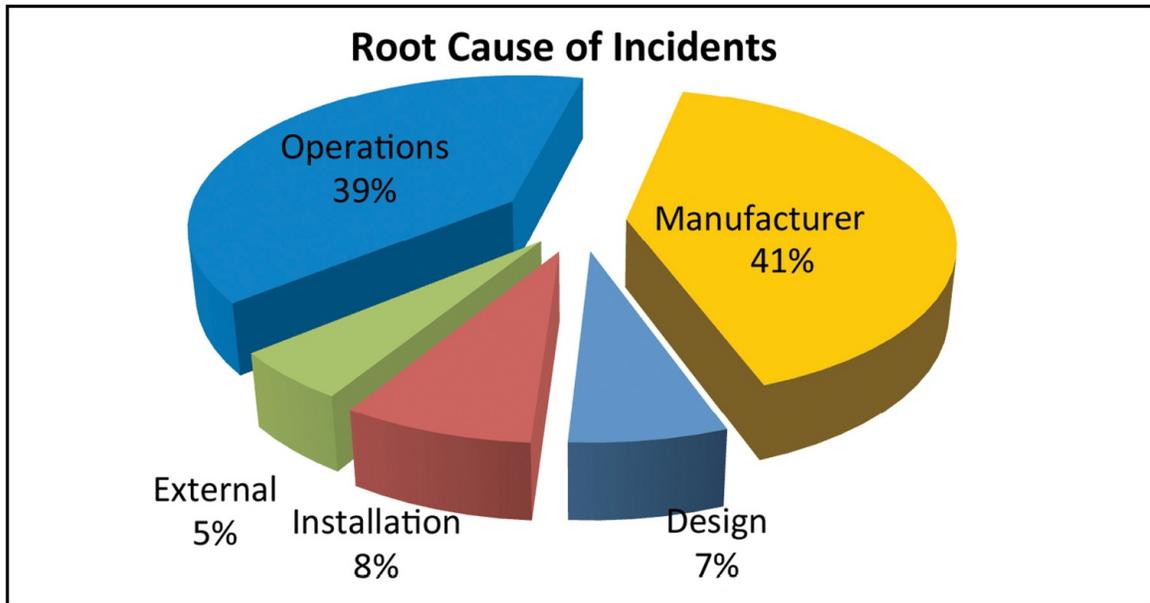


Figure 9: Datacenter outage [6]

4.1.4 Data Center Characteristics

Next we identify a number of characteristics when looking at different data centers.

- **Variation in Size:** Data centers range in size from small *edge* facilities to megascale or hyperscale filling large ware houses.
- **Variation in cost per server:** Although many data centers standardize their components, specialized services may be offered not on a 1K server, but on a 50K server.
- **Variation in Infrastructure:** Servers in centers serve a variation of needs and motivate different infrastructure: Use cases, Web Server, E-mail, Machine Learning, Pleasantly Parallel problem, traditional super computing jobs.
- **Energy Cost:** Data centers use a lot of energy. The energy cost varies per region. A motivation to reduce energy use and cost is also been trended by environmental awareness, not only by the operators, but by the community in which such centers operate.

- **Reliability:** Although through operational efforts the data center can be made more reliable, failure still can happen. Examples are
- <https://www.zdnet.com/article/microsoft-south-central-u-s-datacenter-outage-takes-down-a-number-of-cloud-services/>
- <https://www.datacenterknowledge.com/archives/2011/08/07/lightning-in-dublin-knocks-amazon-microsoft-data-centers-offline>
- <https://techcrunch.com/2012/10/29/hurricane-sandy-attacks-the-web-gawker-buzzfeed-and-huffington-post-are-down/>

Hence Data Center IaaS advantages include

- Reduced operational cost
- Increased reliability
- Increased scalability
- Increased flexibility
- Increased support
- Rapid deployment
- Decrease management: Outsourcing expertise that is not related to core business

Datacenter disadvantages include

- Loss of control of the HW
- Loss of control of the data
- Model is preferring many users
- Software to control infrastructure is not accessible
- Variations in performance due to sharing
- Integration requires effort beyond login
- Failures can have a humongous impact

4.1.5 Data Center Metrics

One of the most important factor to ensure smooth operation and offering of services is to employ metrics that will be able to provide significant impacting the operations. Having metrics allows the staff to monitor and adapt to dynamic situations but also to plan operations.

4.1.5.1 Data Center Energy Costs

One of the easiest to monitor metrics for a datacenter is the cost of energy used to operate all of the equipment. Energy is one of the largest costs a datacenter incurs during its operation as all of the servers, networking, and cooling equipment require power 24/7. For electricity, billing is usually measured in terms of kilowatt hours (kWh) and kilowatts (kW). Depending on circumstances, there may also be costs for public purpose programs, cost recovery, and stranded costs, but they are beyond the scope of this book.

To provide a quick understanding, it is best to understand the relation between kilowatt hours and kilowatts. kWh is typically referred to as consumption while kW is referred to as demand and it's important to understand how these two concepts relate to each other. The easiest analogy to describe the relationship is to think of kilowatts (demand) as the size of a water pipe while kilowatt-hours (consumption) is how much water has passed through the pipe. If a server requires 1.2 kW to operate then, after an hour has passed, it will have consumed 1.2 kWh. However, if the server operates at 1.2 kW for 30 minutes and then goes idle and drops to 0.3 kW for another 30 minutes, then total power consumed will be:

$$kWh = 0.3 * 30 / 60 + 1.2 * 30 / 60 = 0.75 \quad (2)$$

Energy costs for a datacenter, then, are composed of two things: charges for energy and charges for demand. Energy is the amount of total energy consumed by the datacenter and will be the total kWh multiplied by the cost per kWh. Demand is somewhat more complicated: it is the highest total consumption measured in a 15 minute period. Taking the previous example, if a datacenter has 1,000 servers, the total energy consumption would be 750 kWh in the hour, but the demand charge would be based off of 1,200 kW (or 1.2 MW).

The costs, then, are how the utility company recoups its expenses: the charge per kWh is it recouping the generation cost while the kW charge is recouping the cost of transmission and distribution (T&D). Typically, the demand charge is much higher and will depend on utility constraints - if a utility is challenged on the T&D front, expect these costs to be over \$6-\$10/kW. If the assumed cost-per-kWh is \$0.12 and cost-per-kW is \$8, the cost to run our servers for a month would be:

$$kWh = 0.75 * 24 * 30 * 0.12 * 1000 = 64,800 \quad (3)$$

$$kW = 1.2 * 8 * 1000 = 9,600 \quad (4)$$

This would total to \$74,400. It's important to note that fixing demand charges can have a tremendous payback: had the servers simply consumed 750 kW over the course of the hour, then our demand charges would've been halved to \$4,800 while the energy costs remained the same. This is also why server virtualization can have a positive impact on energy costs: by having fewer servers running at a higher utilization, the demand charge will tend to level itself out as, on average, each server will be more fully utilized. For example, it's better to pay for 500 servers at 100% utilization than 1000 servers at 50% utilization even though the amount of work done is the same since, if the 1,000 servers momentarily all operate at 100% utilization for even a brief amount of time in a month, the demand charge for the datacenter will be much higher.

4.1.5.2 Data Center Carbon Footprint

Scientists world wide have identified a link between carbon emission and global warming. As the energy consumption of a data center is substantial, it is prudent to estimate the overall carbon emission. Schneider Electric (formerly APC) has provided a report on how to estimate the Carbon footprint of a data center [7]. Although this report is already a bit older, it provides still valuable information. It defines key terms such as

Carbon dioxide emissions coefficient (*carbon footprint*):

- With the increasing demand of data, bandwidth and high performance systems, there is substantial amount of power consumption. This leads to high amount of greenhouses gases emission into the atmosphere, released due to any kind of basic activities like driving a vehicle or running a power plant.

“The measurement includes power generation plus transmission and distribution losses incurred during delivery of the electricity to its point of use.”

Data centers in total used 91 billion kilowatt-hours (kWh) of electrical

energy in 2013, and they will use 139 billion kWh by 2020. Currently, data centers consume up to 3 percent of all global electricity production while producing 200 million metric tons of carbon dioxide. Since world is moving towards cloud, causing more and more data center capacity leading more to power consumption.

Peaker plant:

- Peaking power plants, also known as peaker plants, and occasionally just *peakers*, are power plants that generally run only when there is a high demand, known as peak demand, for electricity. Because they supply power only occasionally, the power supplied commands a much higher price per kilowatt hour than base load power. Peak load power plants are dispatched in combination with base load power plants, which supply a dependable and consistent amount of electricity, to meet the minimum demand. These plants are generally coal-fired which causes a huge amount of CO₂ emissions. A peaker plant may operate many hours a day, or it may operate only a few hours per year, depending on the condition of the region's electrical grid. Because of the cost of building an efficient power plant, if a peaker plant is only going to be run for a short or highly variable time, it does not make economic sense to make it as efficient as a base load power plant. In addition, the equipment and fuels used in base load plants are often unsuitable for use in peaker plants because the fluctuating conditions would severely strain the equipment. For these reasons, nuclear, geothermal, waste-to-energy, coal and biomass are rarely, if ever, operated as peaker plants.

Avoided emissions:

- Emissions avoidance is the most effective carbon management strategy over a multi-decadal timescale to achieve atmospheric CO₂ stabilization and a subsequent decline. This prevents, in the first place, stable underground carbon deposits from entering either the atmosphere or less stable carbon pools on land and in the oceans.

Carbon offsets based on energy efficiency rely on technical efficiencies to reduce energy consumption and therefore reduce CO₂ emissions. Such improvements are often achieved by introducing more energy efficient

lightening, cooking, heating and cooling systems. These are real emission reduction strategies and have created valid offset projects.

This type of carbon offset provides perhaps the simplest options that will ease the adoption of low carbon practice. When these practices become generally accepted (or compulsory), they will no longer qualify as offsets and further efficiencies will need to be promoted.

CO2 (carbon dioxide, or *carbon*):

- Carbon dioxide is the main cause of the greenhouse effect, it is emitted in huge amount into our atmosphere with a life cycle of almost 100 years. Data centers emit during the manufacturing process of all the components that populate a data center (servers, UPS, building shell, cooling, etc.) and during operation of data centers (in terms of electricity consumed), the maintenance of the data centers (i.e. replacement of consumables like batteries, capacitors, etc.), and the disposal of the components of the data centers at the end of the lifecycle. Until now, power plants have been allowed to dump unlimited amounts of carbon pollution into the atmosphere - no rules were in effect that limited their emissions of carbon dioxide, the primary driver of global warming. Now, for the first time, the EPA has finalized new rules, or standards, that will reduce carbon emissions from power plants. Known as the Clean Power Plan, these historic standards represent the most significant opportunity in years to help curb the growing consequences of climate change.

The data center will have a total carbon profile, that includes the many different aspects of a data center contributing to carbon emissions. This includes manufacturing, packaging, transportation, storage, operation of the data center, and decommissioning. Thus it is important to notice that we not only need to consider the operation but also the construction and decommission phases.

4.1.5.3 Data Center Operational Impact

One of the main operational impacts is the cost and emissions of a data center cause by running, and cooling the servers in the data center. Naturally this is dependent on the type of fuel that is used to produce the energy. The actual

carbon impact using electricity certainly depends on the type of powerplant that is used to provide it. These energy costs and distribution of where the energy comes from can often be looked up by geographical regions on the internet or from the local energy provider. Municipal government organizations may also have such information. Tools such as the [Indiana State Profile and Energy Use \[8\]](#).

may provide valuable information to derive such estimates. Correlating a data center with cheap energy is a key factor. To estimate both costs in terms of price and carbon emission Schneider provides a convenient Carbon estimate calculator based on energy consumption.

- <https://www.schneider-electric.com/en/work/solutions/system/s1/data-center-and-network-systems/trade-off-tools/data-center-carbon-footprint-comparison-calculator/tool.html>
- <http://it-resource.schneider-electric.com/digital-tools/calculator-data-center-carbon>

If we calculate the total cost, we need naturally add all costs arising from build and teardown phase as well as operational upgrades.

4.1.5.4 Power Usage Effectiveness

One of the frequent measurements in data centers that is used is the Power usage effectiveness or PUE in short. It is a measurement to identify how much energy is used for the computing equipment versus other energy costs such as air conditioning.

Formally we define it as

PUE is the ratio of total amount of energy used by a computer data center facility to the energy delivered to computing equipment.

PUE was published in 2016 as a global standard under [ISO/IEC 30134-2:2016](#).

The inverse of PUE is the data center infrastructure efficiency (DCIE).

The best value of PUE is 1.0. Any data center must be higher than this value as

offices and other cost surely will arise when we look at the formula

$$\text{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}}$$

$$\text{PUE} = 1 + \frac{\text{Non IT Facility Energy}}{\text{IT Equipment Energy}}$$

According to the PUE calculator at

- <https://www.42u.com/measurement/pue-dcie.htm>

The following ratings are given

PUE	DCIS	Level of Efficiency
3.0	33%	Very Inefficient
2.5	40%	Inefficient
2.0	50%	Average
1.5	67%	Efficient
1.2	83%	Very Efficient

PUE is a very popular metric as it is relatively easy to calculate and provides a metric that can easily compare data centers between each other.

This metric comes also with some drawbacks:

- It does not integrate for example climate based differences, such as that the energy use to cool a data center in colder climates is less than in warmer climates. However, this may actually be a good side-effect as this will likely result in less cooling needs and therefore energy costs.
- It also forces large data centers with many shared servers in contrast to small data centers where operational cost may become relevant.
- It does not take in consideration recycled energy to for example heat other buildings outside of the data center.

Hence it is prudent not to just look at the PUE but also at other metrics that lead to the overall cost and energy usage of the total ecosystem the data center is located in.

Already in 2006, Google reported its six data centers efficiency as 1.21 and Microsoft as 1.22 which at that time were considered very efficient. However over time these target has shifted and today's data centers achieve much lower values. The Green IT Cube in Darmstadt, Germany even reported 1.082. According to Wikipedia an unnamed Fortune 500 company achieved with 30000 SuperMicro blades a PUE of 1.06 in 2017.

Exercises

E.PUE.1: Lowest PUE you can find

What is the lowest PUE you can find. Provide details about the system as well as the date when the PUE was reported.

4.1.5.5 Hot-Cold Aisle

To understand hot-cold aisles, one must take a brief foray into the realm of physics and energy. Specifically, understanding how a temperature gradient tries to equalize. The most important formula to know is the heat transfer [Equation 5](#).

$$q = h_c A (t_a - t_s) \quad (5)$$

Here, q is the amount of heat transferred for a given amount of time. For this example, we will calculate it as W/hour as that is, conveniently, how energy is billed. Air moving at a moderate speed will transfer approximately 8.47 Watts per Square Foot per Hour. A 1U server is 19 inches wide and about 34 inches deep. Multiplying the two values gives us a cross section of 646 square inches, or 4.48 square feet. Plugging these values into our [Equation 5](#) us:

$$q = 8.47 * 4.48 * (t_a - t_s) \quad (6)$$

This begins to point us towards why hot-cold aisles are important. If we introduce cold air from the AC system into the same aisle that the servers are exhausting into, the air will mix and begin to average out. For example, if our servers are producing exhaust at 100F and our AC unit provides 65F at the same rate, then the average air temperature will become 82.5F (assuming balanced air pressure). This has a deleterious effect on our server cooling - warmer air takes heat away from warmer surfaces slower than cooler air:

$$1,328.2 = 8.47 * 4.48 *(100 - 65)$$

$$664.0 = 8.47 * 4.48 *(100 - 82.5))$$

From the previous listing, we can see that a 35 degree delta allows the center to dissipate 1,300 Watts of waste heat from a 1U server while a 17.5 degree delta allows us to only dissipate 664 Watts of energy. If a server is consuming more than 664 Watts, it'll continue to get warmer and warmer until it eventually reaches a temperature differential high enough to create an equilibrium (or reaches a thermal throttle and begins to reduce performance).

To combat this, engineers developed the idea of designating alternating aisles as either hot or cold. All servers in a given aisle are then oriented such that the AC system provides cool air into the cold aisle where it is drawn in by the server which then exhausts it into the hot aisle where the ventilation system removes it from the room. This has the benefit of maximizing the temperature delta between the provided air and the server's processor(s), reducing the amount of quantity of air that must be provided in order to cool the server and improving overall system efficiency.

See [Figure 10](#) to understand how the hot-cold aisle configuration is setup in a data center.

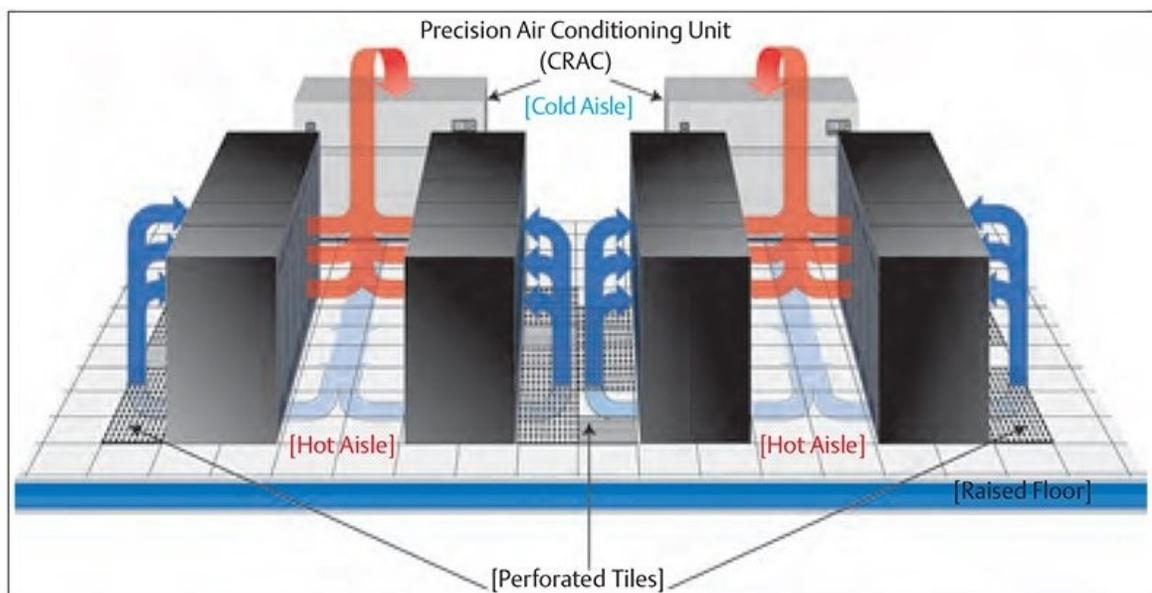


Figure 10: Hot Cold Isle [9]

4.1.5.5.1 Containment

While modern data centers employ highly sophisticated mechanisms to be as energy efficient as possible. One such mechanism which can be seen as an improvement on top of the Hot-Cold aisle arrangement is to use either hot aisle containment or cold aisle containment. Using a containment system can remove the issue with free flowing air.

As the name somewhat implies in cold aisle containment, the data center is designed so that only cold air goes into the cold aisle, this makes sure that the system only draws in cold air for cooling purposes. Conversely in hot aisle containment design, the hot aisle is contained so that the hot air collected in the hot aisle is drawn out by the cooling system and so that the cold air does not flow into the hot aisles[10].

4.1.5.5.1.1 Water Cooled Doors

Another good way of reducing the energy consumption is to install water cooled doors directly at the rack as shown in [Figure 11](#). Cooling even can be actively controlled so that in case of idle servers less energy is spent to conduct the cooling. There are many vendors that provide such cooling solutions.

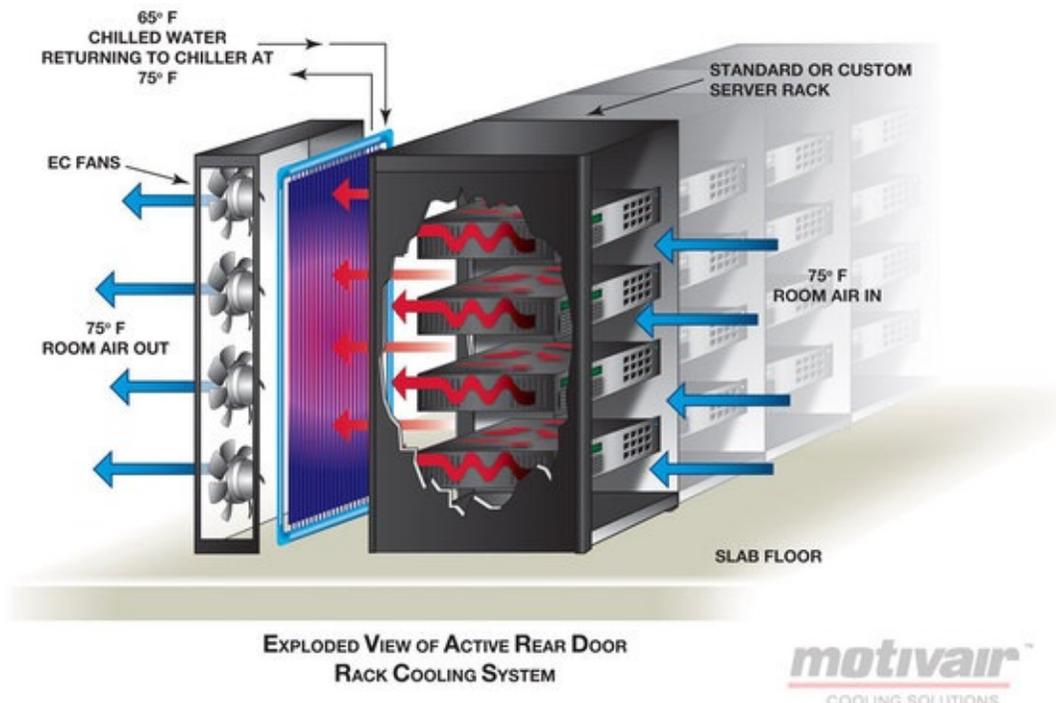


Figure 11: Active Rear Door [link](#)

4.1.5.6 Workload Monitoring

4.1.5.6.1 Workload of HPC in the Cloud

Clouds and especially university data centers do not just provide virtual machines but provide traditional super computer services. This includes the NSF sponsored XSEDE project. As part of this project the "XDMD auditing tool provides, for the first time, a comprehensive tool to measure both utilization and performance of high-end cyberinfrastructure (CI), with initial focus on XSEDE. Several case studies have shown its utility for providing important metrics regarding resource utilization and performance of TeraGrid/XSEDE that can be used for detailed analysis and planning as well as improving operational efficiency and performance. Measuring the utilization of high-end cyberinfrastructure such as XSEDE helps provide a detailed understanding of how a given CI resource is being utilized and can lead to improved performance of the resource in terms of job throughput or any number of desired job characteristics.

Detailed historical analysis of XSEDE usage data using XDMD clearly demonstrates the tremendous growth in the number of users, overall usage, and scale" [11].

Having access to a detailed metrics analysis allows users and center administrators, as well as project managers to better evaluate the use and utilization of such large facilities justifying their existence (see [Figure 12](#))

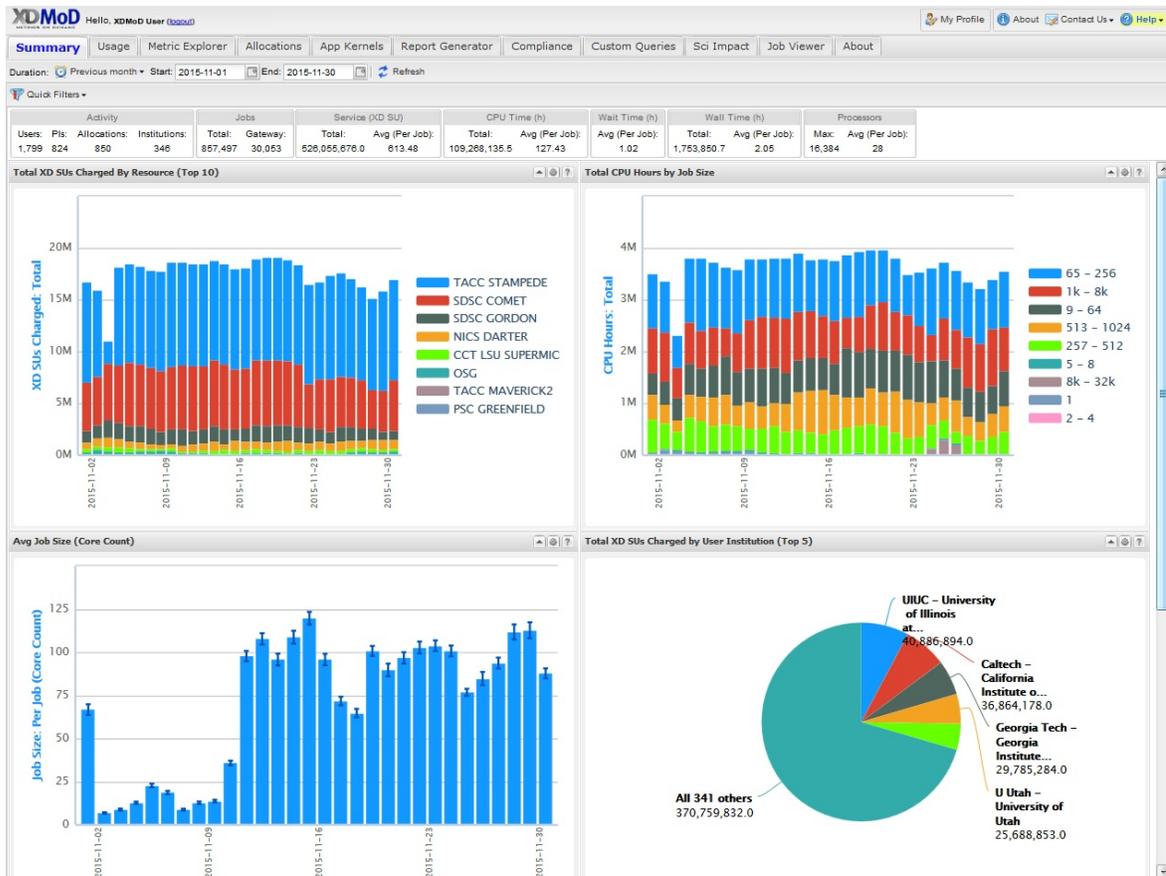


Figure 12: XDMoD: XSEDE Metrics on Demand

Additional information is available at

- <https://open.xdmod.org/7.5/index.html>

4.1.5.6.2 Scientific Impact Metric

Gregor von Laszewski and Fugang Wang are providing a scientific impact metric to XDMoD and XSEDE. It is a framework that (a) integrates publication and citation data retrieval, (b) allows scientific impact metrics generation at different aggregation levels, and (c) provides correlation analysis of impact metrics based on publication and citation data with resource allocation for a computing facility. This framework is used to conduct a scientific impact metrics evaluation of XSEDE, and to carry out extensive statistical analysis correlating XSEDE allocation size to the impact metrics aggregated by project and Field of Science. This analysis not only helps to provide an indication of XSEDE'S scientific impact, but also provides insight regarding maximizing the return on

investment in terms of allocation by taking into account Field of Science or project based impact metrics. The findings from this analysis can be utilized by the XSEDE resource allocation committee to help assess and identify projects with higher scientific impact. Through the general applicability of the novel metrics we invented, it can also help provide metrics regarding the return on investment for XSEDE resources, or campus based HPC centers [12].

4.1.5.6.3 Clouds and Virtual Machine Monitoring

Although no longer in operation in its original form [FutureGrid](#) [13] has pioneered the extensive monitoring and publication of its virtual machine and project usage. We are not aware of a current system that provides this level of detail as of yet. However, efforts as part of XSEDE within the XDMoD project are under way at this time but are not integrated.

Futuregrid provided access to all virtual machine information, as well as usage across projects. An archived portal view is available at [FutureGrid Cloud Metrics](#) [13].

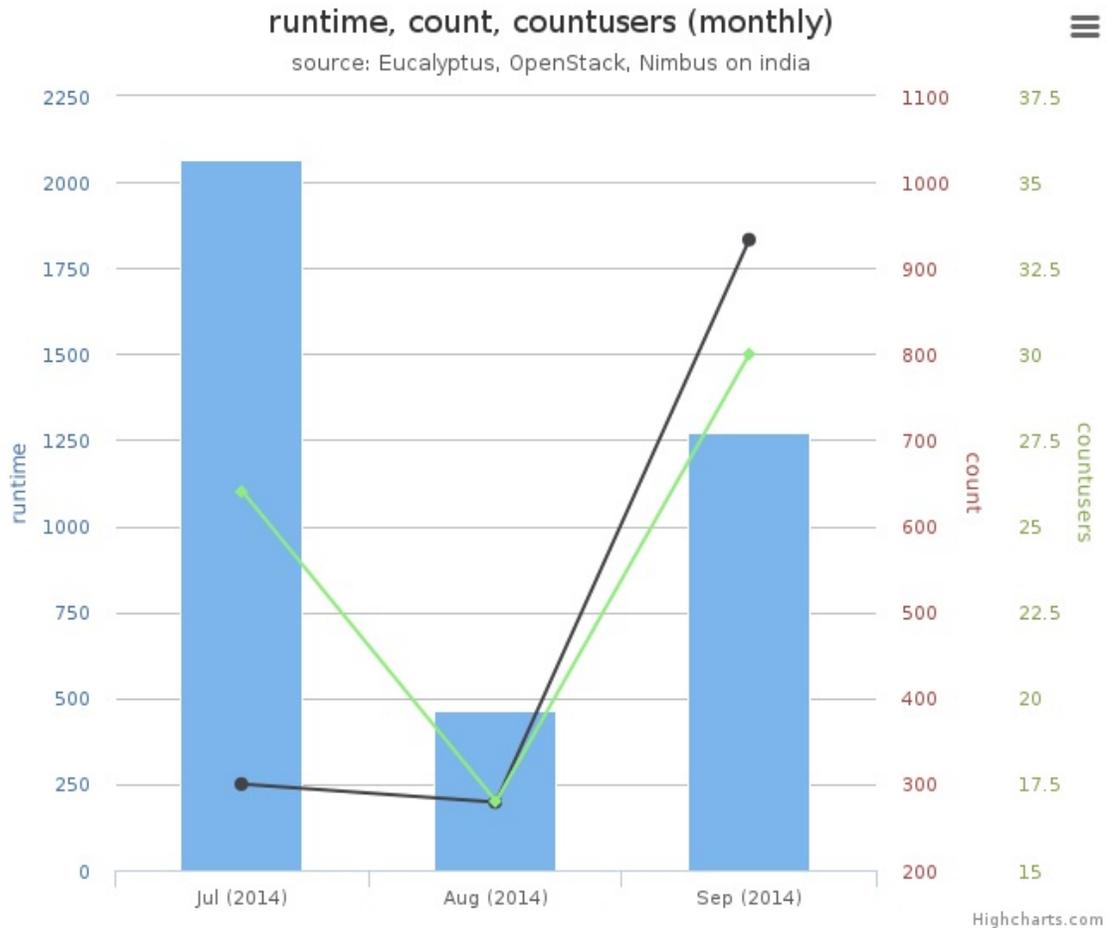


Figure 13: FutureGrid Cloud Metric

Futuregrid offered multiple clouds including clouds based on OpenStack, Eucalyptus, and Nimbus. Nimbus and Eucalyptus are systems that are no longer used in the community. Only OpenStack is the only viable solution in addition to the cloud offerings by Comet that do not uses OpenStack (see [Figure 13](#)).

Futuregrid, could monitor all of them and published its result in its Metrics portal. Monitoring the VMs is an important activity as they can identify VMs that may no longer be used (the user has forgotten to terminate them) or too much usage of a user or project can be detected in early stages.

We like to emphasize several examples where such monitoring is helpful:

- Assume a student participates in a class, metrics and logs allow to identify students that do not use the system as asked for by the instructors. For example it is easy to identify if they logged on and used VMs. Furthermore

the length of running a VM ba

- Let us assume a user with willful ignorance does not shut down VMs although they are not used because research clouds are offered to us for free. In fact, this situation happened to us recently while using another cloud and such monitoring capacities were not available to us (on jetstream). The user single-handedly used up the entire allocation that was supposed to be shared with 30 other users in the same project. All accounts of all users were quasi deactivated as the entire project they belonged to were deactivated. Due to allocation review processes it took about 3 weeks to reactivate full access. sed on the tasks to be completed can be compared against other student members.
- In commercial clouds you will be charged money. Therefore, it is less likely that you forget to shutdown your machine
- In case you use GitHub carelessly and post your cloud passwords or any other passwords in it, you will find that within five minutes your cloud account will be compromised. There are individuals on the network that cleverly mine GitHub for such security lapses and will use your password if you indeed have stored them in it. In fact GitHub's deletion of a file does not delete the history, so as a non expert deleting the password form GitHub is not sufficient. You will have to either delete and rewrite the history, but definitely in this case you will need to reset the password. Monitoring the public cloud usage in the data center is important not only in your region but other regions as the password is valid also there and intruders could hijack and start services in regions that you have never used.

In addition to FutureGrid, we like to point out Comet (see other sections). It contains an exception for VM monitoring as it uses a regular batch queuing system to manage the jobs. Monitoring of the jobs is conducted through existing HPC tools.

4.1.5.6.4 Workload of Containers

Monitoring tools for containers such as for kubernetes are listed at:

- <https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/>

Such tools can be deployed alongside kubernetes in the data center, but will

likely have restrictions to its access. They are for those who operate such services for example in kubernetes. We will discuss this in future sections in more detail.

4.1.6 Example Data Centers

In this section we will be giving some data center examples while looking at some of the mayor cloud providers.

4.1.6.1 AWS

AWS focuses on security aspects of their data centers that include four aspects [14]:

- [Perimeter Layer](#)
- [Infrastructure Layer](#)
- [Data Layer](#)
- [Environmental Layer](#)

The [global infrastructure](#) [15] as of January 2019 includes 60 Availability Zones within 20 geographic Regions. Plans exists to add 12 Availability Zones and four additional Regions in Bahrain, Hong Kong SAR, Sweden, and a second AWS GovCloud Region in the US (see [Figure 14](#)).

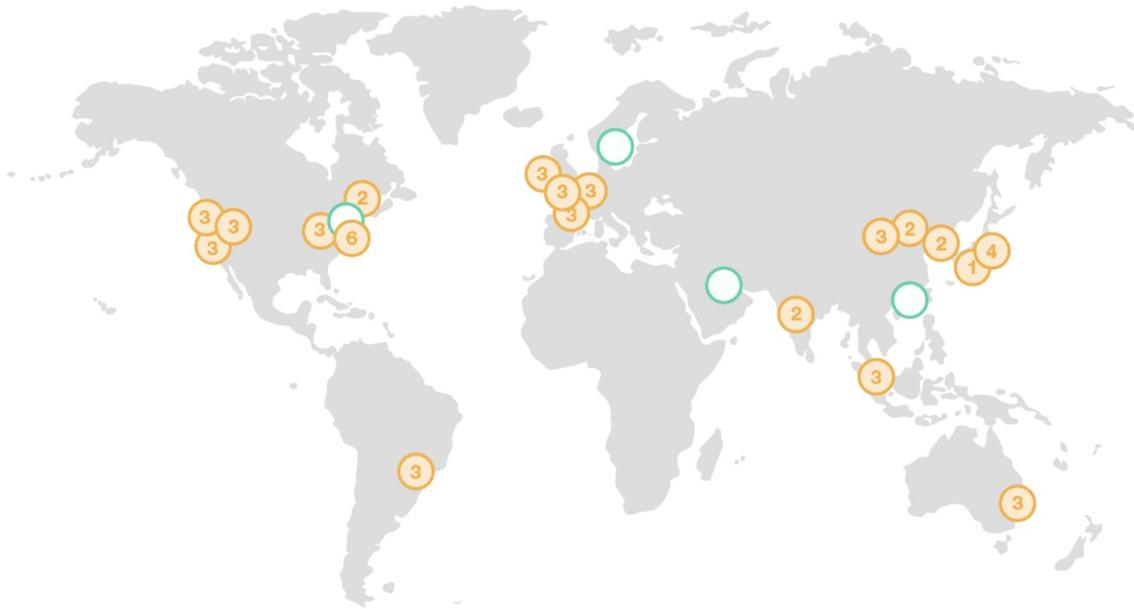


Figure 14: AWS regions [15]

Amazon strives to achieve high availability through multiple availability zones, improved continuity with replication between regions, meeting compliance and data residency requirements as well as providing geographic expansion. See [Figure 15](#)

The regions and number of availability zones are as follows:

- Region US East: N. Virginia (6), Ohio (3) US West N. California (3), Oregon (3)
- Region: Asia Pacific Mumbai (2), Seoul (2), Singapore (3), Sydney (3), Tokyo (4), Osaka-Local (1)1 Canada Central (2) China Beijing (2), Ningxia (3)
- Region: Europe Frankfurt (3), Ireland (3), London (3), Paris (3) South America São Paulo (3)
- Region Gov Cloud: AWS GovCloud (US-West) (3)
- New Region (coming soon): Bahrain, Hong Kong SAR, China, Sweden, AWS GovCloud (US-East)

4.1.6.2 Azure

Azure claims to have more global [regions](#) [16] than any other cloud provider.

They motivate this by their advertisement to bring and applications to the users around the world. The goal is similar as other commercial hyper-scale providers by introducing preserving data residency, and offering comprehensive compliance and resilience. As of Aug 29, 2018 Azure supports 54 regions worldwide. These regions can currently be accessed by users in 140 countries (see [Figure 15](#)). Not every service is offered in every region as the service to region matrix shows:

- <https://azure.microsoft.com/en-us/global-infrastructure/services/>



Figure 15: Azure regions [[16](#)]

4.1.6.3 Google

From [Google](#) [[17](#)] we find that on Aug. 29th Google has the following data center locations (see [Figure 16](#)):

- **North America:** Berkeley County, South Carolina; Council Bluffs, Iowa; Douglas County, Georgia; Jackson County, Alabama; Lenoir, North Carolina; Mayes County, Oklahoma; Montgomery County, Tennessee; The Dalles, Oregon
- **South America:** Quilicura, Chile
- **Asia:** Changhua County, Taiwan; Singapore
- **Europe:** Dublin, Ireland; Eemshaven, Netherlands; Hamina, Finland; St Ghislain, Belgium



Figure 16: Google data centers [17]

Each data center is advertised with a special environmental impact such as a unique cooling system, or wildlife on premise. Google's data centers support its service infrastructure and allow hosting as well as other cloud services to be offered to its customers.

Google highlights its efficiency strategy and methods here:

- <https://www.google.com/about/datacenters/efficiency/>

They summarize their offers are based on

- Measuring the PUE
- Managing airflow
- Adjusting the temperature
- Use free Cooling
- Optimizing the power distribution

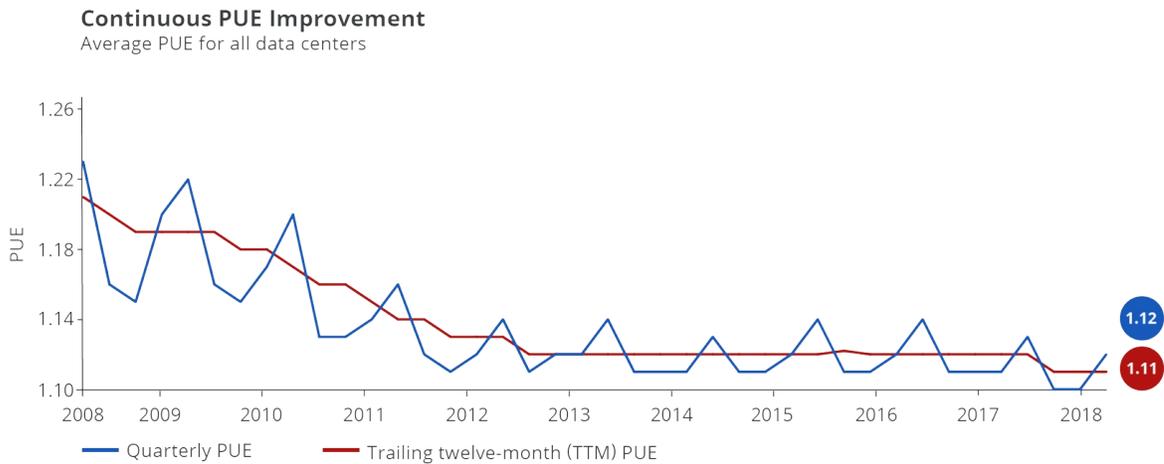


Figure 17: PUE data for all large-scale Google data centers

The [PUE \[18\]](#) data for all large-scale Google data centers is shown in [Figure 17](#)

An important lesson from Google is the PUE boundary. That is the different efficiency based on the closeness of the IT infrastructure to the actual data center building. This indicates that it is important to take at any providers definition of PUE in order not to report numbers that are not comparable between other vendors and are all encompassing.

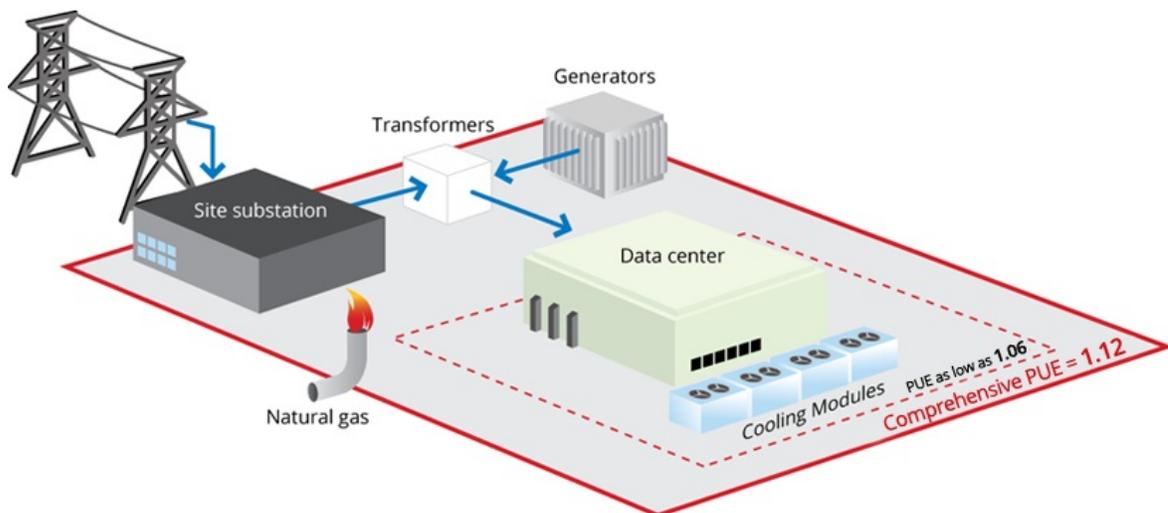


Figure 18: Google data center PUE measurement boundaries [\[18\]](#)

[Figure 18](#) shows the Google data center PUE measurement boundaries. The [average PUE \[18\]](#) for all Google data centers is 1.12, although we could boast a PUE as low as 1.06 when using narrower boundaries.

As a consequence, Google is defining its PUE in detail in [Equation 7](#).

$$PUE = \frac{ESIS + EITS + ETX + EHV + ELV + EF}{EITS - ECRAC - EUPS - ELV + ENet1} \quad (7)$$

where the abbreviations stand for

- ESIS = Energy consumption for supporting infrastructure power substations feeding the cooling plant, lighting, office space, and some network equipment
- EITS = Energy consumption for IT power substations feeding servers, network, storage, and computer room air conditioners (CRACs)
- ETX = Medium and high voltage transformer losses
- EHV = High voltage cable losses
- ELV = Low voltage cable losses
- EF = Energy consumption from on-site fuels including natural gas & fuel oils
- ECRAC = CRAC energy consumption
- EUPS = Energy loss at uninterruptible power supplies (UPSes) which feed servers, network, and storage equipment
- ENet1 = Network room energy fed from type 1 unit substitution

For more [details](#) see [18].

4.1.6.4 IBM

IBM maintains almost 60 data centers, which are placed globally in 6 regions and 18 availability zones. IBM targets businesses while offering local access to its centers to allow for low latency. IBM states that through this localization users can decide where and how data and workloads and address availability, fault tolerance and scalability. As IBM is business oriented it also stresses its certified security.

More information can be obtained from:

- <https://www.ibm.com/cloud/data-centers/>

A special service offering is provided by Watson.

- <https://www.ibm.com/watson/>

which is focusing on AI based services. It includes PaaS services for deep learning, but also services that are offered to the healthcare and other communities as SaaS

4.1.6.5 XSEDE

XSEDE is an NSF sponsored large distributed set of clusters, supercomputers, data services, and clouds, building a “single virtual system that scientists can use to interactively share computing resources, data and expertise”. The Web page of XSEDE is located at

- <https://www.xsede.org/>

Primary compute resources are listed in the resource monitor at

- <https://portal.xsede.org/resource-monitor>

For cloud Computing the following systems are of especial importance although selected others may also host container based systems while using singularity (see [Figure 19](#)):

- Comet virtual clusters
- Jetstream OpenStack

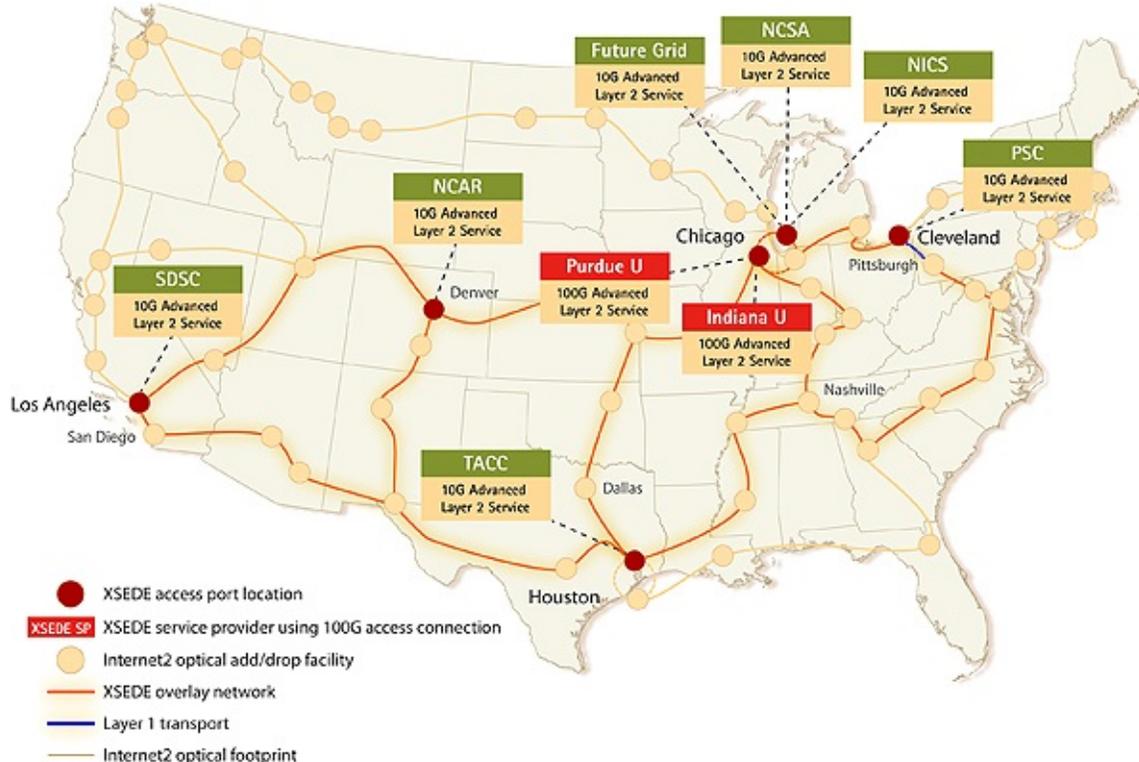


Figure 19: XSEDE distributed resource infrastructure

4.1.6.5.1 Comet

The comet machine is a larger cluster and offers bare metal provisioning based on KVM and SLURM. Thus it is a unique system that can run at the same time traditional super computing jobs such as MPI based programs, as well as jobs that utilize virtual machines. With its availability of >46000 cores it provides one of the largest NSF sponsored cloud environment. Through its ability to provide bare metal provisioning and the access to Infiniband between all virtual machines it is an ideal machine for exploring performance oriented virtualization techniques.

Comet has about 3 times more cores than Jetstream.

4.1.6.5.2 Jetstream

Jetstream is a machine that specializes in offering a user friendly cloud environment. It utilizes an environment called atmosphere that is targeting inexperienced scientific cloud users. It also offers an OpenStack environment that is used by atmosphere and is for classes such as ours the preferred access

method. More information about the system can be found at

- <https://dcops.iu.edu/>

4.1.6.6 Chameleon Cloud

Chameleon cloud is a configurable experimental environment for large-scale cloud research. It is offering OpenStack as a service including some more advanced services that allow experimentation with the infrastructure.

- <https://www.chameleoncloud.org/>

An overview of the hardware can be obtained from

- <https://www.chameleoncloud.org/hardware/>

4.1.6.7 Indiana University

Indiana University has a data center in which many different systems are housed. This includes not only jetstream, but also many other systems. The systems include production, business, and research clusters and servers. See [Figure 20](#)



Figure 20: IU Data Center

On the research cluster side it offers Karst and Carbonate:

- <https://kb.iu.edu/d/bezu> (Karst)
- <https://kb.iu.edu/d/aolp> (Carbonate)

One of the special systems located in the data center and managed by the Digital

Science Center is called Futuresystems, which provides a great resource for the advanced students of Indiana University focusing on data engineering. While systems such as Jetstream and Chameleon cloud specialize in production ready cloud environments, Futuresystems, allows the researchers to experiment with state-of-the-art distributed systems environments supporting research. It is available with Comet and thus could also serve as an on-ramp to using larger scale resources on comet while experimenting with the setup on Futuresystems.

Such an offering is logical as researchers in the data engineering track want to further develop systems such as Hadoop, SPark, or container based distributed environments and not use the tools that are released for production as they do not allow improvements to the infrastructure. Futuresystems is managed and offered by by the Digital Science Center.

Hence IU offers very important but needed services

- Karst for traditional supercomputing
- Jetstream for production use with focus on virtual machines
- Futuresystems for research experiment environments with access to bare metal.

4.1.6.8 Shipping Containers

A few years ago data centers build from shipping containers were very popular. This includes several main Cloud providers. Such providers have found that they are not the best way to develop centers at scale. This includes [Microsoft](#) and [Google](#) The current trend however is to build mega or hyperscale data centers.

4.1.7 Server Consolidation

One of the driving factors in cloud computing and the rise of large scale data centers is the ability to use server virtualization to place more than one server on the same hardware. Formerly the services were hosted on their own servers. Today they are managed on the sae hardware although they look to the customer like separate servers.

As a result we find the following advantages:

- **reduction of administrative and operations cost:** While we reduce the number of servers and utilize hardware to host multiple on them management cost, space, power, and maintenance cost are reduced.
- **better resource utilization:** Through load balancing strategies servers can be better utilized while for example increase load so resource idling is avoided.
- **increased reliability:** As virtualized servers can be snapshotted, and mirrored, these features can be utilized in strategies to increase reliability in case of failure.
- **standardization:** As the servers are deployed in large scale, the infrastructure is implicitly standardized based on server, network, and disk, making maintenance and replacements easier. This also includes the software that is running on such servers (OS, platform and may even include applications).

4.1.8 Data Center Improvements and Consolidation

Due to the immense number of servers in data centers, as well as the increased workload on its servers, the energy consumption of data centers is large not only to run the servers, but to provide the necessary cooling. Thus it is important to revisit the impact such data centers have on the energy consumption. One of the studies that looked into this is from 2016 and is published by [LBNL \[19\]](#) In this study the data center electricity consumption back to 2000 is analyzed while using previous studies and historical shipment data. A forecast is with different assumption is contrasts till 2020

Figure Energy Forecast depicts “an estimate of total U.S. data center electricity use (servers, storage, network equipment, and infrastructure) from 2000-2020” (see [Figure 21](#)).

While in “2014 the data centers in the U.S. consumed an estimated 70 billion kWh” or “about 1.8% of total U.S. electricity consumption”. However, more recent studies find an increase by about 4% from 2010-2014. This contrasts a large derivation from the 24% that were originally predicted several years ago. The study finds that the predicted energy use would be approximately 73 billion

kWh in 2020.

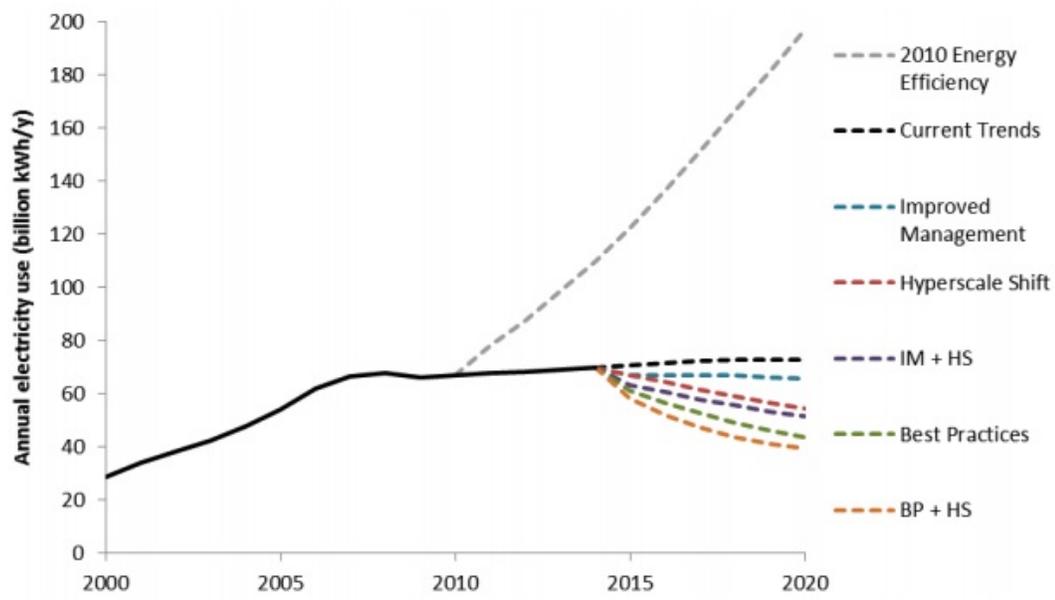


Figure 21: Energy Forecast [19]

It is clear that the original prediction of large energy consumption motivated a trend in industry to provide more energy efficient data centers. However if such energy efficiency efforts would not be conducted or encouraged we would see a completely different scenario.

The scenarios are identified that will significantly impact the prediction:

- **improved management** increases energy-efficiency through operational or technological changes with minimal investment. Strategies include improving the least efficient components.
- **best practices** increases the energy-efficiency gains that can be obtained through the widespread adoption the most efficient technologies and best management practices applicable to each data center type. This scenario focuses on maximizing the efficiency of each type of data center facility.
- **hyperscale data centers** where the infrastructure will be moved from smaller data centers to larger *hyperscale* data centers.

4.1.9 Project Natick

To reduce energy consumption in data centers and reduce cost of cooling Microsoft has developed *Project Natick*. To tackle this problem Microsoft has built underwater datacenter. Another benefit of this project is that data center can be deployed in large bodies of water to serve customers residing in that area so it helps to reduce latency by reducing distance to users and therefore increasing data transfer speed. There are two phases of this project.

The project was executed in two phases.

Phase 1 was executed between August to November 2015. In this phase Microsoft was successfully able to deploy and operate vessel underwater. The vessel was able to tackle cooling issues and effect of biofouling as well. Biofouling is referred to as the fouling of pipes and underwater surfaces by organisms such as barnacles and algae.

The PUE (Power Usage Effectiveness) of Phase1 vessel was 1.07 which is very efficient and a perfect WUE (Water Usage Effectiveness) of exactly 0, while land data centers consume ~4.8 liters of water per KWH. This vessel consumed computer power equivalent to 300 Desktop PCs and was of 38000 lbs and it operated for 105 days.



Figure 22: The Leona Philpot prototype

See [Figure 22](#), The *Leona Philpot* prototype was deployed off the central coast of California on Aug. 10, 2015. Source: [Microsoft \[20\]](#)

The phase 2 started in June 2018 and lasted for 90 days. Microsoft deployed a vessel at the European Marine Energy Center in UK.

The phase 2 vessel was 40ft long and had 12 racks containing 864 servers. Microsoft powered this data center using 100% renewable energy. This data center is claimed to be able to operate without maintenance for 5 years. For cooling Microsoft used infrastructure which pipes sea water through radiators in back on server racks and then move water back in to ocean.

The total estimated lifespan of a Natick datacenter is around 20 years, after which it will be retrieved and recycled.



Figure 23: The Northern Isles prototype

Source: [Microsoft](#) [21]

[Figure 23](#) shows the *Northern Isles* prototype being deployed near Scotland.

Although the cooling provides a significant benefit while using seawater, it is clear that long time studies need to be conducted with this approach and not just studies over a very short period of time. The reason for this is not just the measurement of a PUE factor or the impact of algae on the infrastructure, but also how such a vessel impacts the actual ecosystem itself.

Some thought on this include:

1. How does the vessel increase the surrounding water temperature and effects the ecosystem
2. If placed in saltwater, corrosion can occur that may not only effect the vessel, but also the ecosystem
3. Positive effects could also be created on ecosystems, which is for example demonstrated by creation of artificial reefs. However if the structure has to be removed after 20 years, which impact has it than on the ecosystem.

Find more about this at [\[22\]](#)

4.1.10 Renewable Energy for Data Centers

Due to the high energy consumption of data-centers, especially of hyper-scale data centers. It is prudent to evaluate renewable energies for the operation of such data centers. In particular this includes:

- Solar: <https://9to5google.com/2019/01/17/largest-ever-solar-farms-google/>
- Wind: <https://www.datacenterknowledge.com/wind-powered-data-centers>
- Hydro: <http://www.hydroquebec.com/data-center/advantages/clean-energy.html> there will be others
- Thermal: find better resource <https://spectrum.ieee.org/energywise/telecom/internet/iceland-data-center-paradise>
- Recyclers: <https://www.datacenterknowledge.com/data-centers-that-recycle-waste-heat>

Other aspects may include the storage of energy including:

- Batteries
- Store energy in other forms such as potential energy.

4.1.11 Societal Shift Towards Renewables

We find a world wide trend in society to shift towards renewables. This includes government efforts to support renewable in benefit of the society:

- Germany, China, Island, and many more,

but it also includes commitments by Corporations such as

- [Google](#), [AWS](#), [IBM](#), ...

Also look at the US state of California and others that project renewable energy.

Information about this is provided for example in

- https://www.irena.org/-/media/Files/IRENA/Agency/Publication/2018/Jan/IRENA_2017_Power_C

We predict that any country not being heavily committed in renewable energies will fall behind while missing out on research opportunities in renewable energies themselves. Today the cost of renewable energy producing power plants have so drastically improved they are not only producing less green house gasses but are today even cheaper to operate and build than combustion based energy producing power plants.

4.1.12 Datacenter Risks and Issues

Data Centers may be encounter issues such as outages of a variety of reasons. In this section you will identify risks and issues that you encounter as part of information you find on the Web or literature.

It is important that we recognize that datacenter outages can happen and thus such outages must be build into the operations of the cloud services that are offered to the customers to assure an expectation level of quality of service.

Here is an example:

- Date: Mar 02 2017
- Datacenter: AWS
- Category: Operator error
- Description: Mistake during a routine debugging exercise.
- Duration: Several hours
- Impact: large
- Cost impact > \$160 million
- [Reference](#)
- [FACEBOOK, WHATSAPP, INSTAGRAM DOWN? WORLD'S BIGGEST SOCIAL NETWORKS SUFFER OUTAGES AND STOP LOADING IMAGES, Newsweek](#)

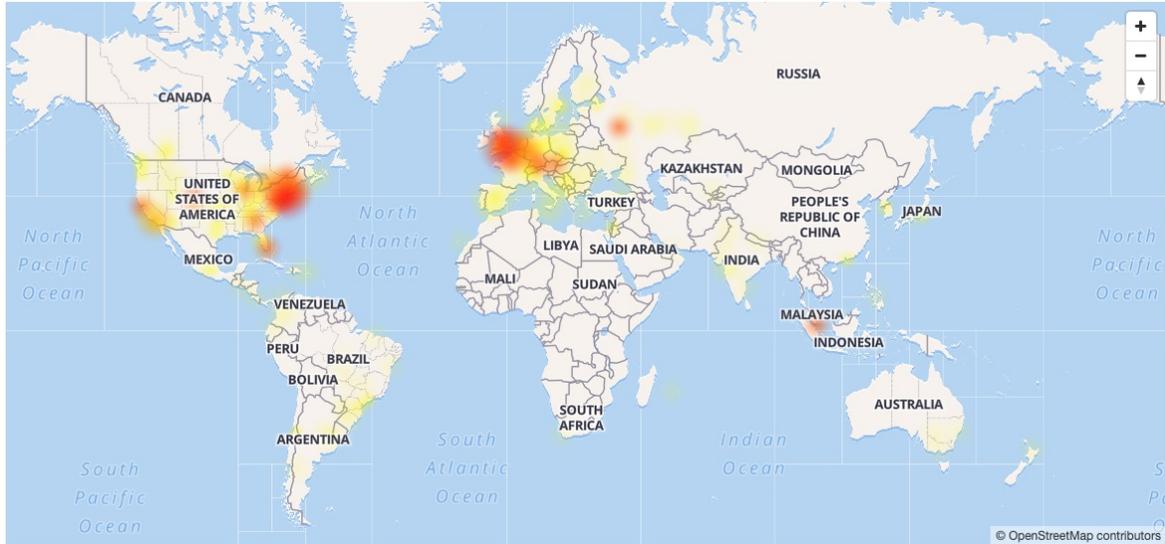


Figure 24: Instagram Outage [Ref](#)

4.1.13 Exercises

Exercises

Prerequisite: Knowledge about plagiarism.

E.Datacenter.1: Carbon footprint of a data center

Complete the definitions of the terms used in the relevant section

E.Datacenter.2: Carbon footprint of data centers

E.Datacenter.2.a: Table

World wide we have many data centers. Your task will be to find the carbon emission of a data center and its cost in \$ based on energy use on a yearly basis. Make sure you avoid redundant reporting and find a new datacenter. Add your data to the table in this [link](#) under the Sheet DataCenter

E.Datacenter.2.b: Table

place a file in your repository called *datacenter.md* (note the spelling all lower case) and describe details about you data center without plagiarizing. Provide

references to back up your data and description.

TAs will integrate your information into the following table

Table: Cost of the data center

Data Center	Location	Year	Electricity Cost*	IT Load	Yearly Cost	Yearly CO2 Footprint	Equip in Ca
⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗

*as adjusted in calculator

If you find other estimates for a data center or an entire data center fleet such as AWS world wide, please provide citations.

E.Datacenter.3: Your own Carbon footprint

It is interesting to compare and measure your own carbon footprint. We will ask you anonymously to report your carbon footprint via a form we will prepare in future. As the time to do this is less than 2 minutes, We ask all students to report their footprint.

Please use the calculator at:

- <http://carbonfootprint.c2es.org/>

Add your data to the table in this [link](#) under the Sheet CarbonFootPrint

E.Datacenter.4:

Pick a renewal energy from [Section 4.1.10](#) and describe what it is. Find data centers that use this energy form. Include the information in your hid directory under the file datacenter.md.

You will do the energy form based on taking the last digit from your HID and getting the modulo 6 and looking up the value in this table

0. Solar
1. Wind
2. Hydro
3. Thermal
4. Recyclers
5. Others

If your modulo is undefined please use 5 (e.g. if your digit at the end of your hid 0)

E.Datacenter.5:

Pick a country, state, or company from [Section 4.1.11](#) and summarize their efforts towards renewable energy and impacts within the society. Create a section and contribute it to the datacenter.md file. Use internet resources and cite them.

E.Datacenter.6:

Write about cooling technologies in datacenter rack doors so it can be contributed to [Section 4.1.5.5.1.1](#).

E.Datacenter.7:

Review: [Nature Article](#). Is there any more up to data available? What lessons do we take away from the article?

E.Datacenter.8:

Find major data center outages and discuss the concrete impact on the internet and user community. Concrete means here not users of the center xyz lost services, but can you identify how many users or how many services were impacted and which? Is there a cost revenue loss projected somewhere? If you find an outage do significant research on it. For example other metrics could include what media impact did this outage have?

E.Datacenter.9:

We encourage you to contribute to this section if you like and enjoy doing so.

5 ARCHITECTURE

5.1 ARCHITECTURES



Learning Objectives

- Review classical architectural models leading up to cloud computing.
 - Review some mayor cloud architecture views.
 - Visualize the NIST cloud architecture
 - Discuss an architecture for multicloud frameworks.
-
-

While we have introduced in our introductory section a number of definitions for cloud computing, as well as an architectural view for clouds based on the *as a Service* model, we will look a bit closer at other alternative views. These views are in some cases important as they provide appropriate abstractions for more detailed implementations.

5.1.1 Evolution of Compute Architectures

We start our observation with some a depiction of some of the important architectural models motivating the current state of information technology services we provide in [Figure 25](#). The are computers used primarily by large organizations for critical applications; bulk data processing, such as census, industry and consumer statistics, enterprise resource planning; and transaction processing. The term originally referred to the large cabinets called “main frames” that housed the central processing unit and main memory of early computers. has been updated by von Laszewski to include the mobile computing and the internet of things phase that is bringing rapid changes to how we perceive and use the cloud in the near future.

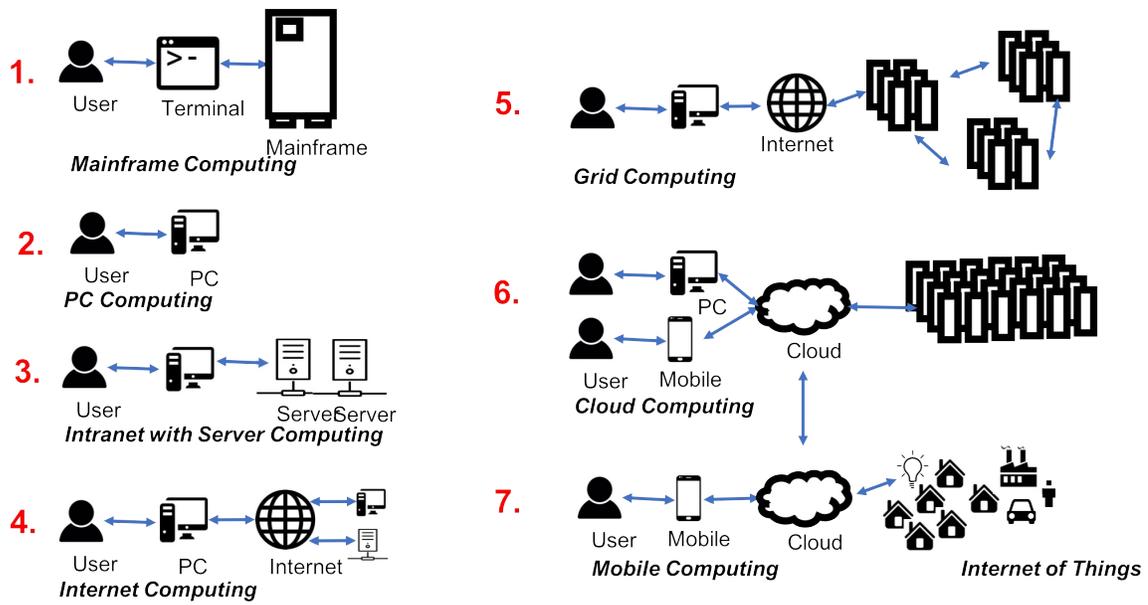


Figure 25: Evolution of Compute Architectures

We define the following terminology based on the evolution of compute architectures.

5.1.1.1 Mainframe Computing

Mainframe computing is using the larger and more reliable computers, like IBM System z9, to run the critical applications, bulk data processing, enterprise resource planning and business transaction processing.

According to [Wikipedia](#), the term mainframe originally referred to the large cabinets called “main frames” that housed the central processing unit and main memory of early computers. Later, the term was used to distinguish high-end commercial machines from less powerful units. Most large-scale computer system architectures were established in the 1960s, but continue to evolve. Mainframe computers are used primarily by large organizations for critical applications; bulk data processing, such as census, industry and consumer statistics, enterprise resource planning; and transaction processing. The term originally referred to the large cabinets called “main frames” that housed the central processing unit and main memory of early computers.

Some key attributes of Mainframes that distinguishes it from other computers include its larger size, speed, throughput, power and environmental requirements, and operating system. Furthermore, we find that they have inbuilt redundancy to address high uptimes as required by business applications. Even some of the earliest Mainframes supported fast I/O and computation via virtualization. The concept of hot swapping of hardware help these machines to run without failure for years.

5.1.1.2 PC Computing

The term PC is short for personal computer. The first PCs were introduced by IBM to the market. PCs need an operating system such as Windows, macOS, or Linux

PC Computing refers to

an era where consumers predominantly used personal computers to conduct their work. Such computers were mostly stand alone without network as early networks were not available to consumers.

5.1.1.3 Intranet and Server Computing

We refer to Intranet and Server Computing as an environment in which

the computers are part of an private network, also called, intranet, that is contained within an enterprise and later on also homes. Intranets are able to connect many local resources within a Local but also a wide area network

5.1.1.4 Grid Computing Computing

and its evolution is defined in [The Grid-Idea and Its Evolution](#). The original definition of Grid computing has been summarised as follows:

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. [23]

However, in the paper we also define that Grids were not just about computing, but introduced an approach that through the introduction of virtual organizations lead to the following definition

A production Grid is a shared computing infrastructure of hardware, software, and knowledge resources that allows the coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations to enable sophisticated international scientific and business-oriented collaborations.

This definition is certainly including services that are today offered by the Cloud. Hence in the early days of cloud computing there was a large debate occurring if cloud is just another term for Grid. In [Cloud Computing and Grid Computing 360-Degree Compared](#) an analysis is conducted between the different architecture models outlining that collective resources and connectivity protocols introduced by the Grid community have been replaced by the cloud with platform and unified resources.

To provide a very simple but possibly incomplete comparison, Cloud computing integrated infrastructure such as supercomputers and other large scale resources through unified protocols. The effort was initially provided by research institutions but have been introduced in business. However, with the growth of the data centers to foster common tasks such as Web hosting, we see a clear difference:

- while the Grid was originally designed to give a few scientist access to the biggest agglomerated research supercomputers,
- business focused on serving originally millions of users with the need to run only a view data or compute services.

This certainly resulted in independent development, while cloud computing has today consumed Grids. Tools such as the GLOBUS toolkit are no longer widely used, and the development has shifted to the support of data services only.

5.1.1.5 Internet Computing

With the ocurance of the WWW protocols, internet commuting brought to the consumers a global computer network providing a variety of information and

communication facilities.

Internet Computing refers to

the infrastructure that enables sharing of data, within the WWW community.

Internet computing also comprises early infrastructures such as AOL, which popularized the term *you got mail*

5.1.1.6 Cloud Computing

Cloud Computing refers to

delivery of services such as database, server, network storage and others over the internet so the user doesnot have to maintain a datacenter and only pays for services in use. This reduces the cost and increases the productivity as services can be available in minutes on demand with state of the art security and no hardware datacenter staff needed on the users side.

We have provided a lecture about the definition of cloud computing previously.

5.1.1.7 Mobile Computing

Mobile Computing refers to

a diverse set of devices allowing users to access data and information from wherever they are with mobile devices such as cell phones or tablet computers. mobile computing is dominated by transmission of data, voice, and video over a network via the mobile device

5.1.1.8 Internet of Things Computing

Internet of Things Computing refers to

devices that are interconnected via the internet while they are embedded in things or common objects. The dives send and receive

data to be integrated into a network with sensors and actuators reacting upon sensory and other data.

5.1.1.9 Edge Computing

In addition, we need to point out two additional terms that we will integrate in this image. Edge Computing and Fog Computing. Currently there is still some debate about what these terms are, but we will follow the following definitions:

Edge Computing refers to

computing conducted on the very edge of infrastructure. This means that data that is not needed in the data center can be calculated and analyzed on the edge devices instead. No interaction between cloud services is needed. Only the absolute required data is sent to the cloud.

5.1.1.10 Fog Computing

FoG Computing refers to

computing conducted in-between the cloud and the edge devices. This could be for example part of a smart network, that hosts a small set of analytics capabilities, so that the data does not have to travel back to the data center, but the edge device is not powerful enough to do the calculation. Thus a Fog computing infrastructure provides the ability to conduct the analysis closer to the edge saving valuable resources while not needing to transmit all data to the data center although it will be analyzed

5.1.2 As a Service Architecture Model

The *as a Service* architecture was one of the earliest definitions of cloud architecture while focusing on the service aspect provided by the cloud. The layers such as IaaS, PaaS, and SaaS provide a layered architecture view while separating infrastructure, platform, and services. This allows a separation of concerns typically between infrastructure providers, platform developers, and

software architects using platforms and or infrastructure services.

The typical triangular diagram (see [Figure 26](#)) is often used to represent it.

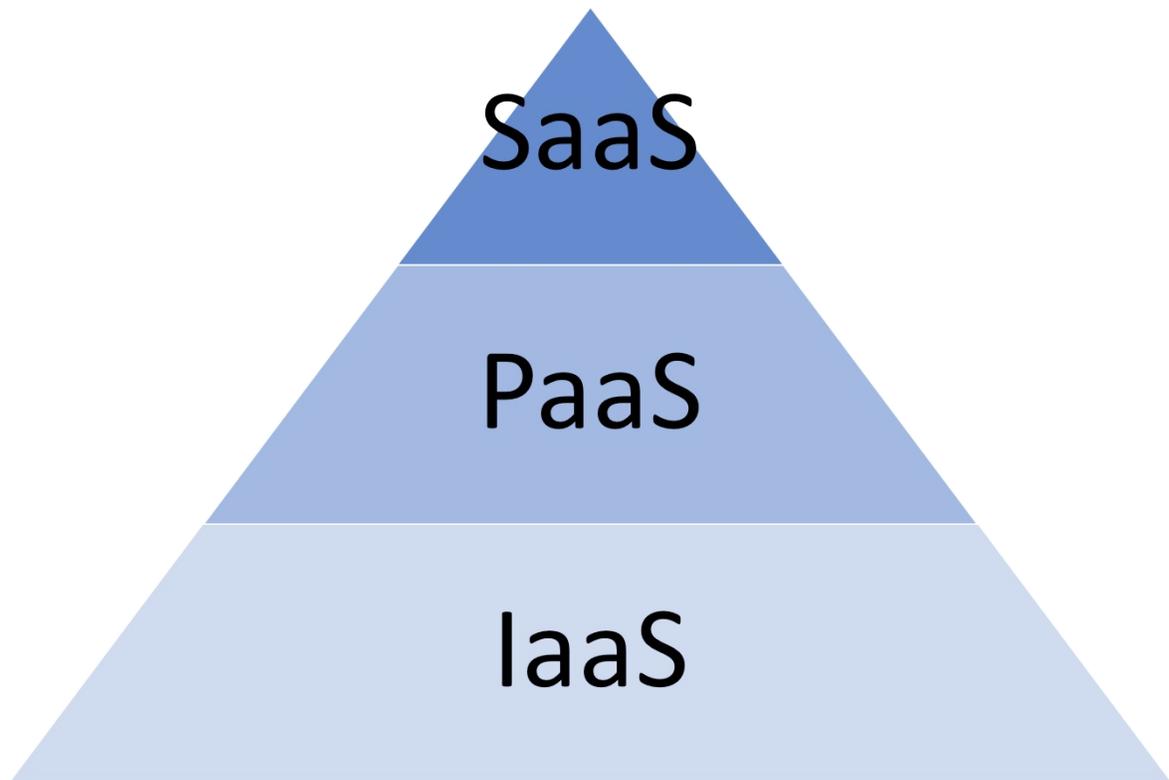


Figure 26: Infrastructure as a Service [Source](#)

5.1.3 Product or Functional Based Model

When we inspect prominent providers such as Amazon, Azure, and Google, we find that on their Web pages they do provide their customers an alternative view that is motivated by exposing numerous products to the customers grouped by functions. These services are often in the hundrest. To achieve the exposure of the products in a meaningful fashion, they introduce a functional view motivation a functional architecture view of the cloud.

When we analyse these functions for example for Amazon Web services we find the following

- Compute
- Storage
- Databases

- Migration
- Networking & Content Delivery
- Developer Tools
- Management Tools
- Media Services
- Security, Identity & Compliance
- Machine Learning
- Analytics
- Mobile
- Augmented reality and Virtual Reality
- Application Integration
- Customer Engagement
- Business Productivity
- Desktop & App Streaming
- Internet of Things
- Game Development
- AWS Marketplace Software
- AWS Cost Management

From this we derive that for the initial contact to the customer the functionality is put in foreground, rather than the distinction between SaaS, PaaS, and IaaS. If we sort these services into the as a Service mode we find:

- IaaS
 - Compute
 - Storage
 - Databases
 - Migration
 - Networking & Content Delivery
- PaaS
 - Developer Tools
 - Management Tools
 - Media Services
 - Security, Identity & Compliance
 - Machine Learning
 - Analytics
 - Mobile

- Augmented reality and Virtual Reality
- Application Integration
- Customer Engagement
- Business Productivity
- Desktop & App Streaming
- Game Development
- AWS Marketplace Software
- AWS Cost Management
- Internet of Things

We observe that AWS focuses on providing infrastructure and platforms so others can provide integrated service to its customers.

Other examples for product lists such as the one from Azure are provided in the Appendix.

5.1.4 NIST Cloud Architecture

In the introduction we have extensively discussed the NIST cloud architecture. A nice visual representation is provided in [Figure 27](#).

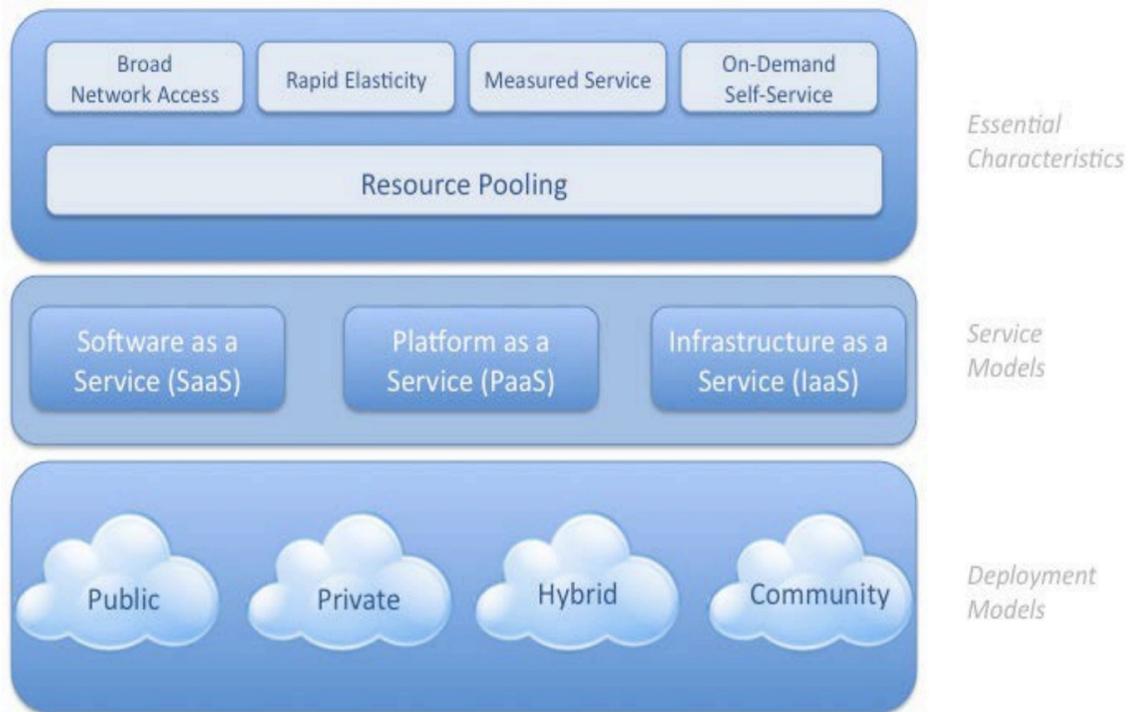


Figure 27: Visual representation of the NIST Cloud Architecture
[Source](#)

5.1.5 Cloud Security Alliance Reference Architecture

Founded in 2008, the Cloud Security Alliance (CSA) is a nonprofit organization that provides a variety of security resources to institutions including guidelines, education and best practices for adoption.

This is a great organization to lean on if you have open questions about architecture and the best way to secure it. There are working groups that look across 38 domains of Cloud Security. These groups meet actively and they cover current topics, opportunities and ask relevant questions. It is a great place to networks with experts in the field and ask questions specific to your company or academic project. You may also find an answer to your question in the white papers, reports, tools, trainings, and services they have available.

The group of industry experts based use the following guiding principles when publishing their reference Architecture.

- Define protections that enable trust in the cloud.

- Develop cross-platform capabilities and patterns for proprietary and open-source providers.
- Will facilitate trusted and efficient access, administration and resiliency to the customer/consumer.
- Provide direction to secure information that is protected by regulations.
- The Architecture must facilitate proper and efficient identification, authentication, authorization, administration and auditability.
- Centralize security policy, maintenance operation and oversight functions.
- Access to information must be secure yet still easy to obtain.
- Delegate or Federate access control where appropriate.
- Must be easy to adopt and consume, supporting the design of security patterns
- The Architecture must be elastic, flexible and resilient supporting multi-tenant, multi-landlord platforms
- The architecture must address and support multiple levels of protection, including network, operating system, and application security needs.

An overview of the architecture is shown in the diagram from the Cloud Security Alliance. See [Figure 28](#)

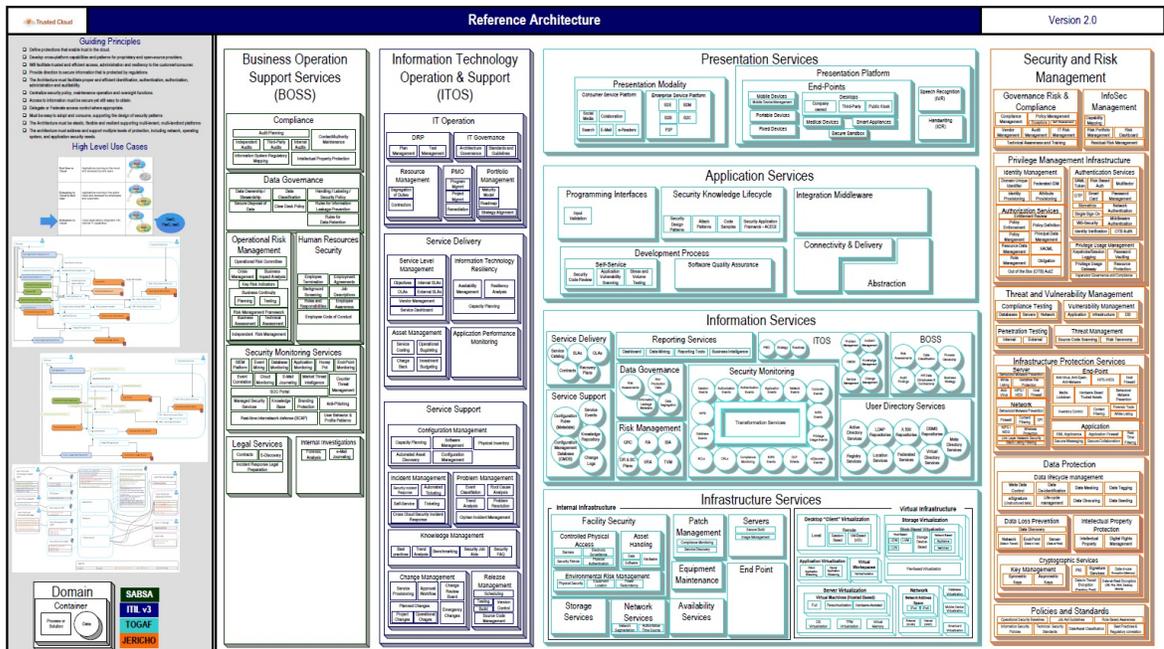


Figure 28: Cloud Security Alliance Reference Architecture [Source](#)

5.1.6 Multicloud Architectures

One of the issues we see today is that it is unrealistic to assume clouds are only provided by one vendor, or that they have all the same interface. Each vendor is advertising their special services to distinguish themselves from the competitors. For the end user and the developer that projects the problem of vendor lockin. However, we need to be aware of efforts that allow an easy of such vendor lockin while for example providing multi cloud solutions. Such solutions integrate multiple vendors and technologies into a single architecture allowing to use multiple cloud vendors at the same time.

5.1.6.1 Cloudmesh Architecture

One of the earliest such tools is Cloudmesh.org, which is lead by von Laszewski. The tool was developed at a time when AWS and Nimbus, and Eucalyptus where predominant players. At that time OpenStack had just transitioned from a NASA project to a community development.

FutureGrid was one of the earliest academic cloud offerings to explore the effectiveness of the different cloud infrastructure solutions. It was clear that a unifying framework and abstraction layer was needed allowing us to utilize them easily. In fact cloudmesh did not only provide a REST based API, but also a commandline shell allowing to switch between clouds with a single variable. It also provided bare metal provisioning before OpenStack even offered it. Through an evolution of developments the current cloudmesh architecture that allows multicloud services is depicted in the next figure. We still distinguish the IaaS level which included not only IaaS Abstractions, but also Containers, and HPC services. Platforms are typically integrated through DevOps that can be hosted on the IaaS. Examples are Hadoop, and Spark The services are exposed through a client API hiding much of the internals to the user. A portal and application services have successfully demonstrated the feasibility of this approach (see [Figure 29](#)).

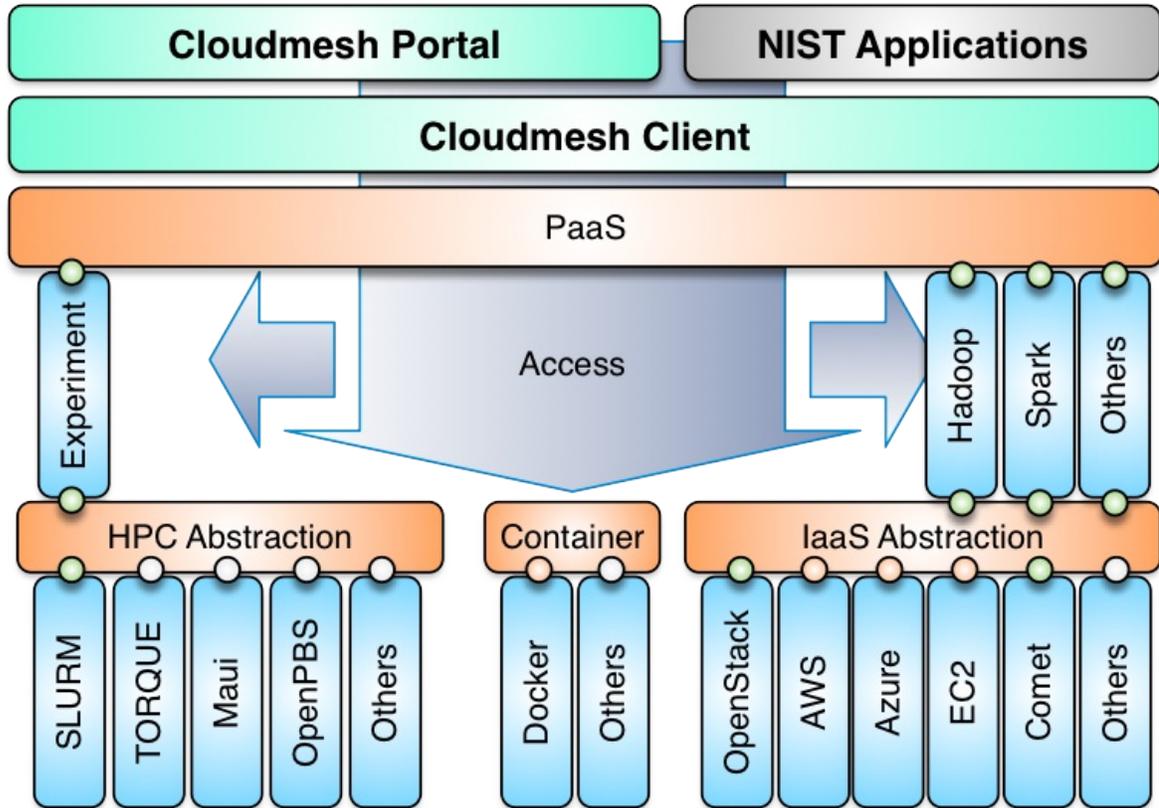


Figure 29: Cloudmesh Arch [1]

Within the hour e516 class we will be developing a modern version of cloudmesh from the ground up by only using python 3 as implementation language, integration of containers, and REST services based on OpenAPI. Local data to manage the different services are hosted in a mongo DB database and exposed through portable containers, so that a single cross-platform environment exists as part of the project deliverables.

Students from e516 can and are in fact expected to participate actively on the development of Cloudmesh v4.0. In addition the OpenAPI service specifications developed for the project will be integrated in Volume 8 of the NIST Big data reference architecture, which is discussed elsewhere.

The advantage of developing such an environment is that we can look at various aspects of cloudcomputing while demonstrating integrated use patterns.

5.1.7 Resources

- [24] <http://www.lifl.fr/iwaise12/presentations/tata.pdf>

- [25] https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
- [26] http://staff.polito.it/alessandro.mantelero/cloud_computing/Sun_Wp2009.pdf
- [27] https://resources.sei.cmu.edu/asset_files/Presentation/2010_017_001_23337
- [28] <https://www.oracle.com/technetwork/articles/entarch/orgeron-top-10-cloud-1957407.pdf>
- [29] <http://www.oracle.com/technetwork/topics/entarch/oracle-wp-cloud-ref-arch-1883533.pdf>
- [30] <https://pdfs.semanticscholar.org/cecd/c193b73ec1e7b42d132b3c340e6dd34f>

5.2 NIST BIG DATA REFERENCE ARCHITECTURE



Learning Objectives

- Obtain an overview of the NIST Big Data Reference Architecture.
 - Understand that you can contribute to it as part of this class.
-
-

One of the major technical areas in the cloud is to define architectures that can work with Big Data. For this reason NIST has worked now for some time on identifying how to create a data interoperability framework. The idea here is that at one point architecture designers can pick services that they can choose to combine them as part of their data pipeline and integrate in a convenient fashion into their solution.

Besides just being a high level description NIST also encourages the verification of the architecture through interface specifications, especially those that are currently under way in Volume 8 of the document series. You have the unique opportunity to help shape this interface and contribute to it. We will provide you not only mechanisms on how you theoretically can do this, but also how you practically can contribute.

As part of your projects in 516 you will need to integrate a significant service

that you can contribute to the NIST document in form of a specification and in form of an implementation.

5.2.1 Pathway to the NIST-BDRA

The Nist Big Data Public Working Group (NBD-PWG) was established as collaboration between industry, academia and government “to create a consensus-based extensible Big Data Interoperability Framework (NBDIF) which is a vendor-neutral, technology- and infrastructure-independent ecosystem” [31]. It will be helpful for Big Data stakeholders such as data architects, data scientists, researchers, implementers to integrate and utilize “the best available analytics tools to process and derive knowledge through the use of standard interfaces between swappable architectural components” [31]. The NBDIF is being developed in three stages:

- Stage 1: “Identify the high-level Big Data reference architecture key components, which are technology, infrastructure, and vendor agnostic,” [31] introduction of the Big Data Reference Architecture (NBD-RA);
- Stage 2: “Define general interfaces between the NBD-RA components with the goals to aggregate low-level interactions into high-level general interfaces and produce set of white papers to demonstrate how NBD-RA can be used” [31];
- Stage 3: “Validate the NBD-RA by building Big Data general applications through the general interfaces.[31]”

Nist has developed the following volumes as listed in *Table: BDRA volumes* that surround the creation of the NIST-BDRA. We recommend that you take a closer look at these documents as in this section we provide a focussed summary with the aspect of cloud computing in mind.

Table: NIST BDRA Volumes

Volumes	Volume	Title
NIST SP1500-1r1	Volume 1	Definitions
NIST SP1500-2r1	Volume 2	Taxonomies

NIST SP1500-3r1	Volume 3	Use Cases and Requirements
NIST SP1500-4r1	Volume 4	Security and Privacy
NIST SP1500-5	Volume 5	Reference Architectures White Paper Survey
NIST SP1500-6r1	Volume 6	Reference Architecture
NIST SP1500-7r1	Volume 7	Standards Roadmap
NIST SP1500-9	Volume 8	Reference Architecture Interface (new)
NIST SP1500-10	Volume 9	Adoption and Modernization (new)

5.2.2 Big Data Characteristics and Definitions

Volume 1 of the series introduces the community to common definitions that are used as part of the field of Big data. This includes the analysis of characteristics such as volume, velocity, variety, variability and the use of structures and unstructured data. As part of the field of data science and engineering it lists a number of areas that are to be believed to be essential including that they must master including data structures, parallelism, metadata, flow rate, visual communication. In addition we believe that an additional skill set must be prevalent that allows a data engineer to deploy such technologies onto actual systems.

We have submitted the following proposal to NIST:

3.3.6. Deployments:

A significant challenge exists for data engineers to develop architectures and their deployment implications. The volume of data and the processing power needed to analysis them may require many thousands of distributed compute resources. They can be part of private data centers, virtualized with the help of virtual machines or containers and even utilize serverless computing to focus integration of Big Data Function as a Service based architectures. As such architectures are assumed to be large community standards such as leveraging DevOps will be necessary for the engineers to setup and manage such architectures. This is especially important with the swift development of the field that may require rolling updates without interruption of the services offered.

This addition reflects the newest insight into what a data scientist needs to know and the newest job trends that we observed.

To identify what big data is we find the following characteristics

Volume: Big for data means lots of bytes. This could be achieved in many different ways. Typically we look at the actual size of a data set, but also how this data set is stored for example in many thousands of smaller files that are part of the data set. It is clear that in many of such cases analysis of a large volume of data will impact the architectural design for storage, but also the workflow on how this data is processed.

Velocity: We see often that big data is associated with high data flow rates caused by for example streaming data. It can however also be caused by functions that are applied to large volumes of data and need to be integrated quickly to return the result as fast as possible. Needs for real time processing as part of the quality of service offered contribute also to this. Examples of IoT devices that integrate not only data in the cloud, but also on the edge need to be considered.

Variety: In today's world we have many different data resources that motivate sophisticated data mashup strategies. Big data hence not only deals with information from one source but a variety of sources. The architectures and services utilized are multiple and needed to enable automated analysis while incorporating various data source.

Another aspect of variety is that data can be structured or unstructured. NIST finds this aspect so important that they included its own section for it.

Variability: Any data over time will change. Naturally that is not an exception in Big data where data may be a time to live or needs to be updated in order not to be stale or obsolete. Hence one of the characteristics that big data could exhibit is that its data be variable and is prone to changes.

In addition to these general observations we also have to address important characteristics that are attached with the Data itself. This includes

Veracity: Veracity refers to the accuracy of the data. Accuracy can be increased

by adding metadata.

Validity: Refers to data that is valid. While data can be accurately measured, it could be invalid by the time it is processed.

Volatility: Volatility refers to the change in the data values over time.

Value: Naturally we can store lots of information, but if the information is not valuable then we may not need to store it. This is recently been seen as a trend as some companies have transitioned data sets to the community as they do not provide value to the service provider to justify its prolonged maintenance.

In other cases the data has become so valuable and that the services offered have been reduced for example as they provide too many resource needs by the community. A good example is Google scholar that used to have much more liberal use and today its services are significantly scaled back for public users.

5.2.3 Big Data and the Cloud

While looking at the characteristics of Big Data it is obvious that Big data is on the one hand a motivator for cloud computing, but on the other hand existing Big Data frameworks are a motivator for developing Big Data Architectures a certain way.

Hence we have to always look from both sides towards the creation of architectures related to a particular application of big data.

This is also motivated by the rich history we have in the field of parallel and distributed computing. For a long time engineers have dealt with the issue of *horizontal scaling*, which is defined by adding more nodes or other resources to a cluster. Such resources may include

- shared disk file systems,
- distributed file systems,
- distributed data processing and concurrency frameworks, such as Concurrent sequential processes, workflows, MPI, map/reduce, or shared memory,
- resource negotiation to establish quality of service,

- data movement,
- and data tiers (as showcased in high energy physics Ligo [32] and Atlas)

In addition to the horizontal scaling issues we also have to worry about the *vertical scaling* issues, this is how the overall system architecture fits together to address an end-to-end use case. In such efforts we look at

- interface designs,
- workflows between components and services,
- privacy of data and other security issues,
- reusability within other use-cases.

Naturally the cloud offers the ability to *cloudify* existing relational databases as cloud services while leveraging the increased performance and special hardware and software support that may be otherwise unaffordable for an individual user. However we see also the explosive growth of non sql databases because some of them can more effectively deal with the characteristics of big data than traditional mostly well structured data bases. In addition many of these frameworks are able to introduce advanced capability such as distributed and reliable service integration.

Although we have been used to the term cloud while using virtualized resources and the term Grid by offering a network of supercomputers in a virtual organization, We should not forget that Cloud service providers also offer High performance computers resources for some of their most advanced users.

Naturally such resources can be used not only for numerical intensive computations but also for big data applications as the Physics community has demonstrated.

5.2.4 Big Data, Edge Computing and the Cloud

When looking at the number of devices that are being added daily to the global IT infrastructure we observe that cellphones and soon Internet of Things (IoT) devices will produce the bulk of all data. However not all data will be moved to the cloud and lots of data will be analyzed locally on the devices or even not being considered to be uploaded to the cloud either because it project to low or to high value to be moved. However a considerable portion will put new

constraints on our services we offer in the cloud and any architecture addressing this must be properly deal with scaling early on in the architectural design process.

5.2.5 Reference Architecture

Next we present the Big data reference architecture. It is Depicted in [Figure 30](#). According to the document (Volume 2) the five main components representing the central roles include

- System Orchestrator: Defines and integrates the required data application activities into an operational vertical system;
- Data Provider: Introduces new data or information feeds into the Big Data system;
- Big Data Application Provider: Executes a life cycle to meet security and privacy requirements as well as System Orchestrator-defined requirements;
- Big Data Framework Provider: Establishes a computing framework in which to execute certain transformation applications while protecting the privacy and integrity of data; and
- Data Consumer: Includes end users or other systems who use the results of the Big Data Application Provider.

In addition we recognize two fabrics layers:

- Security and Privacy Fabric
- Management Fabric

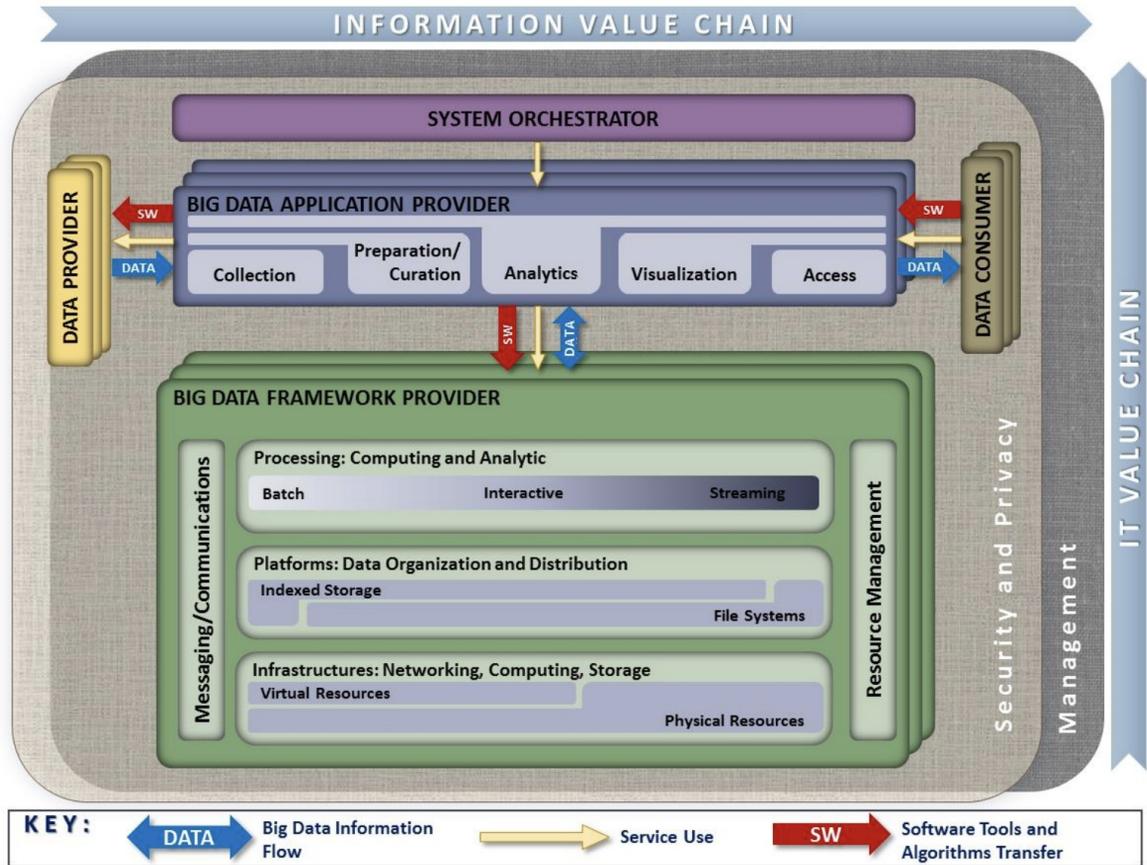


Figure 30: NIST-BDRA (see [Volume 2](#))

While looking at the actors depicted in [Figure 31](#) we need to be aware that in each of the categories a service can be added. This is an important distinction to the original depiction in the definition as it is clear that an automated service could act in behalf of the actors listed in each of the categories.

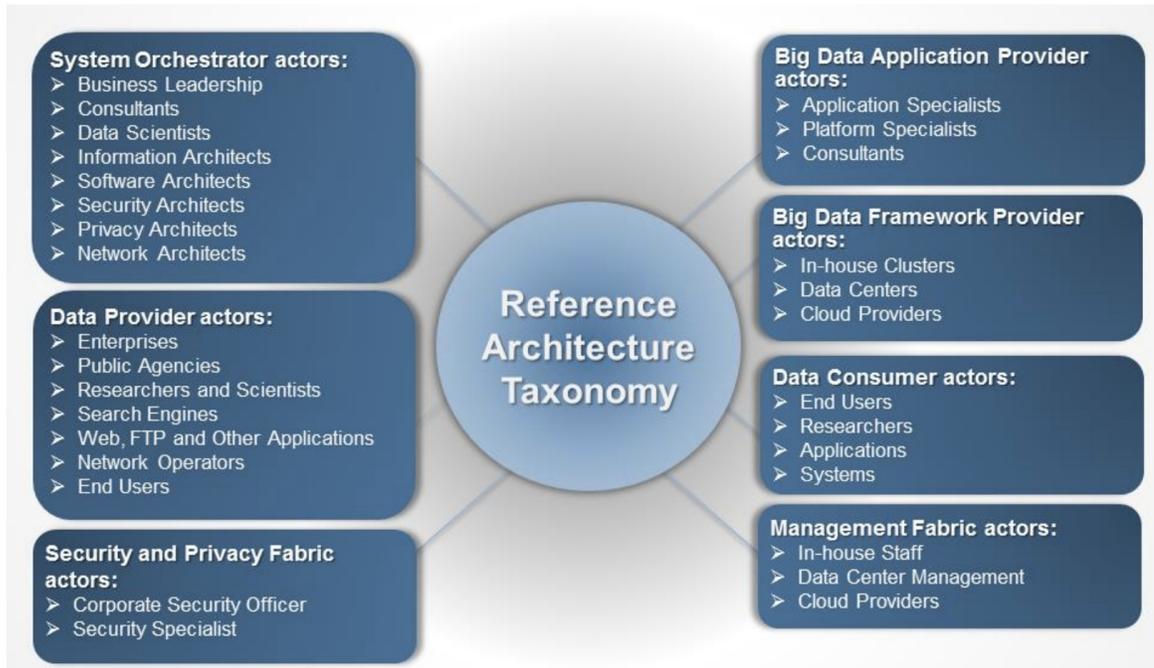


Figure 31: NIST Roles (see [Volume 2](#))

For a detailed definition which is beyond the scope of this document we refer to the Volume 2 of the documents.

5.2.6 Framework Providers

Traditionally cloud computing has started with offering IaaS, followed by PaaS and SaaS. We see the IaaS reflected in three categories for big data:

1. Traditional compute and network resources including virtualization frameworks
2. Data Organization and Distribution systems such as offered in Indexed Storage and File Systems
3. Processing engines offering batch, interactive, and streaming services to provide computing and analytics activities

Messaging and communication takes place between these layers while resource management is used to address efficiency.

Frameworks such as Spark and Hadoop include components from multiple of these categories to create a vertically integrated system. Often they are offered by a service provider. However, one needs to be reminded that such offerings may

not be tailored to the individual use-case and inefficiencies could be prevalent because the service offer is outdated, or it is not explicitly tuned to the problem at hand.

5.2.7 Application Providers

The underlying infrastructure is reused by big data application providers supporting services and tasks such as

- Data collections
- Data curation
- Data Analytics
- Data Visualization
- Data Access

Through the interplay between these services data consumers and data producers can be served.

5.2.8 Fabric

Security and general management are part of the governing fabric in which such an architecture is deployed.

5.2.9 Interface definitions

The interface definitions for the BDRA are specified in Volume 8. We are in the second phase of our document specification while we switch from our pure Resource description to an OpenAPI specification. Before we can provide more details we need to introduce you to REST which is an essential technology for many modern cloud computing services.

5.3 THE Y-SCHEDULING ARCHITECTURE VIEW

Previous architecture views were concerned about high level interactions such as the view projected by NIST that introduces a service model based on infrastructure, platform and application.

However such a view may provide too little detail to develop meaningful services that use cloud resources in a multi-cloud environment. For this reason von Laszewski has devised a Y diagram that showcases the interaction between the different layers more clearly we like to refer the reader to the Y-cloud scheduling diagram.

In this taxonomy we are concerned about how resources are placed on physical models and are interconnected with each other to facilitate for example scheduling algorithms. [@{fig:graph-y}](#) depicts the different models integrated in the Taxonomy. It includes:

- **Physical Model:** that represents major physical resource layers to enable a hierarchical scheduling strategy across multiple data centers, data centers, racks, servers, and computing cores.
- **Resource Model:** that represents models that the scheduling algorithm addresses including containers and functions, virtual machines and jobs, virtual clusters, provider managed resources, and multi-region provider managed resources.
- **Connectivity Model:** that introduces a connectivity between components when addressing scheduling. This includes components such as memory, processes, connectivity to distributed resources, hyper-graphs to formulate hierarchies of provider based resources, and region enhanced hyper-graphs. The connectivity model allows us to leverage classical scheduling algorithms while applying such models and leveraging established or new scheduling algorithms for these models.

To for example develop scheduling algorithms a layered approach can be chosen to separate concerns between different layers while utilizing an abstracting services the models project in each layer.

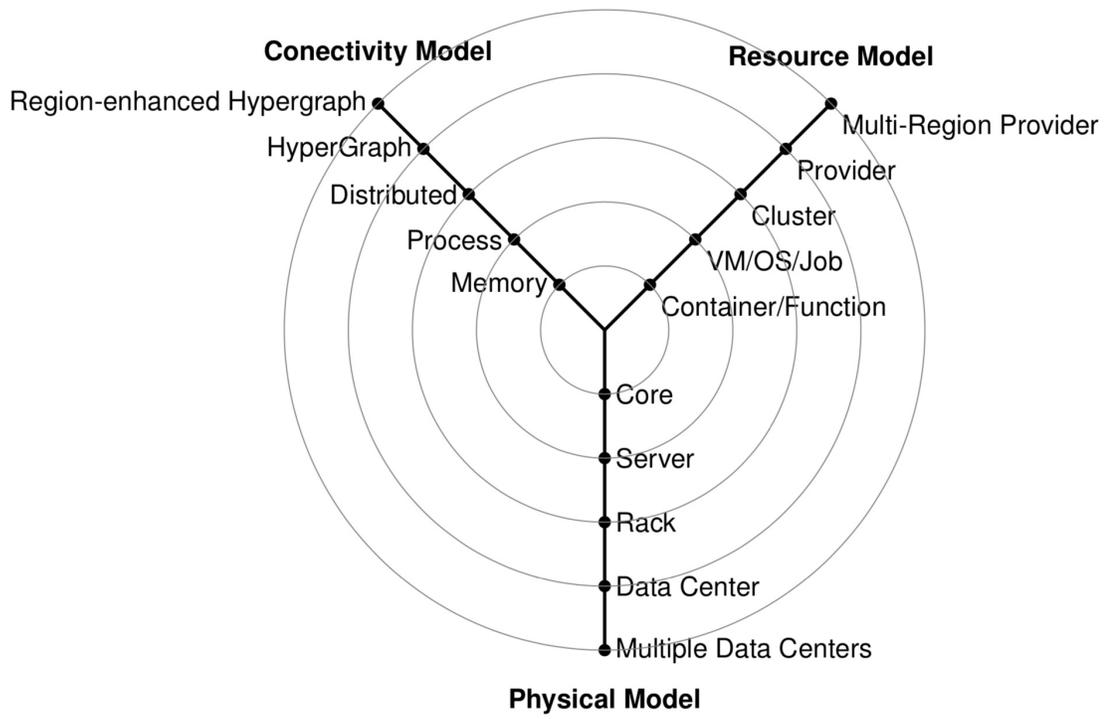


Figure 32: Von Laszewski's Y-scheduling Cloud Architecture view

6 REST

6.1 INTRODUCTION TO REST



Learning Objectives

- Understand REST Services.
 - Understand OpenAPI.
 - Develop REST services in Python using Eve.
 - Develop REST services in Python using OpenAPI with swagger.
-
-

REST stands for **RE**presentational **St**ate **T**ransfer. REST is an architecture style for designing networked applications. It is based on stateless, client-server, cacheable communications protocol. In contrast to what some others write or say, REST is not a *standard*. Although not based on http, in most cases, the HTTP protocol is used. In that case, RESTful applications use HTTP requests to (a) post data while creating and/or updating it, (b) read data while making queries, and (c) delete data.

REST was first introduced in a [thesis from Roy T. Fielding \[33\]](#).

Hence REST can use HTTP for the four CRUD operations:

- **C**reate resources
- **R**ead resources
- **U**ppdate resources
- **D**elate resources

As part of the HTTP protocol we have methods such as GET, PUT, POST, and DELETE. These methods can then be used to implement a REST service. This is not surprising as the HTTP protocol was explicitly designed to support these operations. As REST introduces collections and items we need to implement the CRUD functions for them. We distinguish single resources and collection of resources. The semantics for accessing them is explained next illustrating how to

implement them with HTTP methods (See [REST on Wikipedia](#) [34]).

6.1.0.1 Collection of Resources

Let us assume the following URI identifies a collection of resources

`http://.../resources/`

than we need to implement the following CRUD methods:

GET

List the URIs and perhaps other details of the collections members

PUT

Replace the entire collection with another collection.

POST

Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.

DELETE

Delete the entire collection.

6.1.0.2 Single Resource

Let us assume the following URI identifies a single resource in a collection of resources

`http://.../resources/item42`

than we need to implement the following CRUD methods:

GET

Retrieve a representation of the addressed member of the collection, expressed in an appropriate internet media type.

PUT

Replace the addressed member of the collection, or if it does not exist, create it.

POST

Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.

DELETE

Delete the addressed member of the collection.

6.1.0.3 REST Tool Classification

Due to the well defined structure that REST provides a number of tools have been created that manage the creation of the specification for rest services and their programming. We distinguish several different categories:

REST Specification Frameworks:

These are frameworks that help defining rest service through specifications to generate REST services in a language and framework independent way. This includes for example Swagger 2.0 [35], OpenAPI 3.0 [36], and RAML [37].

REST programming language support:

These tools and services are targeting a particular programming language. Such tools include Flask Rest [38], and Django Rest Services [39], some of which we will explore in more detail.

REST documentation based tools:

These tools are primarily focusing on documenting REST specifications. Such tools include Swagger [40], which we will explore in more detail.

REST design support tools:

These tools are used to support the design process of developing REST services while abstracting on top of the programming languages and define reusable specifications that can be used to create clients and servers for particular technology targets. Such tools include also swagger [40] as additional tools are available that can generate code from OpenAPI specifications [41], which we will explore in more detail.

A list of such efforts is available at [OpenAPI Tools](#) [42]

6.2 OPENAPI 3.0

6.2.1 REST Specifications

RESTful services have undoubtedly become the de-facto software architectural style for creating Web services. A REST API specification would defines the attributes and constraints to be used in the web service. There have been multiple specifications that have been in use such as [OpenAPI \(formally called Swagger\)](#) [36], [RAML](#) [37], [tinyspec](#) [43], and [API Blueprint](#) [44].

6.2.1.1 OPENAPI

Over the years, Open API specification has become the most popular with a much larger community behind it. Therefore, this section would focus on the latest specification, [OpenAPI 3.0 \(OAS 3.0\)](#) [45].

According to the [OAS documentation](#) [46], it allows users to,

- Describe endpoints and operations on each endpoint
- Specify operation parameters, inputs, and outputs for each operation
- Handle authentication
- Describe contact, license, terms of use and other information

API specifications can be written in YAML or JSON. OAS also comes with a rich toolkit that includes [Swagger Editor](#) [47], [Swagger UI](#) [48] and [Swagger Codegen](#) [41], that creates an end-to-end development environment, even for the users who are new to OAS.

Section [OpenAPI Specification](#) details more on the OAS 2.0 specification.

6.2.1.1.1 Open API 3.0 Specification (OAS 3.0)

OAS 3.0 key definitions can be depicted in the following figure.

OpenAPI 3

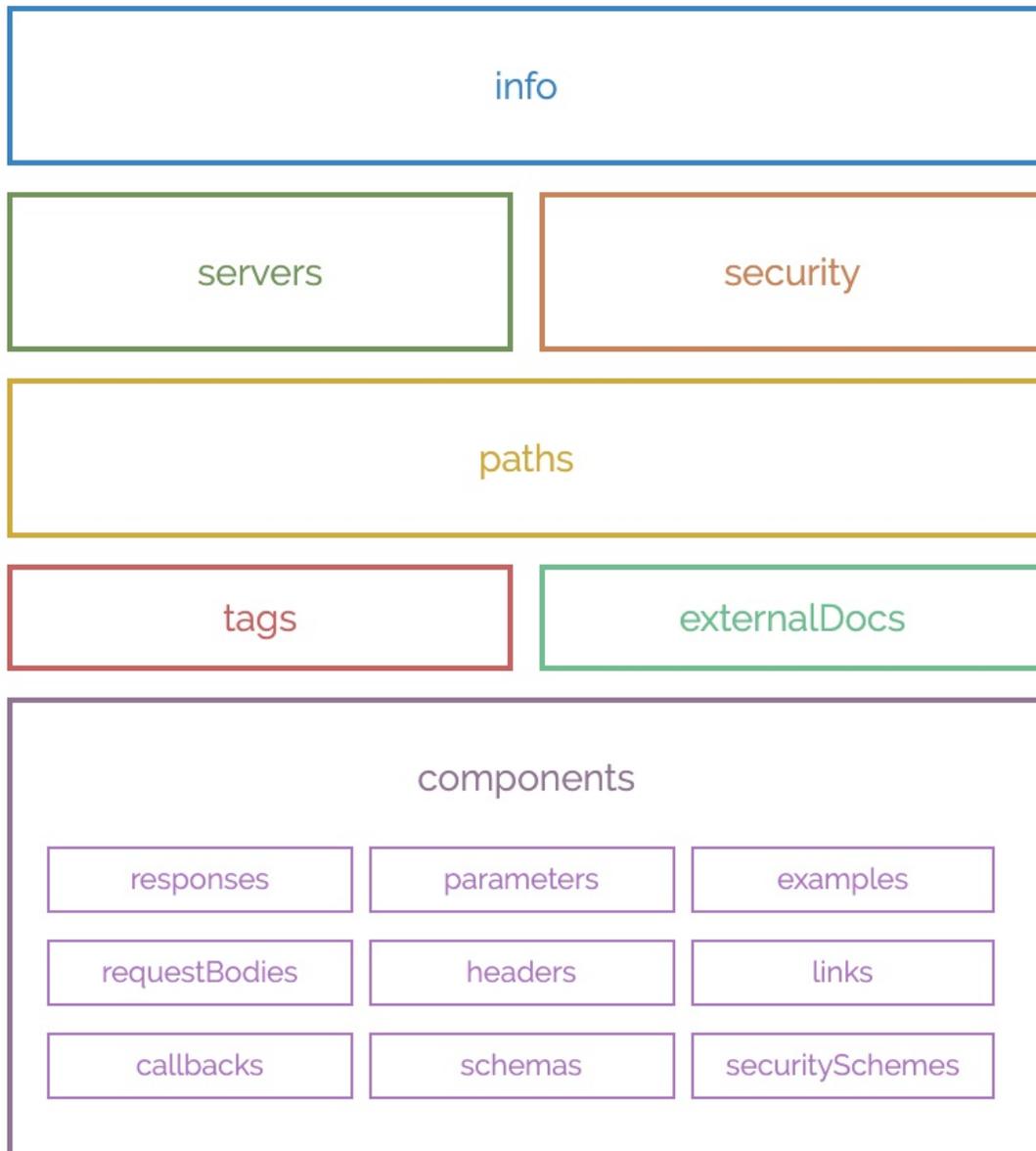


Figure 33: Components of OAS 3.0 [Source](#)

Basic structure of the definitions would look like this. The sample REST service, exposes `http://localhost:8080/cloudmesh` basepath. Under that base path, an endpoint has been exposed as `cloudmesh/cpu` which would return CPU information of the server. It uses a predefined schema to return the results, which is defined under the `components/schemas`. See the Section [OpenAPI REST](#)

[Service via Introspection](#) for the detailed example.

```
openapi: 3.0.2
info:
  title: cpuinfo
  description: A simple service to get cpuinfo as an example of using OpenAPI 3.0
  license:
    name: Apache 2.0
    version: 0.0.1

servers:
  - url: http://localhost:8080/cloudmesh

paths:
  /cpu:
    get:
      summary: Returns cpu information of the hosting server
      operationId: cpu.get_processor_name
      responses:
        '200':
          description: cpu info
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/cpu"

components:
  schemas:
    cpu:
      type: "object"
      required:
        - "model"
      properties:
        model:
          type: "string"
```

6.2.1.1.1.1 Definitions

Metadata:

OAS 3.0 requires a specification definition at the start under the *openapi* field.

```
openapi: 3.0.2
```

Next, metadata can be specified under *info* field such as *title*, *version*, *description*, etc. Additionally, license, contact information can also be specified. *title* and *version* are mandatory fields under *info*.

```
info:
  title: cpuinfo
  description:
    A simple service to get cpuinfo as an example of using OpenAPI 3.0
  license:
    name: Apache 2.0
  version: 0.0.1
```

Servers:

The *servers* section defines the server URLs with the basepath. Optionally, a *description* can be added.

```
servers:
  - url: http://localhost:8080/cloudmesh
    description: Cloudmesh server basepath
```

Paths:

The *paths* section specifies all the endpoints exposed by the API and the HTTP operations supported by these endpoints.

```
paths:
  /cpu:
    get:
      summary: Returns cpu information of the hosting server
      operationId: cpu.get_processor_name
      responses:
        '200':
          description: cpu info
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/cpu"
```

Operation ID:

When using introspection for REST services (using Connexion), we would need to point to the operation that would ultimately carry out the request. This operation is specified by the *operationID*.

```
...
paths:
  /cpu:
...
    operationId: cpu.get_processor_name
```

Parameters:

If the service endpoint accepts URL parameters (ex: `/cpu/cache/{cache_level}` or `/cpu?arch=x86`), headers or cookies, those can also be specified under a *path*.

```
paths:
  /cpu/cache/{cache_level}:
    get:
      summary: Returns the cache size of the specified level
      parameters:
        - name: cache_level
          in: path
          required: true
          description: Parameter description in CommonMark or HTML.
          schema:
            type: string
            minimum: 1
      responses:
        '200':
          description: OK
```

Request Body:

When a request is sent with a body, such as *POST*, that will be specified in the

requestBody under a *path*.

```
paths:
  /upload:
    post:
      summary: upload input
      requestBody:
        content:
          multipart/form-data:
            schema:
              type: object
              properties:
                file:
                  type: string
                  format: binary
      responses:
        '200':
          description: OK
```

Responses:

For each path, *responses* can be specified with the corresponding status codes such as 200 OK or 404 Not Found. A response may return a response body, that can be defined under *content*.

```
...
paths:
  /cpu:
...
  responses:
    '200':
      description: cpu info
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/cpu"
```

Schemas:

The *components/schemas* section allows users to define schemas for inputs or outputs that can be referenced via *\$ref* tag.

```
...
paths:
  /cpu:
...
  responses:
    '200':
      description: cpu info
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/cpu"
...

components:
  schemas:
    cpu:
      type: "object"
      required:
        - "model"
      properties:
        model:
          type: "string"
```

Authentication:

Under the *components* sections, *securitySchemes* can also be specified such as Basic Auth.

```
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic

security:
  - BasicAuth: []
```

According to the current OAS 3.0, supported authentication methods are,

- HTTP authentication: Basic, Bearer, and so on.
- API key as a header or query parameter or in cookies
- OAuth2
- OpenID Connect Discovery

6.2.1.2 RAML

[RAML](#) [37] (RESTful API Modeling Language) is a specification proposed in 2013 and it is based on YAML format. The specification is managed by the RAML Worker Group. It initially came out as a proprietary vendor language (specification) but later was open-sourced. As of Sep 2019, the latest specification is [RAML 1.0](#) [49].

Following is an example RAML specification from the [RAML.org](#)

```
##%RAML 1.0
title: Hello world # required title

/helloworld: # optional resource
  get: # HTTP method declaration
    responses: # declare a response
      200: # HTTP status code
        body: # declare content of response
          application/json: # media type
            type: | # structural definition of a response (schema or type)
              {
                "title": "Hello world Response",
                "type": "object",
                "properties": {
                  "message": {
                    "type": "string"
                  }
                }
              }
            example: | # example of how a response looks
              {
                "message": "Hello world"
              }
```

In the current context, the industry seems to be adopting OpenAPI more than RAML. Consequently, some of the main contributors of RAML such as MuleSoft have joined the Open API Initiative since 2017. Hence, it is safe to conclude that Open API would be the dominant REST API specification in the web services domain.

Furthermore, there are tools available to switch between the specifications, such as [RAML Web API Parser](#) which can convert RAML to Open API and vice-versa.

6.2.1.3 API Blueprint

[API Blueprint](#) [44] is another specification available currently which uses Markdown syntax. As of Sep, 2019 the latest version available is 1A-rev9.

6.2.1.4 JsonAPI

As the name suggests, [JSON API](#)[50] attempts to leverage web services specifications using JSON format. It reached a stable version 1.0 in May, 2015, but there have been no revisions since then.

6.2.1.5 Tinyspec

[Tinyspec](#)[43] is a lightweight alternative to Open API. It has not being able to enter into the mainstream thus far, unfortunately.

6.2.1.6 Tools

There are a number of tools available in the REST webservice specification domain. A classification of REST tools can be found in the [Section 6.1.1.3](#) section.

6.2.1.6.1 Connexion

[Connexion](#)[51] is one such tool that is based on Open API and it is widely used in the Python environment. This framework allows users to define webservice

in Open API and then map those services to Python functions conveniently. We would be using Connexion when we create REST services using introspection [Section 6.2.2](#).

Here is an example from the [Connexion official website](#) [51].

```
openapi: "3.0.0"

info:
  title: Hello World
  version: "1.0"
servers:
  - url: http://localhost:9090/v1.0

paths:
  /greeting/{name}:
    post:
      summary: Generate greeting
      description: Generates a greeting message.
      operationId: hello.post_greeting
      responses:
        200:
          description: greeting response
          content:
            text/plain:
              schema:
                type: string
                example: "hello dave!"
      parameters:
        - name: name
          in: path
          description: Name of the person to greet.
          required: true
          schema:
            type: string
            example: "dave"
```

This service would map to the following *post_greeting* Python function.

```
import connexion

def post_greeting(name: str) -> str:
    return 'Hello {name}'.format(name=name)

if __name__ == '__main__':
    app = connexion.FlaskApp(__name__, port=9090, specification_dir='openapi/')
    app.add_api('helloworld-api.yaml', arguments={'title': 'Hello World Example'})
    app.run()
```

6.2.2 OpenAPI 3.0 REST Service via Introspection

The simplest way to create an OpenAPI service is to use the connexion service and read in the specification from its yaml file. It will than be introspected and dynamically methods are created that are used for the implementation of the server.

The full example for this is available in

- <https://github.com/cloudmesh-community/book/tree/master/examples/rest/cpu>

An extensive documentation is available at

- <https://media.readthedocs.org/pdf/connexion/latest/connexion.pdf>

This example will return dynamically the cpu information of a computer to demonstrate how simple it is to generate in python a REST service from an OpenAPI specification.

Our requirements.txt file includes

```
flask
connexion[swagger-ui]
```

as dependencies. The `server.py` file simply contains the following code:

```
from flask import jsonify
import connexion

# Create the application instance
app = connexion.App(__name__, specification_dir="./")

# Read the yaml file to configure the endpoints
app.add_api("cpu.yaml")

# create a URL route in our application for "/"
@app.route("/")
def home():
    msg = {"msg": "It's working!"}
    return jsonify(msg)

if __name__ == "__main__":
    app.run(port=8080, debug=True)
```

This will run our REST service under the assumption we have a `cpu.yaml` and a `cpu.py` files as our yaml file calls out methods from `cpu.py`

The yaml file looks as follows

```
openapi: 3.0.2
info:
  title: cpuinfo
  description: A simple service to get cpuinfo as an example of using OpenAPI 3.0
  license:
    name: Apache 2.0
  version: 0.0.1

servers:
  - url: http://localhost:8080/cloudmesh

paths:
  /cpu:
    get:
      summary: Returns cpu information of the hosting server
```

```

    operationId: cpu.get_processor_name
    responses:
      '200':
        description: cpu info
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/cpu"

components:
  schemas:
    cpu:
      type: "object"
      required:
        - "model"
      properties:
        model:
          type: "string"

```

Here we simply implement a get method and associate it with the URL /cpu. The operationid, defines the method that we call which as we used the local directory is included in the file `cpu.py`. This is controlled by the prefix in the operation id.

A very simple function to return the cpu information is defined in `cpu.py` which we list next

```

import os, platform, subprocess, re
from flask import jsonify

def get_processor_name():
    if platform.system() == "Windows":
        p = platform.processor()
    elif platform.system() == "Darwin":
        command = "/usr/sbin/sysctl -n machdep.cpu.brand_string"
        p = subprocess.check_output(command, shell=True).strip().decode()
    elif platform.system() == "Linux":
        command = "cat /proc/cpuinfo"
        all_info = subprocess.check_output(command, shell=True).strip().decode()
        for line in all_info.split("\n"):
            if "model name" in line:
                p = re.sub(".*model name.*:", "", line, 1)
    else:
        p = "cannot find cpuinfo"
    pinfo = {"model": p}
    return jsonify(pinfo)

```

We have implemented this function to return a jsonified information from the dict pinfo.

To simplify working with this example, we also provide a makefile for OSX that allows us to call the server and the call to the server in two different terminals

```

define terminal
  osascript -e 'tell application "Terminal" to do script "cd $(PWD); $1"'
endef

install:
  pip install -r requirements.txt

demo:
  $(call terminal, python server.py)
  sleep 3
  @echo "======"

```

```
@echo "Get the info"
@echo "=====
curl http://localhost:8080/cloudmesh/cpu
@echo
@echo "=====
```

When we call

```
make demo
```

our demo is run.

6.2.2.1 Verification

It is important to be able to verify if a yaml file is correct. To identify this, the easiest method is to use the swagger editor. There is an online version available at:

- <https://editor.swagger.io/>

Go to the Web site, remove the current petstore example and simply paste your yaml file in it. Debug messages will be helping you to correct things.

A terminal based command may also be helpful, but is a bit difficult to read.

```
$ connexion run cpu.yaml --stub --debug
```

6.2.2.2 Swagger-UI

Swagger comes with a convenient UI to invoke REST API calls using the web browser rather than relying on the curl commands.

Once the request and response definitions are properly specified, you can start the server by,

```
$ python server.py
```

Then the UI would also be spawned under the service URL *http://[service url]/ui/*

Example: <http://localhost:8080/cloudmesh/ui/>

6.2.2.3 Mock service

In some cases it may be useful to develop the API without having yet developed methods that you call with the OperationI. In this case it is useful to run a mock service. YOU can invoce such a service with

```
$ connexion run cpu.yaml --mock=all -v
```

6.2.2.4 Exercise

OpenAPI.Conexion.1:

Modify the makefile so it works also on ubuntu, but do not disable the ability to run it correctly on OSX. Tip use if's in makefiles base on the OS. You can look at the makefiles that create this book as example. find alternatives to sarting a terminal in Linux.

OpenAPI.Conexion.2:

Modify the makefile so it works also on Windows 10, but do not disable the ability to run it correctly on OSX. Tip use ifs in makefiles. You can look at the makefiles that create this book as example. Find alternatives to start a powershell or cmd.exe in windows. Maybe you need to use gitbash.

OpenAPI.Conexion.3:

Implement a swagger specification of an issue related to the NIST BDRA. Implement it. Please remember this could prepare you for a project good topics include:

- *virtual compute service interfacing with aws, azure, google or openstack*
- *virtual directory service interfacing with google drive, box, github, iCloud, ftp, scp, and others*

As there are so many possibilities to contribute, come up in class with one specification and than implement it for different providers. The difficulty here is that it is not done for one IaaS, but for all of them and all can be integrated.

This exercise is typically growing to be part of your class project.

OpenAPI.Conexion.4:

Develop instructions on how to integrate the OpenAPI service framework in a WSGI based Web service. Chose a service you like so that the service could run in production.

OpenAPI.Conexion.5:

Develop instructions on how to integrate the OpenAPI service framework in Tornado so the service could run in production.

6.2.3 REST AI services Example

This is a more involved example which uses OpenAPI 3.0 specification to invoke  [K-means Clustering](#) routine in scikit-learn [52]. Scikit-learn k-means user-guide can be found Scikit-learn K-Means package [53].

This involves the following.

- Upload a file with points to create the k-means clustering model.
- Method to call scikit-learn KMeans module
- Upload a file with points that need to be predicted and return a file with the predicted cluster IDs.
- Additionally, scikit-learn KMeans module provides routines to get the cluster centers, labels, etc. which can also be exposed as REST services.

To create the REST services, we would be using OpenAPI 3.0 REST service via introspection.

6.2.3.1 Service Endpoints/ Paths

6.2.3.1.1 Path *kmeans/upload*

A POST request with a file containing points to create the k-means clustering model. POST content would be *multipart/form-data*.

For an example consider the following 6 points in XY dimensions,

```
1, 2
1, 4
1, 0
10, 2
10, 4
10, 0
```

Curl command:

```
$ curl -X POST "http://localhost:8080/kmeans/upload" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-F "file=@model.csv;type=text/csv"
```

Service implementation would look like this. File content will be received as a [werkzeug.datastructures.FileStorage](#) object in *Flask*, which can be used to stream into the filesystem. Backend keeps two dicts to map Job ID to file and vice-versa (*inputs* and *inputs_r*).

```
def upload_file(file=None):
    filename = file.filename

    in_file = INPUT_DIR + '/' + filename
    if not os.path.exists(in_file):
        file.save(in_file) # save the input file

    if in_file not in inputs_r:
        job_id = len(inputs)
        inputs.update({job_id: in_file})
        inputs_r.update({in_file: job_id})
    else:
        job_id = inputs_r[in_file]

    return jsonify({'job_id': job_id, 'filename': filename})
```

If the request is successful, a *JSON* will be returned with the file name and the associated job ID. Job ID can be considered ID that would connect, inputs to the models and the predicted outputs.

```
{
  "filename": "model.csv",
  "job_id": 0
}
```

6.2.3.1.2 Path *kmeans/fit*

A POST request with a *JSON* body containing Job ID and model parameters that need to be passed on to the scikit-learn KMeans model initialization such as, number of clusters (*n_clusters*), maximum iterations (*max_iter*), etc.

Example:

```
{
```

```
"job_id": 0,
"model_params": {
  "n_clusters": 3
}
}
```

curl command:

```
$ curl -X POST "http://localhost:8080/kmeans/fit" \
-H "accept: text/csv" \
-H "Content-Type: application/json" \
-d "{\"job_id\":0,\"model_params\":{\"n_clusters\":3}}"
```

Service implementation looks like this. POST request body will be populated as a dict and passed on to the method by Flask (*body*). Once the model is fitted, it will be put into a in memory dict (*models*) against its Job ID. Labels will be written to disk as a file, and the content will be returned as a CSV.

```
def kmeans_fit(body):
    print(body)

    job_id = body['job_id']

    if job_id not in inputs or not os.path.exists(inputs[job_id]):
        abort(500, "input file missing for job id " + str(job_id))
        return
    in_file = inputs[job_id]

    X = np.genfromtxt(in_file, delimiter=",") # create the model

    params = dict(default_model_params)
    params.update(body['model_params'])

    kmeans = KMeans(**params).fit(X)

    models.update({job_id: kmeans}) # add the model in to the dict

    labels = OUTPUT_DIR + "/" + str(job_id) + ".labels"
    np.savetxt(labels, kmeans.labels_, delimiter=",")

    return send_file(labels)
```

The response CSV file will be returned with the corresponding labels for the input points.

```
1.0000000000000000e+00
1.0000000000000000e+00
1.0000000000000000e+00
0.0000000000000000e+00
0.0000000000000000e+00
2.0000000000000000e+00
```

6.2.3.1.3 Path *kmeans/predict*

A POST request with a file containing the points to be predicted and the corresponding Job ID as *multipart/form-data*.

```
job_id=0
```

Points to be predicted

```
0, 0  
12, 3
```

curl command:

```
$ curl -X POST "http://localhost:8080/kmeans/predict" \  
-H "accept: text/csv" \  
-H "Content-Type: multipart/form-data" \  
-F "job_id=0" \  
-F "file=@predict.csv;type=text/csv"
```

Service implementation looks like this. Note that there is a strange behavior in *Flask* with *Connexion* where the file content will be passed on to the *file* object as a [werkzeug.datastructures.FileStorage](#) object but the Job ID is passed as a dict to *body* object.

```
def kmeans_predict(body, file=None):  
    job_id = int(body['job_id'])  
  
    if job_id in models:  
        p_file = OUTPUT_DIR + '/' + str(job_id) + '.p'  
        file.save(p_file)  
  
        p = np.genfromtxt(p_file, delimiter=',') # read the predictions  
  
        result = models[job_id].predict(p)  
  
        print(result)  
  
        res_file = OUTPUT_DIR + "/" + str(job_id) + ".out"  
        np.savetxt(res_file, result, delimiter=",")  
  
        return send_file(res_file)  
  
    else:  
        abort(500, "model not found for job id " + str(job_id))  
    return
```

The response would send out the corresponding labels of the passed points as a CSV file.

```
1.0000000000000000e+00  
0.0000000000000000e+00
```

6.2.3.2 Files

Files of this example can be found [here](#).

- Open API 3 service definitions - [api.yaml](#)
- Flask server - [server.py](#)
- Kmeans service implementation - [kmeans.py](#)

- Python requirements - [requirements.txt](#)
- Example files [model.csv](#) and [predict.csv](#)

6.2.3.3 Running the example

- Go to the example directory.
- Activate the Python3 venv used for *Cloudmesh*
- Install requirements.txt

```
$ pip install -r requirements.txt
```

- Start the server

```
$ python server.py
```

- Upload a file

```
$ curl -X POST "http://localhost:8080/kmeans/upload" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-F "file=@model.csv;type=text/csv"
```

- Fit the kmeans model

```
$ curl -X POST "http://localhost:8080/kmeans/fit" \
-H "accept: text/csv" \
-H "Content-Type: application/json" \
-d "{\"job_id\":0,\"model_params\":{\"n_clusters\":3}}"
```

- Predict using the fitted kmeans model

```
$ curl -X POST "http://localhost:8080/kmeans/predict" \
-H "accept: text/csv" \
-H "Content-Type: multipart/form-data" \
-F "job_id=0" \
-F "file=@predict.csv;type=text/csv"
```

- Additionally, you can access the Swagger UI for *kmeans* service in your Flask server from [here](#)

6.2.3.4 Notes

- Above services can easily be combined together in the backend to accept a model file, together with a prediction input
- File and to return the predicted output file (synchronous operation). But usually, we can expect AI jobs to be long running, hence the services would

need to be handled asynchronously.

- Additionally, once a model is fitted, users should be able to reuse the model for multiple predictions. Hence it is sensible to separate out model fitting and predictions into separate services.

6.3 FLASK RESTFUL SERVICES

Flask is a micro services framework allowing to write web services in python quickly. One of its extensions is Flask-RESTful. It adds for building REST APIs based on a class definition making it relatively simple. Through this interface we can then integrate with your existing Object Relational Models and libraries. As Flask-RESTful leverages the main features from Flask an extensive set of documentation is available allowing you to get started quickly and thoroughly. The Web page contains extensive documentation:

- <https://flask-restful.readthedocs.io/en/latest/>

We will provide a simple example that showcases some *hard coded* data to be served as a rest service. It will be easy to replace this for example with functions and methods that obtain such information dynamically from the operating system.

This example has not been tested. We like that the class defines a beautiful example to contribute to this section and explains what happens in this example.

```
from flask import Flask
from flask_restful import reqparse, abort
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)

COMPUTERS = {
    'computer1': {
        'processor': 'iCore7'
    },
    'computer2': {
        'processor': 'iCore5'
    },
    'computer3': {
        'processor': 'iCore3'
    },
}

def abort_if_cluster_doesnt_exist(computer_id):
    if computer_id not in COMPUTERS:
        abort(404, message="Computer {} does not exist".format(computer_id))

parser = reqparse.RequestParser()
parser.add_argument('processor')
```

```

class Computer(Resource):
    ''' shows a single computer item and lets you delete a computer
    item.'''

    def get(self, computer_id):
        abort_if_computer_doesnt_exist(computer_id)
        return COMPUTERS[computer_id]

    def delete(self, computer_id):
        abort_if_computer_doesnt_exist(computer_id)
        del COMPUTERS[computer_id]
        return '', 204

    def put(self, computer_id):
        args = parser.parse_args()
        processor = {'processor': args['processor']}
        COMPUTERS[computer_id] = processor
        return processor, 201

# ComputerList
class ComputerList(Resource):
    ''' shows a list of all computers, and lets you POST to add new computers'''

    def get(self):
        return COMPUTERS

    def post(self):
        args = parser.parse_args()
        computer_id = int(max(COMPUTERS.keys()).lstrip('computer')) + 1
        computer_id = 'computer%i' % computer_id
        COMPUTERS[computer_id] = {'processor': args['processor']}
        return COMPUTERS[computer_id], 201

##
## Setup the Api resource routing here
##
api.add_resource(ComputerList, '/computers')
api.add_resource(Computer, '/computers/<computer_id>')

if __name__ == '__main__':
    app.run(debug=True)

```

6.4 DJANGO REST FRAMEWORK

Django REST framework is a large toolkit to develop Web APIs. The developers of the framework provide the following reasons for using it according to the developers of that module:

1. *The Web browsable API improves usability.*
2. *Authentication policies including packages for OAuth1a and OAuth2.*
3. *Serialization that supports both ORM and non-ORM data sources.*
4. *Customizable all the way down - just use regular function-based views if you do not need the more powerful features.*
5. *Extensive documentation, and great community support.*
6. *Used and trusted by internationally recognised companies*

including Mozilla, Red Hat, Heroku, and Eventbrite."

- <https://www.django-rest-framework.org/>

An example is provided on their Web Page at

- <https://www.django-rest-framework.org/#example>

To document your django framework with Swagger you can look at this example:

- <https://www.django-rest-framework.org/topics/documenting-your-api/>

However, we believe that for our purposes the approach to use connexion from an OpenAPI is much more appealing, also using connexion and also flask for the REST service is easier to accomplish. Django is a large package that will take more time to getting used to.

6.5 GITHUB REST SERVICES

In this section we want to explore a more features of REST services and how to access them. Naturally many cloud services provide such REST sinterfaces. This is valid for IaaS, PaaS, and SaaS.

Instead of using a REST service for IaaS, let us here inspect a REST service for the Github.com platform.

Its interfaces are documented nicely at

- <https://developer.github.com/v3/>

We see that Github offers many resources that can be accessed by the users which includes

- Activities
- Checks
- Gists
- Git Data

- GitHub Apps
- Issues
- Migrations
- Miscellaneous
- Organizations
- Projects
- Pull Requests
- Reactions
- Repositories
- Searches
- Teams
- Users

Most likely we forgot the one or the other Resource that we can access via REST. It will be out of scope for us to explore all of these issues, so let us focus on how we for example access Github Issues. In fact we will use the script that we use to create issue tables for this book to showcase how easy the interaction is and to retrieve the information.

6.5.1 Issues

The REST service for issues is described in the following Web page as specification

- <https://developer.github.com/v3/issues/>

We see the following functionality:

- [List issues](#)
- [List issues for a repository](#)
- [Get a single issue](#)
- [Create an issue](#)
- [Edit an issue](#)
- [Lock an issue](#)
- [Unlock an issue](#)
- [Custom media types](#)

As we have learned in our REST section we need to issue GET requests to

obtain information about the issues. Such as

```
GET /issues
GET /user/issues
```

As response we obtain a json object with the information we need to further process it. Unfortunately, the free tier of github has limitations in regards to the frequency we can issue such requests to the service, as well as in the volume in regards to number of pages returned to us.

Let us now explore how to easily query some information. In our example we like to retrieve the list of issues for a repository as LaTeX table but also as markdown. This way we can conveniently integrate it in documents of either format. As LaTeX has a more sophisticated table management, let us first create a LaTeX table document and then use a program to convert LaTeX to markdown. For the later we can reuse a program called `pandoc` that can convert the table for LaTeX to markdown.

Let us assume we have a program called `issues.py` that prints the table in markdown format

```
$ python issues.py
```

An example for such a program is listed at.

- <https://github.com/cloudmesh-community/book/blob/master/bin/issues.py>

Although python provides the very nice module `requests` which we typically use for such issues. we have here just wrapped the commandline call to `curl` into a system command and redirect its output to a file. However, as we only get limited information back in pages, we need to continue such a request multiple times. To keep things simple we identified that for the project at this time not more than `n` pages need to be fetched, so we append the output from each page to the file.

Your task is it to improve this script and automatize this activity so that no maximum fetches have to be entered.

The reason why this program is so short is that we leverage the build in function for `json` data structure manipulation, here a read and a dump. When we look in the `issue.json` file that is created as intermediary file we see a list of items such as

```

[
  ...
  {
    "url": "https://api.github.com/repos/cloudmesh-community/book/issues/46",
    "repository_url": "https://api.github.com/repos/cloudmesh-community/book",
    "labels_url": "https://api.github.com/repos/cloudmesh-community/book/issues/46/labels{/name}",
    "comments_url": "https://api.github.com/repos/cloudmesh-community/book/issues/46/comments",
    "events_url": "https://api.github.com/repos/cloudmesh-community/book/issues/46/events",
    "html_url": "https://github.com/cloudmesh-community/book/issues/46",
    "id": 360613438,
    "node_id": "MDU6SXNzdWUzNjA2MTM0Mzg=",
    "number": 46,
    "title": "Taken: Virtualization",
    "user": {
      "login": "laszewsk",
      "id": 425045,
      "node_id": "MDQ6VXNlcjQyNTA0NQ==",
      "avatar_url": "https://avatars1.githubusercontent.com/u/425045?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/laszewsk",
      "html_url": "https://github.com/laszewsk",
      "followers_url": "https://api.github.com/users/laszewsk/followers",
      "following_url": "https://api.github.com/users/laszewsk/following{/other_user}",
      "gists_url": "https://api.github.com/users/laszewsk/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/laszewsk/starred{/owner}/{repo}",
      "subscriptions_url": "https://api.github.com/users/laszewsk/subscriptions",
      "organizations_url": "https://api.github.com/users/laszewsk/orgs",
      "repos_url": "https://api.github.com/users/laszewsk/repos",
      "events_url": "https://api.github.com/users/laszewsk/events{/privacy}",
      "received_events_url": "https://api.github.com/users/laszewsk/received_events",
      "type": "User",
      "site_admin": false
    },
    "labels": [],
    "state": "open",
    "locked": false,
    "assignee": {
      "login": "laszewsk",
      "id": 425045,
      "node_id": "MDQ6VXNlcjQyNTA0NQ==",
      "avatar_url": "https://avatars1.githubusercontent.com/u/425045?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/laszewsk",
      "html_url": "https://github.com/laszewsk",
      "followers_url": "https://api.github.com/users/laszewsk/followers",
      "following_url": "https://api.github.com/users/laszewsk/following{/other_user}",
      "gists_url": "https://api.github.com/users/laszewsk/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/laszewsk/starred{/owner}/{repo}",
      "subscriptions_url": "https://api.github.com/users/laszewsk/subscriptions",
      "organizations_url": "https://api.github.com/users/laszewsk/orgs",
      "repos_url": "https://api.github.com/users/laszewsk/repos",
      "events_url": "https://api.github.com/users/laszewsk/events{/privacy}",
      "received_events_url": "https://api.github.com/users/laszewsk/received_events",
      "type": "User",
      "site_admin": false
    },
    "assignees": [
      {
        "login": "laszewsk",
        "id": 425045,
        "node_id": "MDQ6VXNlcjQyNTA0NQ==",
        "avatar_url": "https://avatars1.githubusercontent.com/u/425045?v=4",
        "gravatar_id": "",
        "url": "https://api.github.com/users/laszewsk",
        "html_url": "https://github.com/laszewsk",
        "followers_url": "https://api.github.com/users/laszewsk/followers",
        "following_url": "https://api.github.com/users/laszewsk/following{/other_user}",
        "gists_url": "https://api.github.com/users/laszewsk/gists{/gist_id}",
        "starred_url": "https://api.github.com/users/laszewsk/starred{/owner}/{repo}",
        "subscriptions_url": "https://api.github.com/users/laszewsk/subscriptions",
        "organizations_url": "https://api.github.com/users/laszewsk/orgs",
        "repos_url": "https://api.github.com/users/laszewsk/repos",
        "events_url": "https://api.github.com/users/laszewsk/events{/privacy}",
        "received_events_url": "https://api.github.com/users/laszewsk/received_events",
        "type": "User",
        "site_admin": false
      }
    ],
    "milestone": null,
    "comments": 0,
    "created_at": "2018-09-16T07:35:35Z",
  }
]

```

```
"updated_at": "2018-09-16T07:35:35Z",
"closed_at": null,
"author_association": "CONTRIBUTOR",
"body": "Develop a section about Virtualization"
},
...
]
```

As we can see from this entry there is a lot of information associated that for our purposes we do not need, but certainly could be used to mine github in general.

We like to point out that github is actively mined for exploits where passwords are posted in clear text for AWS, Azure and other clouds. This is a common mistake as many sample programs ask the student to place the password directly into their programs instead of using a configuration file that is never part of the code repository.

6.5.2 Exercise

E.github.issues.1:

Develop a new code like the one in this section, but use python requests instead of the `os.system` call.

E.github.issues.2:

In the simple program we hardcoded the number of page requests. How can we find out exactly how many pages we need to retrieve? Implement your solution

E.github.issues.3:

Be inspired by the many REST interfaces. How can they be used to mine interesting things.

E.github.issues.4:

Can you create a project, author, or technology map based on information that is available in github. For example python projects may include a requirements file, or developers may work on some projects together, but others do other projects with others can you create a network?

E.github.issues.5:

Use github to develop some cool python programs that show some statistics about github. An example would be: Given a github repository, show the checkins by data and visualize them graphically for one committer and all committers. Use bokeh or matplotlib.

E.github.issues.6:

Develop a python program that retrieves a file. Develop a python program that uploads a file. Develop a class that does this and use it in your program. Use docopt to create a manual page. Please remember this prepares you for your project so this is very useful to do.

6.6 OPENAPI REST SERVICES WITH SWAGGER

Swagger <https://swagger.io/> is a tool for developing API specifications based on the OpenAPI Specification (OAS). It allows not only the specification, but the generation of code based on the specification in a variety of languages.

Swagger itself has a number of tools which together build a framework for developing REST services for a variety of languages.

6.6.1 Swagger Tools

The major Swagger tools of interest are:

Swagger Core

includes libraries for working with Swagger specifications
<https://github.com/swagger-api/swagger-core>.

Swagger Codegen

allows to generate code from the specifications to develop Client SDKs, servers, and documentation. <https://github.com/swagger-api/swagger-codegen>

Swagger UI

is an HTML5 based UI for exploring and interacting with the specified APIs <https://github.com/swagger-api/swagger-ui>

Swagger Editor

is a Web-browser based editor for composing specifications using YAML <https://github.com/swagger-api/swagger-editor>

Swagger Hub

is a Web service to collaboratively develop and host OpenAPI specifications <https://swagger.io/tools/swaggerhub/>

The developed APIs can be hosted and further developed on an online repository named SwaggerHub <https://app.swaggerhub.com/home> The convenient online editor is available which also can be installed locally on a variety of operating systems including macOS, Linux, and Windows.

6.6.2 Swagger Community Tools

notify us about other tools that you find and would like us to mention here.

6.6.2.1 Converting Json Examples to OpenAPI YAML Models

Swagger toolbox is a utility that can convert json to swagger compatible yaml models. It is hosted online at

- <https://swagger-toolbox.firebaseio.com/>

The source code to this tool is available on github at

- <https://github.com/essuraj/swagger-toolbox>

It is important to make sure that the json model is properly configured. As such each datatype must be wrapped in “quotes” and the last element must not have a , behind it.

In case you have large models, we recommend that you gradually add more and more features so that it is easier to debug in case of an error. This tool is not designed to provide back a full featured OpenAPI, but help you getting started deriving one.

Let us look at a small example. Let us assume we want to create a REST service to execute a command on the remote service. We know this may not be a good idea if it is not properly secured, so be extra careful. A good way to simulate this is to just use a return string instead of executing the command.

Let us assume the json schema looks like:

```
{
  "host": "string",
  "command": "string"
}
```

The output the swagger toolbox creates is

```
---
required:
- "host"
- "command"
properties:
  host:
    type: "string"
  command:
    type: "string"
```

As you can see it is far from complete, but it could be used to get you started.

Based on this tool develop a rest service to which you send a schema in JSON format from which you get back the YAML model.

6.7 REST WITH EVE

6.7.1 Rest Services with Eve

Next, we will focus on how to make a RESTful web service with Python Eve. Eve makes the creation of a REST implementation in python easy. More information about Eve can be found at:

- <http://python-eve.org/>

Although we do recommend Ubuntu 17.04, at this time there is a bug that forces

us to use 16.04. Furthermore, we require you to follow the instructions on how to install pyenv and use it to set up your python environment. We recommend that you use either python 2.7.14 or 3.6.4. We do not recommend you to use anaconda as it is not suited for cloud computing but targets desktop computing. If you use pyenv you also avoid the issue of interfering with your system wide python install. We do recommend pyenv regardless if you use a virtual machine or are working directly on your operating system. After you have set up a proper python environment, make sure you have the newest version of pip installed with

```
$ pip install pip -U
```

To install Eve, you can say

```
$ pip install eve
```

As Eve also needs a backend database, and as MongoDB is an obvious choice for this, we will have to first install MongoDB. MongoDB is a Non-SQL database which helps to store light weight data easily.

6.7.1.1 Ubuntu install of MongoDB

On Ubuntu you can install MongoDB as follows

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 \
--recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu \
xenia/mongodb-org/3.6 multiverse" | \
sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

6.7.1.2 macOS install of MongoDB

On macOS you can use the command

```
$ brew update
$ brew install mongodb
```

6.7.1.3 Windows 10 Installation of MongoDB

A student or student group of this class are invited to discuss on piazza on how to install mongoDB on Windows 10 and come up with an easy installation

solution. Naturally we have the same 2 different ways on how to run mongo. In user space or in the system. As we want to make sure your computer stays secure. the solution must have an easy way on how to shut down the Mongo services.

An enhancement of this task would be to integrate this function into cloudmesh cmd5 with a command *mongo* that allows for easily starting and stopping the service from *cms*.

6.7.1.4 Database Location

After downloading Mongo, create the *db* directory. This is where the Mongo data files will live. You can create the directory in the default location and assure it has the right permissions. Make sure that the */data/db* directory has the right permissions by running

6.7.1.5 Verification

In order to check the MongoDB installation, please run the following commands in one terminal:

```
$ mkdir -p ~/cloudmesh/data/db
$ mongod --dbpath ~/cloudmesh/data/db
```

In another terminal we try to connect to mongo and issue a mongo command to show the databases:

```
$ mongo --host 127.0.0.1:27017
$ show databases
```

If they execute without errors, you have successfully installed MongoDB. In order to stop the running database instance run the following command. simply CTRL-C the running mongod process

6.7.1.6 Building a simple REST Service

In this section we will focus on creating a simple rest service. To organize our work we will create the following directory:

```
$ mkdir -p ~/cloudmesh/eve
$ cd ~/cloudmesh/eve
```

As Eve needs a configuration and it is read in by default from the file `settings.py` we place the following content in the file `~/cloudmesh/eve/settings.py`:

```
MONGO_HOST = 'localhost'
MONGO_PORT = 27017
MONGO_DBNAME = 'student_db'
DOMAIN = {
    'student': {
        'schema': {
            'firstname': {
                'type': 'string'
            },
            'lastname': {
                'type': 'string'
            },
            'university': {
                'type': 'string'
            },
            'email': {
                'type': 'string',
                'unique': True
            }
        },
        'username': {
            'type': 'string',
            'unique': True
        }
    }
}
RESOURCE_METHODS = ['GET', 'POST']
```

The DOMAIN object specifies the format of a `student` object that we are using as part of our REST service. In addition we can specify `RESOURCE_METHODS` which methods are activated for the REST service. This way the developer can restrict the available methods for a REST service. To pass along the specification for mongoDB, we simply specify the hostname, the port, as well as the database name.

Now that we have defined the settings for our example service, we need to start it with a simple python program. We could name that program anything we like, but often it is called simply `run.py`. This file is placed in the same directory where you placed the **settings.py**. In our case it is in the file `~/cloudmesh/eve/run.py` and contains the following python program:

```
from eve import Eve
app = Eve()

if __name__ == '__main__':
    app.run()
```

This is the most minimal application for Eve, that uses the `settings.py` file for its configuration. Naturally, if we were to change the configuration file and for example change the DOMAIN and its schema, we would naturally have to remove the database previously created and start the service new. This is especially important as during the development phase we may frequently change

the schema and the database. Thus it is convenient to develop necessary cleaning actions as part of a Makefile which we leave as easy exercise for the students.

Next, we need to start the services which can easily be achieved in a terminal while running the commands:

Previously we started the mongoDB service as follows:

```
$ mongod --dbpath ~/cloudmesh/data/db/
```

This is done in its own terminal, so we can observe the log messages easily. Next we start in another window the Eve service with

```
$ cd ~/cloudmesh/eve
$ python run.py
```

You can find the codes and commands up to this point in the following document.

6.7.1.7 Interacting with the REST service

Yet in another window, we can now interact with the REST service. We can use the commandline to save the data in the database using the REST api. The data can be retrieved in XML or in json format. Json is often more convenient for debugging as it is easier to read than XML.

Naturally, we need first to put some data into the server. Let us assume we add the user Albert Zweistein.

```
$ curl -H "Content-Type: application/json" -X POST \
-d '{"firstname":"Albert","lastname":"Zweistein", \
  "school":"ISE","university":"Indiana University", \
  "email":"albert@iu.edu", "username": "albert"}' \
http://127.0.0.1:5000/student/
```

To achieve this, we need to specify the header using **H** tag saying we need the data to be saved using json format. And **X** tag says the HTTP protocol and here we use POST method. And the tag **d** specifies the data and make sure you use json format to enter the data. Finally, the REST api endpoint to which we must save data. This allows us to save the data in a table called **student** in MongoDB within a database called **eve**.

In order to check if the entry was accepted in mongo and included in the server

issue the following command sequence in another terminal:

```
$ mongo
```

Now you can query mongo directly with its shell interface

```
> show databases
> use student_db
> show tables # query the table names
> db.student.find().pretty() # pretty will show the json in a clear way
```

Naturally this is not really necessary for A REST service such as eve as we show you next how to gain access to the data via mongo while using REST calls. We can simply retrieve the information with the help of a simple URI:

```
$ curl http://127.0.0.1:5000/student?firstname=Albert
```

Naturally, you can formulate other URLs and query attributes that are passed along after the `?`.

This will now allow you to develop sophisticated REST services. We encourage you to inspect the documentation provided by Eve to showcase additional features that you could be using as part of your efforts.

Let us explore how to properly use additional REST API calls. We assume you have MongoDB up and running. To query the service itself we can use the URI on the Eve port

```
$ curl -i http://127.0.0.1:5000
```

Your payload should look like the one listed next, if your output is not formatted like this try adding `?pretty=1`

```
$ curl -i http://127.0.0.1:5000?pretty=1
```

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 150
Server: Eve/0.7.6 Werkzeug/0.11.15 Python/2.7.16
Date: Wed, 17 Jan 2018 18:34:07 GMT
```

```
{
  "_links": {
    "child": [
      {
        "href": "student",
        "title": "student"
      }
    ]
  }
}
```

Remember that the API entry points include additional information such as links

and a child, and href.

Set up a python environment that works for your platform. Provide explicit reasons why anaconda and other prepackaged python versions have issues for cloud related activities. When may you use anaconda and when should you not use anaconda. Why would you want to use pyenv?

What is the meaning and purpose of links, child, and href

In this case how many child resources are available through our API?

Develop a REST service with Eve and start and stop it

Define curl calls to store data into the service and retrieve it.

Write a Makefile and in it a target clean that cleans the data base. Develop additional targets such as start and stop, that start and stop the mongoDB but also the Eve REST service

Issue the command

```
$ curl -i http://127.0.0.1:5000/people
```

What does the `_links` section describe?

What does the `_items` section describe?

```
{
  "_items": [],
  "_links": {
    "self": {
      "href": "people",
      "title": "people"
    },
    "parent": {
      "href": "/",
      "title": "home"
    }
  },
  "_meta": {
    "max_results": 25,
    "total": 0,
    "page": 1
  }
}
```

6.7.1.8 Creating REST API Endpoints

Next we want to enhance our example a bit. First, let us get back to the eve

working directory with

```
$ cd ~/cloudmesh/eve
```

Add the following content to a file called **run2.py**

```
from eve import Eve
from flask import jsonify
import os
import getpass
app = Eve ()
@app.route('/student/albert')
def alberts_information():
    data = {
        'firstname': 'Albert',
        'lastname': 'Zweistsein',
        'university': 'Indiana University',
        'email': 'albert@example.com'
    }
    try:
        data['username'] = getpass.getuser()
    except:
        data['username'] = 'not-found'
    return jsonify(**data)

if __name__ == '__main__':
    app.run(debug=True, host="127.0.0.1")
```

After creating and saving the file. Run the following command to start the service

```
$ python run2.py
```

After running the command, you can interact with the service while entering the following url in the web browser:

```
http://127.0.0.1:5000/student/alberts
```

You can also open up a second terminal and type in it

```
$ curl http://127.0.0.1:5000/student/alberts
```

The following information will be returned:

```
{
  "firstname": "Albert",
  "lastname": "Zweistain",
  "university": "Indiana University",
  "email": "albert@example.com",
  "username": "albert"
}
```

This example illustrates how easy it is to create REST services in python while combining information from a dict with information retrieved from the system. The important part is to understand the decorator **app.route**. The parameter specifies the route of the API endpoint which will be the address appended to the

base path, `http://127.0.0.1:5000`. It is important that we return a jsonified object, which can easily be done with the `jsonify` function provided by flask. As you can see the name of the decorated function can be anything you like. The route specifies how we access it from the service.

6.7.1.9 REST API Output Formats and Request Processing

Another way of managing the data is to utilize class definitions and response types that we explicitly define.

If we want to create an object like Student, we can first define a python class. Create a file called **student.py**. Please, note the get method that returns simply the information in the dict for the class. It is not related to the REST get function.

```
class Student(object):
    def __init__(self, firstname, lastname, university, email):
        self.firstname = firstname
        self.lastname = lastname
        self.university = university
        self.email = email
        self.username = 'undefined'
    def get(self):
        return self.__dict__
    def setUsername(self, name):
        self.username = name
        return name
```

Next we define a REST service with Eve as shown in the following listing

```
from eve import Eve
from student import Student
import platform
import psutil
import json
from flask import Response
import getpass
app = Eve()
@app.route('/student/albert', methods=['GET'])
def processor():
    student = Student("Albert",
                     "Zweistein",
                     "Indiana University",
                     "albert@example.edu")

    response = Response()
    response.headers["Content-Type"] = "application/json; charset=utf-8"

    try:
        student.setUsername(getpass.getuser())
        response.headers["status"] = 200
    except:
        response.headers["status"] = 500

    response.data = json.dumps(student.get())
    return response

if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1')
```

In contrast to our earlier example, we are not using the `jsonify` object, but create explicitly a response that we return to the clients. The response includes a header that we return the information in json format, a status of 200, which means the object was returned successfully, and the actual data.

6.7.1.10 REST API Using a Client Application

 This example is not tested. Please provide feedback and improve.

In the Section [Rest Services with Eve](#) we created our own REST API application using Python Eve. Now once the service running, a we need to learn how to interact with it through clients.

First go back to the working folder:

```
$ cd ~/cloudmesh/eve
```

Here we create a new python file called **client.py**. The file include the following content.

```
import requests
import json

def get_all():
    response = requests.get("http://127.0.0.1:5000/student")
    print(json.dumps(response.json(), indent=4, sort_keys=True))

def save_record():
    headers = {
        'Content-Type': 'application/json'
    }

    data = '{"firstname":"Gregor",
           "lastname":"von Laszewski",
           "university": "Indiana University",
           "email":"jane@iu.edu",
           "username": "jane"}'

    response = requests.post('http://localhost:5000/student/',
                             headers=headers,
                             data=data)
    print(response.json())

if __name__ == '__main__':
    save_record()
    get_all()
```

Run the following command in a new terminal to execute the simple client by

```
$ python client.py
```

Here when you run this class for the first time, it will run successfully, but if you tried it for the second time, it will give you an error. Because we did set the email to be a unique field in the schema when we designed the settings.py file in the beginning. So if you want to save another record you must have entries with unique emails. In order to make this dynamic you can include a input reading by using the terminal to get the student data first and instead of the static data you can use the user input data from the terminal to get dynamic data. But for this exercise we do not expect that or any other form data functionality.

In order to get the saved data, you can comment the record saving function and uncomment the get all function. In python commenting is done by using #.

This client is using the **requests** python library to send GET, POST and other HTTP requests to the server so you can leverage build in methods to simplify your work.

The `get_all` function provides a way to get the output to the console with all the data in the student database. The `save_record` function provides a way to save data in the database. You can create dynamic functions in order to save dynamic data. However it may take some time for you to apply as exercise.

Write a RESTful service to determine a useful piece of information off of your computer i.e. disk space, memory, RAM, etc. In this exercise what you need to do is use a python library to extract data about computer information mentioned previously and send these information to the user once the user calls an API endpoint like `http://localhost:5000/performance/ram`, it must return the RAM value of the given machine. For each information like disk space, RAM, etc you can use an endpoint per each feature needed. As a tip for this exercise, use the `psutil` library in python to retrieve the data, and then get these information into an string then populate a class called `Computer` and try to save the object like wise.

6.7.1.11 Towards cmd5 extensions to manage eve and mongo



 *Part of this section related to management of the mongo db service is done by the cm4 command we will be developing as part of*

this class `cms mongo admin` that does all of the things explained next and more.

Naturally it is of advantage to have in cms administration commands to manage mongo and eve from cmd instead of targets in the Makefile. Hence, we **propose** that the class develops such an extension. We will create in the repository the extension called admin and hope that students through collaborative work and pull requests complete such an admin command.

The proposed command is located at:

- <https://github.com/cloudmesh/cloudmesh.rest/blob/master/cloudmesh/admin>

It will be up to the class to implement such a command. Please coordinate with each other.

The implementation based on what we provided in the Make file seems straight forward. A great extension is to load the objects definitions or eve e.g. settings.py not from the class, but from a place in .cloudmesh. I propose to place the file at:

```
~/cloudmesh/db/settings.py
```

the location of this file is used when the Service class is initialized with None. Prior to starting the service the file needs to be copied there. This could be achieved with a set command.

6.7.2 HATEOAS

In the previous section we discussed the basic concepts about RESTful web service. Next we introduce you to the concept of HATEOAS

HATEOAS stands for Hypermedia as the Engine of Application State and this is enabled by the default configuration within Eve. It is useful to review the terminology and attributes used as part of this configuration. HATEOS explains how REST API endpoints are defined and it provides a clear description on how the API can be consumed through these terms:

`_links`

Links describe the relation of current resource being accessed to the rest of the resources. It is like if we have a set of links to the set of objects or service endpoints that we are referring in the RESTful web service. Here an endpoint refers to a service call which is responsible for executing one of the CRUD operations on a particular object or set of objects. More on the links, the links object contains the list of serviceable API endpoints or list of services. When we are calling a GET request or any other request, we can use these service endpoints to execute different queries based on the user purpose. For instance, a service call can be used to insert data or retrieve data from a remote database using a REST API call. About databases we will discuss in detail in another chapter.

title

The title in the rest endpoint is the name or topic that we are trying to address. It describes the nature of the object by a single word. For instance student, bank-statement, salary,etc can be a title.

parent

The term parent refers to the very initial link or an API endpoint in a particular RESTful web service. Generally this is denoted with the primary address like <http://example.com/api/v1/>.

href

The term href refers to the url segment that we use to access the a particular REST API endpoint. For instance “student?page=1” will return the first page of student list by retrieving a particular number of items from a remote database or a remote data source. The full url will look like this, “<http://www.exampleapi.com/student?page=1>”.

In addition to these fields eve will automatically create the follwoing information when resources are created as showcased ot

- <http://python-eve.org/features.html>

Field	Description
-------	-------------

<code>_created</code>	item creation date.
<code>_updated</code>	item last updated on.
<code>_etag</code>	ETag, to be used for concurrency control and conditional requests.
<code>_id</code>	unique item key, also needed to access the individual item endpoint.

Pagination information can be included in the `_meta` field.

6.7.2.1 Filtering

Clients can submit query strings to the rest service to retrieve resources based on a filter. This also allows sorting of the results queried. One nice feature about using mongo as a backend database is that Eve not only allows python conditional expressions, but also mongo queries.

A number of examples to conduct such queries include:

```
$ curl -i -g http://eve-demo.herokuapp.com/people?where={%22lastname%22:%20%22Doe%22}
```

A python expression

```
$ curl -i http://eve-demo.herokuapp.com/people?where=lastname=="Doe"
```

6.7.2.2 Pretty Printing

Pretty printing is typically supported by adding the parameter `?pretty` OR `?pretty=1`

If this does not work you can always use python to beautify a json output with

```
$ curl -i http://localhost/people?pretty
```

or

```
$ curl -i http://localhost/people | python -m json.tool
```

6.7.2.3 XML

If for some reason you like to retrieve the information in XML you can specify this for example through curl with an Accept header

```
$ curl -H "Accept: application/xml" -i http://localhost
```

6.7.3 Extensions to Eve

A number of extensions have been developed by the community. This includes eve-swagger, eve-sqlalchemy, eve-elastic, eve-mongoengine, eve-neo4j, eve.net, eve-auth-jwt, and flask-sentinel.

Naturally there are many more.

Students have the opportunity to pick one of the community extensions and provide a section for the handbook.

Pick one of the extension, research it and provide a small section for the handbook so we add it.

6.7.3.1 Object Management with Eve and Evegenie

<http://python-eve.org/>

Eve makes the creation of a REST implementation in python easy. We will provide you with an implementation example that showcases that we can create REST services without writing a single line of code. The code for this is located at <https://github.com/cloudmesh/rest>

This code will have a master branch but will also have a dev branch in which we will add gradually more objects. Objects in the dev branch will include:

- virtual directories
- virtual clusters
- job sequences
- inventories

You may want to check our active development work in the dev branch. However for the purpose of this class the master branch will be sufficient.

6.7.3.1.1 Installation

First we have to install mongodb. The installation will depend on your operating system. For the use of the rest service it is not important to integrate mongodb

into the system upon reboot, which is focus of many online documents. However, for us it is better if we can start and stop the services explicitly for now.

On ubuntu, you need to do the following steps:

🕒 TODO: Section can be contributed by student.

On windows 10, you need to do the following steps:

🕒 TODO: Section can be contributed by student. If you elect Windows 10. You could be using the online documentation provided by starting it on Windows, or running it in a docker container.

On macOS you can use home-brew and install it with:

```
$ brew update
$ brew install mongodb
```

In future we may want to add ssl authentication in which case you may need to install it as follows:

```
$ brew install mongodb --with-openssl
```

6.7.3.1.2 Starting the service

We have provided a convenient Makefile that currently only works for macOS. It will be easy for you to adapt it to Linux. Certainly you can look at the targets in the makefile and replicate them one by one. Important targets are deploy and test.

When using the makefile you can start the services with:

```
$ make deploy
```

IT will start two terminals. IN one you will see the mongo service, in the other you will see the eve service. The eve service will take a file called sample.settings.py that is base on sample.json for the start of the eve service. The mongo service is configured in such a way that it only accepts incoming connections from the local host which will be sufficient for our case. The mongo data is written into the `$(USER)/.cloudmesh` directory, so make sure it exists.

To test the services you can say:

```
$ make test
```

You will see a number of json text been written to the screen.

6.7.3.1.3 Creating your own objects

The example demonstrated how easy it is to create a mongodb and an eve rest service. Now let us use this example to create your own. For this we have modified a tool called evegenie to install it onto your system.

The original documentation for evegenie is located at:

- <http://evegenie.readthedocs.io/en/latest/>

However, we have improved evegenie while providing a commandline tool based on it. The improved code is located at:

- <https://github.com/cloudmesh/evegenie>

You clone it and install on your system as follows:

```
$ cd ~/github
$ git clone https://github.com/cloudmesh/evegenie
$ cd evegenie
$ python setup.py install
$ pip install .
```

This should install in your system evegenie. YOU can verify this by typing:

```
$ which evegenie
```

If you see the path evegenie is installed. With evegenie installed its usage is simple:

```
$ evegenie

Usage:
  evegenie --help
  evegenie FILENAME
```

It takes a json file as input and writes out a settings file for the use in eve. Lets assume the file is called sample.json, than the settings file will be called sample.settings.py. Having the evegenie program will allow us to generate the settings files easily. You can include them into your project and leverage the

Makefile targets to start the services in your project. In case you generate new objects, make sure you rerun eve, kill all previous windows in which you run eve and mongo and restart. In case of changes to objects that you have designed and run previously, you need to also delete the mongod database.

6.8 OPENAPI 2.0

6.8.1 OpenAPI 2.0 Specification

Swagger provides through its specification the definition of REST services through a YAML or JSON document.

When following the API-specification-first approach to define and develop a RESTful service, the first and foremost step is to define the API conforming to the OpenAPI specification, and then using codegen tools to conveniently generate server side stub code, client code, documentations, in the language you desire. In Section [REST Service Generation with OpenAPI](#) we have introduced the codegen tool and how to use that to generate server side and client side code and documentation. In this Section [The Virtual Cluster example API Definition](#) we will use a slightly more complex example to show how to define an API following the OpenAPI 2.0 specification. The example is to retrieve virtual cluster (VC) object from the server.

The OpenAPI Specification is formerly known as Swagger RESTful API Documentation Specification. It defines a specification to describe and document a RESTful service API. It is also known under version 3.0 of swagger. However, as the tools for 3.0 are not yet completed, we will continue for now to use version swagger 2.0, till the transition has been completed. This is especially of importance, as we need to use the swagger codegen tool, which currently support only up to specification v2. Hence we are at this time using OpenAPI/Swagger v2.0 in our example. There are some structure and syntax changes in v3, while the essence is very similar. For more details of the changes between v3 and v2, please refer to A document published on the Web titled [Difference between OpenAPI 3.0 and Swagger 2.0](#).

You can write the API definition in json for yaml format. Let us discuss this format briefly and focus on yaml as it is easier to read and maintain.

On the root level of the yml document we see fields like *swagger*, *info*, and so on. Among these fields, *swagger*, *info*, and *path* are **required**. Their meaning is as follows:

swagger

specifies the version number. In our case a string value '2.0' is used as we are writing the definition conforming to the v2.0 specification.

info

defines metadata information related to the API. E.g., the API *version*, *title* and *description*, *termsOfService* if applicable, *contact* information and *license*, etc. Among these attributes, **version** and **title** are required while others are optional.

path

defines the actual endpoints of the exposed RESTful API service. Each endpoint has a *field pattern* as the key, and a *Path Item Object* as the value. In this example we have defined */vc* and */vc/{id}* as the two service endpoints. They will be part of the final service URL, appended after the service *host* and *basePath*, which will be explained later.

Let us focus on the *Path Item Object*. It contains one or more supported *operations* on the service endpoint. An *operation* is keyed by a valid HTTP operation verb, e.g., one of **get**, **put**, **post**, **delete**, or **patch**. It has a value of *Operation Object* that describes the operations in more detail.

The *Operation Object* will always **require** a *Response Object*. A *Response Object* has a *HTTP status code* as the key, e.g., **200** as successful return; **40X** as authentication and authorization related errors; and **50x** as other server side servers. It can also has a default response keyed by **default** for undeclared http status return code. The *Response Object* value has a **required** *description* field, and if anything is returned, a *schema* indicating the object type to be returned, which could be a primitive type, e.g., *string*, or an *array* or customized *object*. In case of *object* or an *array* of *object*, use *\$ref* to point to the definition of the object. In this example, we have

\$ref: “#/definitions/VC”

to point to the *VC* definition in the *definitions* section in the same specification file, which will be explained later.

Besides the required field, the *Operation Object* **can** have *summary* and *description* to indicate what the operation is about; and *operationId* to uniquely identify the operation; and *consumes* and *produces* to indicate what MIME types it expects as input and for returns, e.g., *application/json* in most modern RESTful APIs. It can further specify what input parameter is expected using *parameters*, which requires a *name* and *in* fields. *name* specifies the name of the parameter, and *in* specifies from where to get the parameter, and its possible values are *query*, *header*, *path*, *formData* or *body*. In this example in the */vc/{id}* path we obtain the *id* parameter from the URL path so it has the *path* value. When the *in* has *path* as its value, the *required* field is required and has to be set as *true*; when the *in* has value other than *body*, a *type* field is required to specify the type of the parameter.

While the three root level fields mentioned previously are required, in most cases we will also use other optional fields.

host

to indicate where the service is to be deployed, which could be *localhost* or a valid IP address or a DNS name of the host where the service is to be deployed. If other port number other than *80* is to be used, write the port number as well, e.g., *localhost:8080*.

schemas

to specify the transfer protocol, e.g., *http* or *https*.

basePath

to specify the common base URL to be append after the *host* to form the base path for all the endpoints, e.g., */api* or */api/1.0/*. In this example with the values specified we would have the final service endpoints *http://localhost:8080/api/vcs* and *http://localhost:8080/api/vc/{id}* by combining the *schemas*, *host*, *basePath* and *paths* values.

consumes and produces

can also be specified on the top level to specify the default MIME types of the input and return if most *paths* and the defined operations have the same.

definitions

as used in in the *paths* field, in order to point to a customized object definition with a *\$ref* keyword.

The *definitions* field really contains the object definition of the customized objects involved in the API, similar to a class definition in any Object Oriented programming language. In this example, we defined a *VC* object, and hierarchically a *Node* object. Each object defined is a type of *Schema Object* in which many field could be used to specify the object (see details in the REF link at top of the document), but the most frequently used ones are:

type

to specify the type, and in the customized definition case the value is mostly *object*.

required

field to list the names of the required attributes of the object.

properties

has the detailed information of each attribute/property of the object, e.g, *type*, *format*. It also supports hierarchical object definition so a property of one object could be another customized object defined elsewhere while using *schema* and *\$ref* keyword to point to the definition. In this example we have defined a *VC*, or virtual cluster, object, while it contains another object definition of

Node

as part of a cluster.

6.8.1.1 The Virtual Cluster example API Definition

6.8.1.1.1 Terminology

VC

A virtual cluster, which has one Front-End (FE) management node and multiple compute nodes. A VC object also has *id* and *name* to identify the VC, and *nnodes* to indicate how many compute nodes it has.

FE

A management node from which to access the compute nodes. The FE node usually connects to all the compute nodes via private network.

Node

A computer node object that the info *ncores* to indicate number of cores it has, and *ram* and *localdisk* to show the size of RAM and local disk storage.

6.8.1.1.2 Specification

```
swagger: "2.0"
info:
  version: "1.0.0"
  title: "A Virtual Cluster"
  description: "Virtual Cluster as a test of using swagger-2.0 specification and codegen"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "IU ISE software and system team"
  license:
    name: "Apache"
host: "localhost:8080"
basePath: "/api"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /vcs:
    get:
      description: "Returns all VCs from the system that the user has access to"
      produces:
        - "application/json"
      responses:
        "200":
          description: "A list of VCs."
          schema:
            type: "array"
            items:
              $ref: "#/definitions/VC"
```

```

/vcs/{id}:
  get:
    description: "Returns all VCs from the system that the user has access to"
    operationId: getVCById
    parameters:
      - name: id
        in: path
        description: ID of VC to fetch
        required: true
        type: string
    produces:
      - "application/json"
    responses:
      "200":
        description: "The vc with the given id."
        schema:
          $ref: "#/definitions/VC"
      default:
        description: unexpected error
        schema:
          $ref: '#/definitions/Error'
definitions:
  VC:
    type: "object"
    required:
      - "id"
      - "name"
      - "nnodes"
      - "FE"
      - "computes"
    properties:
      id:
        type: "string"
      name:
        type: "string"
      nnodes:
        type: "integer"
        format: "int64"
      FE:
        type: "object"
        schema:
          $ref: "#/definitions/Node"
      computes:
        type: "array"
        items:
          $ref: "#/definitions/Node"
      tag:
        type: "string"
  Node:
    type: "object"
    required:
      - "ncores"
      - "ram"
      - "localdisk"
    properties:
      ncores:
        type: "integer"
        format: "int64"
      ram:
        type: "integer"
        format: "int64"
      localdisk:
        type: "integer"
        format: "int64"
  Error:
    required:
      - code
      - message
    properties:
      code:
        type: integer
        format: int32
      message:
        type: string

```

6.8.1.2 References

[The official OpenAPI 2.0 Documentation](#)

6.8.2 OpenAPI REST Service via Introspection

The simplest way to create an OpenAPI service is to use the connexion service and read in the specification from its yaml file. It will then be introspected and dynamically methods are created that are used for the implementation of the server.

The full example for this is available in

- <https://github.com/cloudmesh-community/nist/tree/master/examples/flask-connexion-swagger>

An extensive documentation is available at

- <https://media.readthedocs.org/pdf/connexion/latest/connexion.pdf>

This example will return dynamically the cpu information of a computer to demonstrate how simple it is to generate in python a REST service from an OpenAPI specification.

Our requirements.txt file includes

```
flask
connexion[swagger-ui]
```

as dependencies. The `server.py` file simply contains the following code:

```
from flask import jsonify
import connexion

# Create the application instance
app = connexion.App(__name__, specification_dir="./")

# Read the yaml file to configure the endpoints
app.add_api("cpu.yaml")

# create a URL route in our application for "/"
@app.route("/")
def home():
    msg = {"msg": "It's working!"}
    return jsonify(msg)

if __name__ == "__main__":
    app.run(port=8080, debug=True)
```

This will run our REST service under the assumption we have a `cpu.yaml` and a `cpu.py`

files as our yaml file calls out methods from `cpu.py`

The yaml file looks as follows

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "cpuinfo"
  description: "A simple service to get cpuinfo as an example of using swagger-2.0 specification and codegen"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "Cloudmesh REST Service Example"
  license:
    name: "Apache"
host: "localhost:8080"
basePath: "/cloudmesh"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /cpu:
    get:
      tags:
        - CPU
      operationId: cpu.get_processor_name
      description: "Returns cpu information of the hosting server"
      produces:
        - "application/json"
      responses:
        "200":
          description: "CPU info"
          schema:
            $ref: "#/definitions/CPU"
definitions:
  CPU:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"
```

Here we simply implement a get method and associate it with the URL `/cpu`. The operationid, defines the method that we call which as we used the local directory is included in the file `cpu.py`. This is controlled by the prefix in the operation id.

A very simple function to return the cpu information is defined in `cpu.py` which we list next

```
import os, platform, subprocess, re
from flask import jsonify

def get_processor_name():
    if platform.system() == "Windows":
        p = platform.processor()
    elif platform.system() == "Darwin":
        command = "/usr/sbin/sysctl -n machdep.cpu.brand_string"
        p = subprocess.check_output(command, shell=True).strip().decode()
    elif platform.system() == "Linux":
        command = "cat /proc/cpuinfo"
        all_info = subprocess.check_output(command, shell=True).strip().decode()
        for line in all_info.split("\n"):
            if "model name" in line:
                p = re.sub(".*model name.*:", "", line, 1)
```

```
else:
    p = "cannot find cpuinfo"
    pinfo = {"model": p}
    return jsonify(pinfo)
```

We have implemented this function to return a jsonified information from the dict pinfo.

To simplify working with this example, we also provide a makefile for OSX that allows us to call the server and the call to the server in two different terminals

```
define terminal
    osascript -e 'tell application "Terminal" to do script "cd $(PWD); $1"'
endef

install:
    pip install -r requirements.txt

demo:
    $(call terminal, python server.py)
    sleep 3
    @echo "======"
    @echo "Get the info"
    @echo "======"
    curl http://localhost:8080/cloudmesh/cpu
    @echo
    @echo "======"
```

When we call

```
make demo
```

our demo is run.

6.8.2.1 Verification

It is important to be able to verify if a yaml file is correct. To identify this, the easiest method is to use the swagger editor. There is an online version available at:

- <https://editor.swagger.io/>

Go to the Web site, remove the current petstore example and simply paste your yaml file in it. Debug messages will be helping you to correct things.

A terminal based command may also be helpful, but is a bit difficult to read.

```
$ connexion run cpu.yaml --stub --debug
```

6.8.2.2 Mock service

In some cases it may be useful to develop the API without having yet developed methods that you call with the OperationI. In this case it is useful to run a mock service. YOU can invoce such a service with

```
$ connexion run cpu.yaml --mock=all -v
```

6.8.2.3 Exercise

OpenAPI.Conexion.1:

Modify the makefile so it works also on ubuntu, but do not disable the ability to run it correctly on OSX. Tip use if's in makefiles base on the OS. You can look at the makefiles that create this book as example. find alternatives to sarting a terminal in Linux.

OpenAPI.Conexion.2:

Modify the makefile so it works also on Windows 10, but do not disable the ability to run it correctly on OSX. Tip use ifs in makefiles. You can look at the makefiles that create this book as example. Find alternatives to start a powershell or cmd.exe in windows. Maybe you need to use gitbash.

OpenAPI.Conexion.3:

Implement a swagger specification of an issue related to the NIST BDRA. Implement it. Please remember this could prepare you for a project good topics include:

- *virtual compute service interfacing with aws, azure, google or openstack*
- *virtual directory service interfacing with google drive, box, github, iCloud, ftp, scp, and others*

As there are so many possibilities to contribute, come up in class with one specification and than implement it for different providers. The difficulty here is that it is not done for one IaaS, but for all of them and all can be integrated.

This exercise is typically growing to be part of your class project.

OpenAPI.Conexion.4:

Develop instructions on how to integrate the OpenAPI service framework in a WSGI based Web service. Chose a service you like so that the service could run in production.

OpenAPI.Conexion.5:

Develop instructions on how to integrate the OpenAPI service framework in Tornado so the service could run in production.

6.8.3 OpenAPI REST Service via Codegen



[REST 36:02 Swagger](#)

In this subsection we are discussing how to use OpenAPI 2.0 and Swagger Codegen to define and develop a REST Service.

We assume you have been familiar with the concept of REST service, OpenAPI as discussed in section [Overview of Rest](#).

In next section we will further look into the Swagger/OpenAPI 2.0 specification [Swagger Specification](#) and use a slight more complex example to walk you through the design of a RESTful service following the OpenAPI 2.0 specifications.

We will use a simple example to demonstrate the process of developing a REST service with Swagger/OpenAPI 2.0 specification and the tools related to is. The general steps are:

- Step 1 (Section [Step 1: Define Your REST Service](#). Define the REST service conforming to Swagger/OpenAPI 2.0 specification. It is a YAML document file with the basics of the REST service defined, e.g., what resources it has and what actions are supported.
- Step 2 (Section [Step 2: Server Side Stub Code Generation and](#)

[Implementation](#). Use Swagger Codegen to generate the server side stub code. Fill in the actual implementation of the business logic portion in the code.

- Step 3 (Section [Step 3: Install and Run the REST Service](#). Install the server side code and run it. The service will then be available.
- Step 4 (Section [Step 4: Generate Client Side Code and Verify](#). Generate client side code. Develop code to call the REST service. Install and run to verify.

6.8.3.1 Step 1: Define Your REST Service

In this example we define a simple REST service that returns the hosting server's basic CPU info. The example specification in yaml is as follows:

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "cpuinfo"
  description: "A simple service to get cpuinfo as an example of using swagger-2.0 specification and codegen"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "Cloudmesh REST Service Example"
  license:
    name: "Apache"
host: "localhost:8080"
basePath: "/api"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /cpu:
    get:
      description: "Returns cpu information of the hosting server"
      produces:
        - "application/json"
      responses:
        "200":
          description: "CPU info"
          schema:
            $ref: "#/definitions/CPU"
definitions:
  CPU:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"
```

6.8.3.2 Step 2: Server Side Stub Code Generation and Implementation

With the REST service having been defined, we can now generate the server

side stub code easily.

6.8.3.2.1 Setup the Codegen Environment

You will need to [install the Swagger Codegen tool](#) if not yet done so. For macOS we recommend that you use the homebrew install via

```
$ brew install swagger-codegen
```

On Ubuntu you can install swagger as follows (update the version as needed):

```
$ mkdir ~/swagger
$ cd ~/swagger
$ wget https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-codegen-cli/2.3.1/swagger-codegen-cli-2.
$ alias swagger-codegen="java -jar ~/swagger/swagger-codegen-cli-2.3.1.jar"
```

Add the alias to your `.bashrc` or `.bash_profile` file. After you start a new terminal you can use in that terminal now the command

```
swagger-codegen
```

For other platforms you can just use the `.jar` file, which can be downloaded from [this link](#).

Also make sure Java 7 or 8 is installed in your system. To have a well defined location we recommend that you place the code in the directory `~/cloudmesh`. In this directory you will also place the `cpu.yaml` file.

6.8.3.2.2 Generate Server Stub Code

After you have the codegen tool ready, and with Java 7 or 8 installed in your system, you can run the following to generate the server side stub code:

```
$ swagger-codegen generate \
  -i ~/cloudmesh/cpu.yaml \
  -l python-flask \
  -o ~/cloudmesh/swagger_example/server/cpu/flaskConnexion \
  -D supportPython2=true
```

or if you have not created an alias

```
$ java -jar swagger-codegen-cli.jar generate \
  -i ~/cloudmesh/cpu.yaml \
  -l python-flask \
  -o ~/cloudmesh/swagger_example/server/cpu/flaskConnexion \
  -D supportPython2=true
```

In the specified directory under *flaskConnexion* you will find the generated python flask code, with python 2 compatibility. It is best to place the swagger code under the directory `~/cloudmesh` to have a location where you can easily find it. If you want to use python 3 make sure to chose the appropriate option. To switch between python 2 and python 3 we recommend that you use pyenv as discussed in our python section.

6.8.3.2.3 Fill in the actual implementation

Under the *flaskConnexion* directory, you will find a *swagger_server* directory, under which you will find directories with *models* defined and *controllers* code stub resides. The models code are generated from the definition in Step 1. On the controller code though, we will need to fill in the actual implementation. You may see a `default_controller.py` file under the *controllers* directory in which the resource and action is defined but yet to be implemented. In our example, you will find such a function definition which we list next:

```
def cpu_get(): # noqa: E501
    """cpu_get

    Returns cpu info of the hosting server # noqa: E501

    :rtype: CPU
    """
    return 'do some magic!'
```

Please note the `do some magic!` string at the return of the function. This ought to be replaced with actual implementation what you would like your REST call to be really doing. In reality this could be some call to a backend database or datastore; a call to another API; or even calling another REST service from another location. In this example we simply retrieve the cpu model information from the hosting server through a simple python call to illustrate this principle. Thus you can define the following code:

```
import os, platform, subprocess, re

def get_processor_name():
    if platform.system() == "Windows":
        return platform.processor()
    elif platform.system() == "Darwin":
        command = "/usr/sbin/sysctl -n machdep.cpu.brand_string"
        return subprocess.check_output(command, shell=True).strip()
    elif platform.system() == "Linux":
        command = "cat /proc/cpuinfo"
        all_info = subprocess.check_output(command, shell=True).strip()
        for line in all_info.split("\n"):
            if "model name" in line:
                return re.sub(".*model name.*:", "", line, 1)
    return "cannot find cpuinfo"
```

And then change the `cpu_get()` function to the following:

```
def cpu_get(): # noqa: E501
    """cpu_get

    Returns cpu info of the hosting server # noqa: E501

    :rtype: CPU
    """
    return CPU(get_processor_name())
```

Please note we are returning a CPU object as defined in the API and later generated by the codegen tool in the `models` directory.

It is best *not* to include the definition of `get_processor_name()` in the same file as you see the definition of `cpu_get()`. The reason for this is that in case you need to regenerate the code, your modified code will naturally be overwritten. Thus, to minimize the changes, we do recommend to maintain that portion in a different filename and import the function as to keep the modifications small.

At this step we have completed the server side code development.

6.8.3.3 Step 3: Install and Run the REST Service:

Now we can install and run the REST service. It is strongly recommended that you run this in a pyenv or a virtualenv environment.

6.8.3.3.1 Start a virtualenv:

In case you are not using pyenv, please use virtual env as follows:

```
$ virtualenv RESTServer
$ source RESTServer/bin/activate
```

6.8.3.3.2 Make sure you have the latest pip:

```
$ pip install -U pip
```

6.8.3.3.3 Install the requirements of the server side code:

```
$ cd ~/cloudmesh/swagger_example/server/cpu/flaskConnexion
$ pip install -r requirements.txt
```

6.8.3.3.4 Install the server side code package:

Under the same directory, run:

```
$ python setup.py install
```

6.8.3.3.5 Run the service

Under the same directory:

```
$ python -m swagger_server
```

You should see a message like this:

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

6.8.3.3.6 Verify the service using a web browser:

Open a web browser and visit:

- <http://localhost:8080/api/cpu>

to see if it returns a json object with cpu model info in it.

Assignment: How would you verify that your service works with a `curl` call?

6.8.3.4 Step 4: Generate Client Side Code and Verify

In addition to the server side code swagger can also create a client side code.

6.8.3.4.1 Client side code generation:

Generate the client side code in a similar fashion as we did for the server side code:

```
$ java -jar swagger-codegen-cli.jar generate \
  -i ~/cloudmesh/cpu.yaml \
  -l python \
  -o ~/cloudmesh/swagger_example/client/cpu \
  -D supportPython2=true
```

6.8.3.4.2 Install the client side code package:

Although we could have installed the client in the same python pyenv or virtualenv, we showcase here that it can be installed in a completely different

environment. That would make it even possible to use a python 3 based client and a python 2 based server showcasing interoperability between python versions (although we just use python 2 here). Thus we create a new python virtual environment and conduct our install.

```
$ virtualenv RESTClient
$ source RESTClient/bin/activate
$ pip install -U pip
$ cd swagger_example/client/cpu
$ pip install -r requirements.txt
$ python setup.py install
```

6.8.3.4.3 Using the client API to interact with the REST service

Under the directory *swagger_example/client/cpu* you will find a README.md file which serves as an API documentation with example client code in it. E.g., if we save the following code into a `.py` file:

```
from __future__ import print_function
import time
import swagger_client
from swagger_client.rest import ApiException
from pprint import pprint
# create an instance of the API class
api_instance = swagger_client.DefaultApi()

try:
    api_response = api_instance.cpu_get()
    pprint(api_response)
except ApiException as e:
    print("Exception when calling DefaultApi->cpu_get: %s\n" % e)
```

We can then run this code to verify the calling to the REST service actually works. We are expecting to see a return similar to this:

```
{'model': 'Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz'}
```

Obviously, we could have applied additional cleanup of the information returned by the python code, such as removing duplicated spaces.

6.8.3.5 Towards a Distributed Client Server

Although we develop and run the example on one localhost machine, you can separate the process into two separate machines. E.g., on a server with external IP or even DNS name to deploy the server side code, and on a local laptop or workstation to deploy the client side code. In this case please make changes on the API definition accordingly, e.g., the **host** value.

6.9 EXERCISES

E.OpenAPI.1:

In Section [OpenAPI 3.0 REST Service via Introspection](#), we introduced a schema. The question relates to termsOfService: Investigate what the termOfService attribute is and suggest a better value. Discuss on piazza.

E.OpenAPI.2:

In Section [OpenAPI 3.0 REST Service via Introspection](#), we introduced a schema. The question relates to model: What is the meaning of model under the definitions?

E.OpenAPI.3:

In Section [OpenAPI 3.0 REST Service via Introspection](#), we introduced a schema. The question relates to \$ref: what is the meaning of the \$ref. Discuss on piazza, come up with a student answer in class.

E.OpenAPI.4:

In Section [OpenAPI 3.0 REST Service via Introspection](#), we introduced a schema. What does the response 200 mean. Do you need other responses?

E.OpenAPI.5:

After you have gone through the entire section and verified it works for you add create a more sophisticated schema and add more attributes exposing more information from your system.

How can you for example develop a rest service that exposes portions of your file system serving large files, e.g. their filenames and their size? How would you download these files? Would you use a rest service, or would you register an alternative service such as ftp,

DAV, or others? Please discuss in piazza. Note this will be a helping you to prepare a larger assignment. Think about this first before you implement.

You can try expand the API definition with more resources and actions included. E.g., to include more detailed attributes in the CPU object and to have those information provided in the actual implementation as well. Or you could try defining totally different resources.

The codegen tool provides a convenient way to have the code stubs ready, which frees the developers to focus more on the API definition and the real implementation of the business logic. Try with complex implementation on the back end server side code to interact with a database/datastore or a 3rd party REST service.

For advanced python users, you can naturally use function assignments to replace the `cpu_get()` entirely even after loading the instantiation of the server. However, this is not needed. If you are an advanced python developer, please feel free to experiment and let us know how you suggest to integrate things easily.

7 GRAPHQL



Learning Objectives

- Learn about GraphQL
 - Develop a GraphQL Server in Python
-

GraphQL is a data query language developed by Facebook.

GraphQL allows clients to request they need while specifying attributes in the query without thinking much about the API implementation. It simplifies access and reduces traffic as the application has control over the data it needs and its format. Hence GraphQL reduces the network traffic as only the necessary data is transferred from server to client.

Unlike REST APIs, which require often loading data via multiple queries, GraphQL can get typically all the data in a single request. GraphQL APIs are defined in terms of types and fields. Types help GraphQL to ensure that client only asks for what is possible and in case of faults, provides clear and helpful errors.

Initially GraphQL was implemented in JavaScript. Today there are several other implementations in different languages available. To show case how to use GraphQL, we will explore the *graphql-python* implementation in this chapter. The official documentation of GraphQL is available at [\[54\]](#)

7.1 PREREQUISITES

Before we start we need to install a number of tools that we use throughout the chapter

7.1.1 Install Graphene

In this chapter, we will use [Graphene](#) which is a library for implementing

GraphQL APIs in Python. Use `pip` to install Graphene

```
$ pip install graphene==2.0.1 graphene-django==2.0.0
```

7.1.2 Install Django

For the purpose of demonstrating in this chapter, we will use Django as Python web framework. Django is a popular Python web framework which already comes with a lot of boilerplate code. It is mature and has a very large community. It has inbuilt support for Object Relational Mapping which is based on Database Code-First approach. Please refer [55] for more Django information. Use `pip` to install Django

```
$ pip install django==2.0.2 django-filter==1.1.0
```

7.1.3 Install GraphiQL

In case you prefer to use a browser interface which could be useful for debugging purposes a number of GraphQL browsers are available. A free version is GraphiQL. It is an IDE(Interactive Development Environment) for GraphQL where you can run a *GraphQL Query*. There are many implementations of *GraphiQL* available. For this chapter we will use [GraphiQL](#). You can download the *GraphiQL* installation files specific to your OS from [GraphiQL Releases](#).

For MacOS, you can even use `homebrew` to install it

```
brew cask install graphiql
```

Commercial GraphQL browsers are available from

- [Insomnia](#)
- [Altair](#)

7.2 GRAPHQL TYPE SYSTEM AND SCHEMA

To get started with GraphQL we will first explore the GraphQL type system and schema creation.

7.2.1 Type System

In GraphQL a query is what a client requests from the GraphQL server. The result will be obtained in a *structure* defined by *type* and *schema*. Thus, the client will know ahead of time what it is going to get as result as part of a well formed response. For this to work, the data is often assumed to be structured data.

To demonstrate the type system we will use a simple example while looking at authors and co-authors of papers. We represent in this example a database that contains a number of authors. Each author has a publication count and a number of coauthors that are identified by name. We assume for this simple example that all author names are unique.

Here is how a simple GraphQL query would look like

```
{
  author {
    name
    publication_count
    coauthors {
      name
    }
  }
}
```

The response is

```
{
  "author": {
    "name": "John Doe",
    "publication_count": 25,
    "coauthors": [
      {
        "name": "Mary Jane"
      },
      {
        "name": "David Miller"
      }
    ]
  }
}
```

For this to work, we need to define the types that are going to be honored by the GraphQL service so that when a query is received by the server, it is first validated against a schema that defines the types contained within the GraphQL service.

Hence, types must be defined as part of each GraphQL service. They are defined with the GraphQL schema language which is programming language agnostic. An example of a GraphQL type is:

```
type Author {
  name: String!
  publication_count: Int
  coauthors: [Author!]!
}
```

Here we define the type `author` with three fields `name`, `publication`, and `coauthors`. Note that the `!` indicates a field value, that cannot be null and must have a defined value. `[Author!]!` means that an array is returned, but that array cannot be null and also none of the items in the array can be null.

7.2.2 Scalar Types

GraphQL supports the following scalar types:

- `String`: UTF8 characters
- `Int`: 32 bit signed integer
- `Float`: Double precision floating point value
- `Boolean`: true or false
- `ID`: Represents a unique identifier which can be used as a key to fetch the object

7.2.3 Enumeration Types

`enum` is a scalar type which defines restricted set of values. When a GraphQL schema defines a field of `enum` type, we expect that the field's value be of the type `enum` including only the values that are included in that enumeration. An example of an `enum` type is

```
enum ContainerType {
  Docker
  Kubernetes
  DockerSwarm
}
```

7.2.4 Interfaces

Similar to common programming languages, the GraphQL type system also supports an `interface`. Interfaces allow us to assure that certain fields are part of the definition of a type. When a type implements an `interface`, it needs to specify all the fields that are defined through the `interface`.

We will illustrate this in the following example, where we define simple `ComputeService` interface type. This `interface` declares `id`, `name` and `memory` fields. This means

that a `Container` and a `VirtualMachine` both of which implement `ComputeService`, must have the fields defined in the `interface`. They may or may not have additional fields like we demonstrate in our example with the field `systemType` of type `ContainerType` in case of `Container` and field `systemType` of type `VMBackend` in case of the `VirtualMachine`.

```
interface ComputeService {
  id: ID!
  name: String!
  memory: Int!
}

type Container implements ComputeService {
  id: ID!
  name: String!
  memory: Int!
  systemType: ContainerType!
}

type VirtualMachine implements ComputeService {
  id: ID!
  name: String!
  memory: Int!
  systemType: VMBackend!
}
```

7.2.5 Union Types

As the name suggests a `union` type represents the union of two or more types. The following example shows how we can define a `union` type. As you can see we use the `|` character to indicate the union operator.

```
union ComputeType = Container | VirtualMachine
```

Now when we write a GraphQL query to get the `ComputeType` information, we can ask some of the common fields and some of the specific fields conditionally. In the next example we request `AllComputeTypes` with common fields like `id`, `name` and fields specific to either `VirtualMachine` Or `Container`.

```
{
  AllComputeTypes {
    id
    name
    ... on VirtualMachine {
      user
    }
    ... on Container {
      type
    }
  }
}
```

7.3 GRAPHQL QUERY

An application asks for data from server in form of a GraphQL *query*. A GraphQL query can have different fields and arguments and in this section we describe how to use them.

7.3.1 Fields

A very simple definition of a query is to ask for specific fields that belong to an object stored in GraphQL.

In the next examples we will use data related to repositories in github.

When asking the query

```
{
  repository {
    name
  }
}
```

we obtain the following response

```
{
  "data": {
    "repository": {
      "name": "cm"
    }
  }
}
```

As we can see the response data format looks exactly like the query. This way a client knows exactly what data it has to consume. In the previous example, the `name` field returns the data of type `string`. Clients can also ask for an object representing any match within the GraphQL database.

For example following query

```
{
  community {
    name
    repositories {
      name
    }
  }
}
```

returns the response

```
{
  "data": {
    "community": {
      "name": "cloudmesh-community",
      "repositories": [{
        "name": "S.T.A.R boat"
      }, {
        "name": "book"
      }]
    }
  }
}
```

7.3.2 Arguments

As you may already know in REST services you can pass parameters as part of a request via query parameters through *GET* or a request body through *POST*. However in GraphQL, for every field, you provide an argument restricting the data returned to only the information that you need. This reduces the traffic for returning the information that is needed without doing the postprocessing on the client. These restricting arguments can be of scalar type, enumeration type and others.

Let us look at an example of a query where we only ask for first 3 repositories in cloudmesh community

```
{
  repositories(first: 3) {
    name
    url
  }
}
```

The response will be similar to

```
{
  "data": {
    "repositories": [{
      "name": "boat",
      "url": "https://github.com/cloudmesh-community/boat"
    }, {
      "name": "book",
      "url": "https://github.com/cloudmesh-community/book"
    }, {
      "name": "case",
      "url": "https://github.com/cloudmesh-community/case"
    }
  ]
}
```

7.3.3 Fragments

Fragments allow us to reuse portions of a query. Let us look at the following complex query, which includes repetitive fields:

```
{
  boatRepositoryExample: repository(name: boat) {
    name
    full_name
    url
    description
  }
  cloudRepositoryExample: repository(name: cm) {
    name
    full_name
    url
    description
  }
}
```

As the query gets bigger and more complex, we can use a `fragment` to split it into smaller chunks. This `fragment` can then be re-used, which can significantly reduce the query size and also make it more readable.

A `fragment` can be defined as

```
fragment repositoryInfo on Repository {
  name
  full_name
  url
  description
}
```

and can be used in a query like this

```
{
  boatRepositoryExample: repository(name: boat) {
    ...repositoryInfo
  }
  cloudRepositoryExample: repository(name: cm) {
    ...repositoryInfo
  }
}
```

The response for this query will look like

```
{
  "data": {
    "boatRepositoryExample": {
      "name": "boat",
      "fullName": "cloudmesh-community/boat",
      "url": "https://github.com/cloudmesh-community/boat",
      "description": "S.T.A.R. boat"
    },
    "cloudRepositoryExample": {
      "name": "cm",
      "fullName": "cloudmesh-community/cm",
      "url": "https://github.com/cloudmesh-community/cm",
      "description": "Cloudmesh v4"
    }
  }
}
```

7.3.4 Variables

Variables are used to pass dynamic values to queries. Instead of passing hard-coded values to a query, variables can be defined for these values. Now these variables can be passed to queries.

Variables can be passed to GraphQL queries directly through commandline. Please note that we pretty print the json output with python's `json.tool`. So it is not actually part of the query, but convenient to format the output. Try to see the difference with and without the pipe to `json.tool`

```
curl -X POST \
-H "Content-Type: application/json;" \
-d '{"query": "{ repository (name: $name) { name url } }", "variables": \
```

```
{ "name": "book" }' \
http://localhost:8000/graphql/ | python -m json.tool
```

In case you use GraphQL, variables can be defined in the *Query Variables* panel at left bottom of the *GraphiQL* client. It is defined as a JSON object and this is how it looks like

```
{
  "name": "book"
}
```

and it can be used in the query like this

```
{
  repository(name: $name) {
    name
    url
  }
}
```

which will result in the response

```
{
  "data": {
    "repository": {
      "name": "book",
      "url": "https://github.com/cloudmesh-community/book"
    }
  }
}
```

7.3.5 Directives

Directives are used to change the structure of queries at runtime using variables. Directives provide a way to describe additional options to GraphQL executors. Currently the core GraphQL specification supports two directives

- `@Skip` (if: Boolean) - It skips the field if argument is true
- `@Include` (if: Boolean) - It includes the field if argument is true

To demonstrate its usage, we define the variable `isAdmin` and assign a value of `true` to it.

```
{
  "isAdmin": true
}
```

This variable is passed as an argument `showOwnerInfo` to the query. This argument is in turn passed to `@include` directive to determine whether to include the `ownerInfo` sub-query.

```
{
```

```

repositories(showOwnerInfo: $isAdmin) {
  name
  url
  ownerInfo @Include(if: $showOwnerInfo) {
    name
  }
}

```

Since we have defined `showOwnerInfo` as `true`, the response includes `ownerInfo` data.

```

{
  "data": {
    "repositories": [{
      "name": "book",
      "url": "https://github.com/cloudmesh-community/book",
      "ownerInfo": {
        "name": "cloudmesh-community"
      }
    }]
  }
}

```

7.3.6 Mutations

Mutations are used to modify the server side data. To demonstrate this, let us look at the query and the data to be passed along with it

```

mutation CreateRepositoryForCommunity($community: Community!, $repository: Repository!) {
  createRepository(community: $community, repository: $repository) {
    name
    url
  }
}

```

```

{
  "community": "cloudmesh-community",
  "repository": {
    "name": "cm-burn",
    "url": "https://github.com/cloudmesh/cm-burn"
  }
}

```

The response will be as follow, indicating that a repository has been added.

```

{
  "data": {
    "createRepository": {
      "name": "cm-burn",
      "url": "https://github.com/cloudmesh/cm-burn"
    }
  }
}

```

7.3.7 Query Validation

GraphQL is a language with strong type system. So requesting and providing wrong data will generate an error.

For example the query

```
{
  repositories {
    name
    url
    type
  }
}
```

will give the response

```
{
  "errors": [{
    "message": "Cannot query field \"type\" on type \"Repository\".",
    "locations": [{
      "line": 5,
      "column": 3
    }]
  }]
}
```

In an application we need to validate the user input. If it is invalid we can use the `GraphQLError` class or Python exceptions to raise validation errors.

Let us take an example of mutation query. We want to validate whether repository name is empty or not. We can use `GraphQLError` to raise validation error from our mutation function like this

```
def mutate(self, info, url, name, full_name, description):
    if not name:
        raise GraphQLError('Repository name is required')
    repository = Repository(url=url,
                            name=name,
                            full_name=full_name,
                            description=description)
    repository.save()
```

7.4 GRAPHQL IN PYTHON

We will cover a basic server implementation with schema and queries to fetch and mutate data.

To develop a GraphQL server in Python we will use `Django` as Python web framework and the `Graphene` library which allows us to develop GraphQL APIs. Naturally other frameworks such as Flask could be used, but for this example we focus on Django. The installation for Graphene and Django been already described at the beginning of this chapter. Our example is located in the book repository which you can clone with

```
$ git clone https://github.com/cloudmesh-community/book.git
```

The example itself is located in the directory

- [book/examples/graphql/cloudmeshrepo](#) - A graphql server example with local database

To execute the example you need to go in the specific directory. Thus, for cloudmeshrepo say

```
$ cd book/examples/graphql/cloudmeshrepo
```

Then you can execute following steps

```
$ pip install django-graphql-jwt==0.1.5
$ python manage.py migrate
$ python manage.py runserver
```

The last command will start a server on localhost and you can access it at

- <http://localhost:8000>

It will show you a graphical interface based on *GraphiQL*, allowing you to execute your queries in it.

7.5 DEVELOPING YOUR OWN GRAPHQL SERVER

If you want to create GraphQL server while using django as the web server backend yourself, you can start with following steps

```
$ mkdir -p example/graphql
$ cd example/graphql
$ pip install django-graphql-jwt==0.1.5
$ django-admin startproject cloudmeshrepo
$ cd cloudmeshrepo
$ python manage.py migrate
$ python manage.py runserver
```

The last command will start a server on the localhost and you can access it at the URL

- <http://localhost:8000>

It will show you the welcome page for django. Now open the file

```
cloudmeshrepo/cloudmeshrepo/settings.py
```

file under folder and append following to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # After the default packages
```

```
) 'graphene_django',
```

At the end of `settings.py` add following line

```
GRAPHENE = {  
    'SCHEMA': 'cloudmeshrepo.schema.schema',  
}
```

7.5.1 GraphQL server implementation

Clients can request for data to GraphQL server via GraphQL queries. They can also use mutations to insert data into GraphQL server's database. Django follows the principle of separating different modules in a project into apps. For this example, we will have two apps, one for Users and one for Repositories. For the demo purpose, we have decided not use backend such as MongoDB but instead we will use SQLite.

Django provides `startapp` utility to create blank app with some boilerplate code.

Go to the root dir of project and execute the following command which will create an app for repository.

```
python manage.py startapp repository
```

Now open `Repositories/models.py` and add the `Repository` model class.

```
class Repository(models.Model):  
    url = models.URLField()  
    name = models.TextField(blank=False)  
    full_name = models.TextField(blank=False)  
    description = models.TextField(blank=True)
```

Now open the file `cloudmeshRepository/settings.py` and append following line into `INSTALLED_APPS`

```
INSTALLED_APPS = (  
    # After the graphene_django app  
    'Repositories',  
)
```

Go to root folder and execute following commands. It will create table for new model

```
$ python manage.py makemigrations  
$ python manage.py migrate
```

Naturally, for us to demonstrate the server, we need to ingest some data into the

server. We can easily do this with the django shell while calling

```
$ python manage.py shell
```

Inside the shell, execute following command to create some example data. We have taken this data from github's API and used the repositories in the cloudmesh community at

- <https://api.github.com/users/cloudmesh-community/repos>.

You could use either `wget` or `curl` command to download this data through shell. As this data is huge, we have used a small subset for this example. You can have a python script, shell script or any other program to clean and remodel the data as per your need; the implementation details for the cleaning process is out of scope for this chapter.

```
from Repositories.models import Repository
Repository.objects.create(
    name="boat",
    full_name="cloudmesh-community/boat",
    url="https://github.com/cloudmesh-community/boat",
    description="S.T.A.R. boat")
Repository.objects.create(
    name="book", full_name="cloudmesh-community/book",
    url="https://github.com/cloudmesh-community/book",
    description="Gregor von Laszewski")
Repository.objects.create(name="cm",
    full_name="cloudmesh-community/cm",
    url="https://github.com/cloudmesh-community/cm",
    description="Cloudmesh v4")
Repository.objects.create(name="cm-burn",
    full_name="cloudmesh-community/cm-burn",
    url="https://github.com/cloudmesh/cm-burn",
    description="Burns many SD cards so we can build a Raspberry PI cluster")
exit()
```

Now create the file `Repositories/schema.py` with the following code. The code will create a custom type `Repository` and query with a *resolver function* for Repositories. The GraphQL server uses a resolver function to resolve the incoming queries. Queries can respond to only those fields or entities of schema for which resolver function has been defined. A `Resolver` function's responsibility is to return data for that specific field or entity. We will create one for Repositories list. When you query repositories, resolver function will return all the repositories objects from database.

```
import graphene
from graphene_django import DjangoObjectType

from .models import Repository

class RepositoryType(DjangoObjectType):
    class Meta:
        model = Repository
```

```
class Query(graphene.ObjectType):
    Repositories = graphene.List(RepositoryType)

    def resolve_Repositories(self, info, **kwargs):
        return Repository.objects.all()
```

Next create the file `cloudmeshRepository/schema.py` with following code. It just inherits the query defined in Repositories app. This way we are able to isolate schema to their apps.

```
import graphene

import Repositories.schema

class Query(Repositories.schema.Query, graphene.ObjectType):
    pass

schema = graphene.Schema(query=Query)
```

7.5.2 GraphQL Server Querying

Next, we create a Schema and use it within *GraphiQL* which is a playground for GraphQL queries. Open the file `cloudmeshrepository/urls.py` and append the following code

```
from django.views.decorators.csrf import csrf_exempt
from graphene_django.views import GraphQLView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('graphql/', csrf_exempt(GraphQLView.as_view(graphiql=True))),
]
```

Start your server using the command

```
$ python manage.py runserver
```

Now open in your browser the URL

- <http://localhost:8000/graphql>

You will see *GraphiQL* window. In the left pane you can add queries. Let us add the following query

```
{
  repositories {
    name
    fullName
    url
    description
  }
}
```

In the right pane you will see following output

```
{
  "data": {
    "repositories": [
      {
        "name": "boat",
        "fullName": "cloudmesh-community/boat",
        "url": "https://github.com/cloudmesh-community/boat",
        "description": "S.T.A.R. boat"
      },
      {
        "name": "book",
        "fullName": "cloudmesh-community/book",
        "url": "https://github.com/cloudmesh-community/book",
        "description": "Gregor von Laszewski"
      },
      {
        "name": "cm",
        "fullName": "cloudmesh-community/cm",
        "url": "https://github.com/cloudmesh-community/cm",
        "description": "Cloudmesh v4"
      },
      {
        "name": "cm-burn",
        "fullName": "cloudmesh-community/cm-burn",
        "url": "https://github.com/cloudmesh/cm-burn",
        "description": "Burns many SD cards so we can build a Raspberry PI cluster"
      }
    ]
  }
}
```

7.5.3 Mutation example

Similar to a query, you can add a mutation to create your own data. To achieve this, add a `CreateRepository` class for new repository object which will inherit from graphene's `Mutation` class. This class will accept a new repository as an argument. Please see the following code snippet which is added to

`repositories/models.py`.

```
class CreateRepository(graphene.Mutation):
    url = graphene.String()
    name = graphene.String()
    full_name = graphene.String()
    description = graphene.String()

    class Arguments:
        url = graphene.String()
        name = graphene.String()
        full_name = graphene.String()
        description = graphene.String()

    def mutate(self, info, url, name, full_name, description):
        repository = Repository(url=url, name=name,
                                full_name=full_name,
                                description=description)

        repository.save()

    return CreateRepository(url=repository.url,
                             name=repository.name,
                             full_name=repository.full_name,
                             description=repository.description)
```

Similar to A Query, add a `Mutation` class in the repository's schema in

repositories/schema.py.

```
class Mutation(graphene.ObjectType):
    create_repository = CreateRepository.Field()
```

Now you can run the following mutation on *GraphiQL* to add a new repository

```
mutation {
  createRepository (
    url: "https://github.com/cloudmesh-community/repository-test",
    name: "repository-test",
    fullName: "cloudmesh-community/repository-test",
    description: "Test repository"
  ) {
    url
    name
    fullName
    description
  }
}
```

This will insert a new repository *repository-test* and also immediately return its inserted data fields (`url`, `name`, `fullName`, `description`).

```
{
  "data": {
    "createRepository": {
      "url": "https://github.com/cloudmesh-community/repository-test",
      "name": "repository-test",
      "fullName": "cloudmesh-community/repository-test",
      "description": "Test repository"
    }
  }
}
```

7.5.4 GraphQL Authentication

There a number of ways to add authentication to your GraphQL server

- We can add a REST API endpoint which will take care of authenticating the user and only the logged in users can make GraphQL queries. This method can also be used to restrict only a subset of GraphQL queries. This is ideal for existing applications, which have REST endpoints, and which are trying to migrate over to GraphQL.
- We can add basic authentication to the GraphQL server which will just accept credentials in raw format and once authenticated, logged in user can start GraphQL querying
- We can add JSON Web Token authentication to GraphQL server, since most of the applications these days are stateless.

7.5.5 JSON Web Token Authentication

Next we focus on the JSON web token (JWT) authentication. It is typically preferred as it provides a more secure and sophisticated way of authentication. As part of the authentication process, a client has to provide a username and a password. A limited life time token is generated that is used during the authentication process. Once the token is generated, it needs to be provided with each subsequent GraphQL API call to assure the authentication is valid.

JWT tokens are bearer tokens which need to be passed in HTTP authorization header. JWT tokens are very safe against CSRF attacks and are trusted and verified since they are digitally signed.

The advantage of using frameworks such as the python implementation of GraphQL is that it can leverage existing authentication modules, so we do not have to develop them ourselves. One such module is *JSON Web Token Authentication* or **JWT Authentication*. To use this module, please add it to the `settings.py` file as follows

```
MIDDLEWARE = [
    'graphql_jwt.middleware.JSONWebTokenMiddleware',
]

AUTHENTICATION_BACKENDS = [
    'graphql_jwt.backends.JSONWebTokenBackend',
    'django.contrib.auth.backends.ModelBackend',
]
```

Add the Token mutation to `cloudmeshrepo/schema.py`.

```
class Mutation(users.schema.Mutation, repositories.schema.Mutation, graphene.ObjectType):
    token_auth = graphql_jwt.ObtainJSONWebToken.Field()
```

Run the server using `runserver` command and fire the token mutation providing username and password. You can either run this mutation on *GraphiQL* or using `curl` command.

For *GraphiQL* run this on query panel.

```
mutation {
  tokenAuth (username:"user1", password:"Testing123") {
    token
  }
}
```

Or if you are on bash shell, use this `curl` command

```
curl -X POST \
-H "Content-Type: application/json;" \
-d '{"query": "{ mutation { tokenAuth (username: \"user1\", ' \
' password: \"Testing123\") { token } } }'" \
http://localhost:8000/graphql/ | python -m json.tool
```

This will create a token for us to use in our subsequent calls.

```
{
  "data": {
    "tokenAuth": {
      "token": "eyJ0eXAiOiJKV1... (cut to fit in line)"
    }
  }
}
```

The JWT library comes with a built-in directive called *login_required*. You can add this to any of your Query resolvers to prevent unauthenticated access. We have annotated it to the `resolve_repositories` which means it will throw authentication error to query which does not have JWT token passed. Whenever a valid JWT token is present in the query, it is considered as authenticated or logged in request, and data will be served only to these queries.

```
from graphql_jwt.decorators import login_required
...

class Query(graphene.ObjectType):
    repositories = graphene.List(RepositoryType)

    @login_required
    def resolve_repositories(self, info, **kwargs):
        return Repository.objects.all()
```

Now if you try to query our repositories from GraphQL, you will see this error

```
{
  "errors": [
    {
      "message": "You do not have permission to perform this action",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "repositories"
      ]
    }
  ],
  "data": {
    "repositories": null
  }
}
```

Henceforth you need to pass token with every repository query. This token needs to be passed as header. Unfortunately, the *GraphiQL* UI client does not support this. Hence you can use either a curl query from command line or more advanced GraphQL clients that support authentication.

7.5.5.1 Using Authentication with Curl

To use authentication with curl, you can pass the token to the command. For simplicity we created a TOKEN environment variable in which we store the token so it is easier for us to refer to it in our examples.

```
export TOKEN=eyJ0eXAiOiJKV1... (cut to fit in line)
curl -X POST \
-H "Content-Type: application/json;" \
-H "Authorization: JWT $TOKEN" \
-d '{"query": "{ repositories { url } }"}' \
http://localhost:8000/graphql/ | python -m json.tool
```

The result obtained from running this command is:

```
{"data":{"repositories":[
  {"url":"https://github.com/cloudmesh-community/boat"},
  {"url":"https://github.com/cloudmesh-community/book"},
  {"url":"https://github.com/cloudmesh-community/cm"},
  {"url":"https://github.com/cloudmesh/cm-burn"},
  {"url":"https://github.com/cloudmesh-community/vineet-test-1"},
  {"url":"https://github.com/cloudmesh-community/vineet-test"}
]}}
```

7.5.5.2 Expiration of JWT tokens

JWT tokens have a time-to-life and expire after a while. This is controlled by the GraphQL server and is usually communicated to the client in transparent documented fashion.

If the token is about to expire, you can call the `refreshToken` mutation to refresh the Token and to return the refreshed token to the client. However, if the token has already expired we will need to request a new token by calling `tokenAuth` mutation.

More information about JWT tokens can be found at [56] and the GraphQL authentication page at [57].

7.5.6 GitHub API v4

GraphQL has made already an impact in the cloud services community. In addition to Facebook, Twitter and Pinterest, *GitHub* is now also providing a GraphQL interface, making it an ideal example for us.

GitHub has implemented as part of its API v4 also GraphQL which allows you to query or mutate data of repositories that you can access via `github.com`. To demonstrate its use, we will use *GraphiQL*.

To access the information, we need an OAuth token to access GitHub API. You can generate an OAuth token by following the steps listed at

- <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>

Next we demonstrate the use of Github within a GraphQL browser called Open *GraphiQL*. First you need to click edit headers at upper-right corner and add a new header with key *Authorization* and value *Bearer your_token*.

Next you enter the URL

- <https://api.github.com/graphql>

in the GraphQL endpoint textbox and keep the method as *POST* only. To test if the changes have been applied successfully you can use the query

```
query {
  viewer {
    login
    name
  }
}
```

The query gives the following response

```
{
  "data": {
    "viewer": {
      "login": "**Your GitHub UserId*",
      "name": "**Your Full Name*"
    }
  }
}
```

To get a list of our own repositories add following query

```
query($number_of_repositories:Int!) {
  viewer {
    name
    repositories(last: $number_of_repositories) {
      nodes {
        name
      }
    }
  }
}
```

To limit the responses we can define a use the variable `number_of_repositories`

```
{
  "number_of_repositories": 3
}
```

The query gives the following response

```
{
  "data": {
    "viewer": {
      "name": "*your name*",
      "repositories": {
        "nodes": [
          {
            "name": "*Repository 1*"
          },
          {
            "name": "*Repository 2*"
          },
          {
            "name": "*Repository 3*"
          }
        ]
      }
    }
  }
}
```

To add a comment using mutation we need to get the issue `id` with the query

```
{
  repository(owner:"MihirNS", name:"Temp_Repository") {
    issue(number: 1) {
      id
    }
  }
}
```

The query gives the following response

```
{
  "data": {
    "repository": {
      "issue": {
        "id": "MDU6SXNzdWUzNjUxMDIwOTE="
      }
    }
  }
}
```

Now we can use the id as `subjectId` for mutation to add a comment to an issue with the query

```
mutation AddComment {
  addComment(input:{subjectId:"MDU6SXNzdWUzNjUxMDIwOTE=",body:"This comment is done using GitHub API v4"}) {
    commentEdge {
      node {
        repository{
          nameWithOwner
        }
        url
      }
    }
  }
}
```

The query gives the following response

```
{
  "data": {
    "addComment": {
      "commentEdge": {
        "node": {
          "repository": {
```

```
    "nameWithOwner": "MihirNS/Temp_Repository"  
  },  
  "url": "https://github.com/MihirNS/Temp_Repository/issues/1#issuecomment-425620312"  
}  
}  
}
```

7.6 DYNAMIC QUERIES WITH GRAPHQL

The previous examples served data to and from a database. However, often we need to access dynamic data that is provided through function or system calls.

For this reason we like you to be reminded about the Section describing the [resolver function](#). It allows us to add functions on the server side that return the data from various data sources.

It is similar in nature than our example in the REST OpenAPI section, where we associate call backs that execute dynamic operations. More information about the functionality in REST is provided in the Section [OpenAPI Specification](#) as part of the [path definition](#).

7.7 ADVANTAGES OF USING GRAPHQL

Unlike REST APIs, only the required data is fetched minimizing the data transferred over network.

Seperation of concern is achieved between client and server. Client requests data entities with fields needed for the UI in one query request while server knows about the data structure and how to resolve the data from its sources which could be database, web service, microservice, external APIs, etc.

Versioning is simpler than REST, since we only have to take care of it when we want to remove any of the fields. We can even introduce the property of a deprecated field for a while to inform the service users. At a later time the field could be entirely be removed.

```
type Car {  
  id: ID!  
  make: String  
  description: String @deprecated(reason: "Field is deprecated!")  
}
```

7.8 DISADVANTAGES OF USING GRAPHQL

GraphQL query can get very complex. A client may not necessarily know how expensive the queries can be for server to go and gather the data. This can be overcome by limiting, for example, the query depth and recursion.

Caching gets pretty tricky and messy in case of GraphQL. In REST, you can have separate API url for each resource requested, caching can be done at this resource level. However, in GraphQL you can have different queries but they can operate over a single API url. This means that caching needs to be done at the field level introducing additional complexity.

7.9 CONCLUSION

GraphQL is gaining momentum as growing and the integration into services such as Github, Pinterest, Intuit, Coursera, and Shopify, demonstrates this. Many GraphQL editors, IDEs and packages have recently been developed.

In general there are several reasons to use GraphQL due to its simplicity and flexibility. It also fits well with the microservices architecture which is a new trend in cloud architectures. With that being said, REST APIs still have it is own place and may prove better choice in certain use cases. Both REST and GraphQL have some tradeoffs which need to be understood before making a choice between the one or the other. Github shows that both can be used.

7.9.1 Resources

- Official documentation of Github API v4 is available at [\[58\]](#)
- More GraphQL Python examples available at [\[59\]](#)

7.10 EXCERSISES

E.GraphQL.1:

Github provides a good playground for experimenting with GrapohQL. Develop a command in python that lists all repositories

of an organization. Expand upon this while listing through options all members of each repository. build your own extension. Use docopts or easier just us `cloudmesh sys generate` to create a command `cms git list ORGANIZATION [--users]`

E.GraphQL.2:

The chapter shows you how to develop a GraphQL server with Django. Instead of Django we like you to use Flask. Develop a documented section showcasing this. Use the resolver class to demonstrate a dynamic information example such as the cpu type. Showcase integration with mongoengine and dynamic information retrieval form another information source.

See an example

- https://github.com/graphql-python/graphene-mongo/tree/master/examples/flask_mongoengine
- <https://graphene-mongo.readthedocs.io/en/latest/tutorial.html>

E.GraphQL.3:

Develop a GraphQL server and client that queries your CPU information through a dynamic query using a resolver

E.GraphQL.4: OpenStack VMS

Develop a GraphQL server that returns the information of running virtual machines on OpenStack

E.GraphQL.5: OpenStack Azure

Develop a GraphQL server that returns the information of running virtual machines on OpenStack

E.GraphQL.6: OpenStack Aws

Develop a GraphQL server that returns the information of running virtual machines on OpenStack

E.GraphQL.7: Cloud Service

Pick a Cloud Service and develop a GraphQL interface for it.

E.GraphQL.8: Cloudmesh

Develop a cloudmesh framework that uses all previous clouds while returning the information of all running VMS in a Web page. YOU are allowed to make the Web page beautiful with HTML5 and/or JavaScript if you have the background to do so. Contact Gregor if you like to work on this for your project.

8 HYPERVISOR

8.1 VIRTUALIZATION



Learning Objectives

- Gain understanding of the basic the concepts of virtualization
 - Understand what a virtual machine is
 - Understand what a Hypervisor is
-

Virtualization is one of the important technologies that started the cloud revolution. It provides the basic underlying principles for the development and adoption of clouds. The concept, although old and already used in the early days of computing, has recently been exploited to lead to better utilization of servers as part of data centers, but also the local desktops.

Virtualization enables to execute multiple applications in such a way that the applications seem to run independently form each other in their own virtualized context.

Examples of the usefulness of virtualization include testing of applications and run experiments on a different operating system than the one on our host computer. To enable this we need virtual machines.

8.1.1 Virtual Machines

We define a virtual machine as follows:

A virtual machine (VM) is a software-based emulation of a computer system. This can include process virtualization and physical computer virtualization such as running an operating system. Multiple virtual machines share the resources of the computer or system on which they run.

We distinguish the following types of Virtual machines

- **System Virtual Machines** or **Hardware Virtual Machine**, which is the virtualization of the operating system providing a complete system platform environment emulating the hardware. Here we essentially run another operating system on top of the existing OS while using a software abstraction between them allowing the virtualization.

Examples of such virtualization include VirtualBox and VMWare.

- **Process Virtual Machines** or **Application Virtual Machine** which provides a platform independent programming environment that abstracts the details of the underneath hardware or OS from software or application runtime.

Examples of such virtual machines include Java Virtual Machine (JVM) and the .NET Framework

Next we will be analyzing the system machine virtualization in more detail, as they are one of the reasons for the clouds revolution.

8.1.2 System Virtual Machines

The use of a system as a virtual machine has its clear advantages for the cloud. We distinguish two main ways of system virtualizing:

- **Bare-metal Virtualization** in which the virtual machine monitor is installed directly on top of the hardware so that it has direct access to the underlying hardware. It hosts the operating system. The VMM is also called hypervisor. We also use for bare-metal supporting VMM the term *Type 1 hypervisor*.
- **Hosted Virtualization** in which the base operating system is installed on the hardware. Here a virtual machine monitor (VMM) is installed on top of the host OS allowing the users to run other operating systems on the VMM. In addition, the Virtual Machine Monitor or *Hypervisor* manages the deployments of potentially multiple virtual machines on top of the underlying Operating system. We also use for hosted VMM the term *Type 2*

hypervisor.

In either case the functionality a virtual machine is supported through configuration files, specifications, and access to the physical resources either directly or indirectly through the host OS. A virtual machine provides the same functionality as a physical computer, but with the advantage that through virtualization they are portable, can be managed and provide increased security while shielding the underlying OS from harmful actions. As a virtual machine is in principle a program, it consists of several files including a configuration file, virtual disk files, virtual RAM, and a log file. Virtual machines are configured to run a virtual operating system that allows applications to run on them. Each virtual machine has its own copy of the OS making it independent and more secure.

End users and developers will benefit from using virtual machines in case they need to operate or support on different hardware or porting software on it.

8.1.3 Hosted Virtualization

As in the hosted virtualization the guest operating system accessed the underlying hardware through the host OS, it usually has limited access to the hardware as defined by the host OS. This allows the host OS to impose policies that govern the operation of multiple guest OS concurrently. This includes management and scheduling of processes, memory, I/O operations to assign them appropriately to the guest OS. Through this mechanism the hypervisor provides an emulation of available hardware to each Virtual Machine run on top of it in time-sharing fashion for resource constrained or resource shared activities.

As example, the hypervisor has the ability to present generic I/O devices and it has no access to non-generic I/O devices. Generic I/O devices are network, interface cards, CD-ROMs. Examples for non-generic I/O devices are PCI data acquisition card, etc. However with appropriate driver support even such devices could be made accessible to the VMs.

Often we also find that hosted virtualization supports connected USB drives in the VMs which becomes very practical for USB attached devices needed in storage, or even edge computing applications.

Advantages of Hosted Virtualization include

- Multiple Operating systems run on separate virtual machines on a VMM.
- Different Operating systems run on separate virtual machines on a VMM.
- Hardware level driver support is controlled by VMM, allowing an isolation of certain security aspects for accessing the hardware.
- Installation of software can be done by the owner of the virtual machine and does not have to be conducted by the provider of the hypervisor.

Disadvantages of Hosted Virtualization include

- Increased resource requirements as the Guest OS is running a full copy of the OS. In its worst case this will lead to a significant performance reduction while using resources that are in contention.
- The user of hypervisors must be familiar with operating system management and security to ensure it is safe to use.

8.1.4 Summary

To showcase how these technologies relate to each other you may review [Figure 34](#)

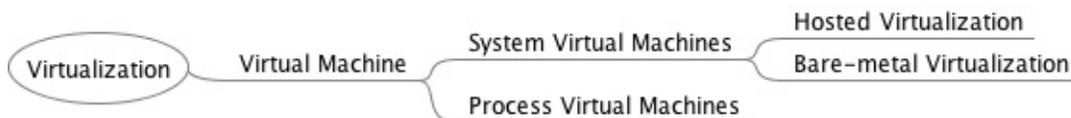


Figure 34: Virtualization Taxonomy [60]

We summarize the following *hypervisor* types:

- Type-1 hypervisors supporting native or bare-metal. They run directly on the host's hardware to control the hardware and to manage guest operating systems.
- Type-2 hypervisors supporting hosted virtualization. They run on a conventional operating system (OS) just as other computer programs do. A guest operating system runs as a process on the host.

8.1.5 Virtualization Approches

Next we look at different virtualization approaches that relate to resource utilization.

8.1.5.1 Full virtualization

When looking at virtualization we often identify it with being a full virtualization. The hypervisor provides a full abstraction of the hardware exposed to the guest OSs. In this case, the guest OSs the virtual machine just run without any special modification on the host OS. It just looks like an independent running computer [60].

8.1.5.2 Paravirtualization

Para – alongside/partial – virtualization is developed to improve performance by interacting between the OS and the hypervisor. This is done for complex and time-consuming tasks that otherwise could not be managed by the VMM manager. Commands sent from the OS to the hypervisor are called *hypercalls* [60].

8.1.6 Virtualization Technologies

In this section we cover introduction to underlying virtualization technologies used on some mainstream platforms.

Cloud providers, such as AWS, Azure, and Google, and OpenStack use for example QEMU and KVM technologies for compute instance virtualization.

8.1.6.1 Selected Hardware Virtualization Technologies

8.1.6.2 AMD-V and Intel-VT

The hardware virtualization support enabled by AMD-V and Intel VT technologies introduces virtualization in the x86 processor architecture. According to Intel, Intel Hyper-Threading Technology allows a single processor to execute two or more separate threads concurrently. When it is enabled, multi-threaded software applications can execute their threads in parallel, thereby

improving their performance.

8.1.6.3 I/O MMU virtualization (AMD-Vi and Intel VT-d)

The term IOMMU is an abbreviation for input–output memory management unit. An IOMMU allows through virtual addresses to interface with physical addresses, allowing external direct-memory-access–capable IO devices to interface with the main memory [61]. AMD’s I/O Virtualization Technology (AMD-Vi) was originally called *IOMMU*.

To use Intel’s *Virtualization Technology for Directed I/O* (VT-d), both the motherboard chipset and system firmware (BIOS or UEFI) need to fully support the IOMMU I/O virtualization functionality for it to be usable [62].

8.1.6.4 Selected VM Virtualization Software and Tools

A number of noteworthy virtualization software and tools exist which make the development and use of virtualization on the hardware possible. They include

- Libvirt
- KVM
- Xen
- Hyper-V
- QEMU
- VMWare
- VirtualBox

We will be discussing them next.

8.1.6.4.1 Libvirt

[Libvirt](#) is an library with an API for managing virtualization solutions such as provided by KVM and Xen. It provides a common management API for them, allowing uniform, cross-hypervisor interfaces for higher-level management tools. `Libvirt` provides a toolkit to manage virtualization hosts and supports a wide set of languages, such as C, Python, Perl, and Java. Drivers are the basic building block for libvirt functionality to support the capability to handle

specific hypervisor driver calls. Drivers are discovered and registered during connection processing as part of the `virInitializeAPI`. Each driver has a registration API which loads up the driver specific function references for the libvirt APIs to call. The following is a simplistic view of the hypervisor driver mechanism. Furthermore, it provides APIs for management of virtual networks and storage on the VM Host Server. The configuration of each VM Guest is stored in an XML file [63]. The official website for `libvirt` is located at

- <https://libvirt.org/>

8.1.6.4.2 QEMU

QEMU is a virtualization technology emulator that allows you to run operating systems and Linux distributions on your current system without installing them or burn their ISO files. When used as a machine emulator, QEMU can run OSs and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). By using dynamic translation, it achieves very good performance. QEMU provides two generic functions. One of them is open source machine emulator and the other is a virtualizer.

- *Machine emulation:* using it as a machine emulator it runs the OSs and programs designed for one machine on a different machine of potential different architecture. It uses dynamic translation through which it achieves very good performance.
- *Virtualizer:* Using it as a virtualizer it executes the guest code directly on the host CPU. This enables QEMU to achieve near native performance.

Once QEMU has been installed, it should be ready to run a guest OS from a disk image. This image is a file that represents the filesystem and OS on a hard disk. From the perspective of the guest OS, it actually is a file on hard disk, and it can create its own filesystem on the virtual disk.

QEMU supports either XEN or KVM to enable virtualization. With the help of KVM, QEMU can virtualize x86, server and embedded PowerPC, 64-bit POWER, S390, 32-bit and 64-bit ARM, and MIPS guests according to the [QEMU Wiki](#).

Useful links include the following:

- An extensive manual is provided at <https://qemu.weilnetz.de/doc/qemu-doc.html>.
- QEMU can be downloaded from <http://www.qemu.org/download/>.
- A collection of images for testing purposes is provided at https://wiki.qemu.org/Testing/System_Images

An example for using QEMU is provided in Section [Virtual Machine Management with QEMU]{???

8.1.6.4.3 KVM

KVM, or Kernel-based Virtual Machine is a popular open-source hypervisor solution. It was released as a virtualization solution for Linux based systems, and later was merged into Linux Kernel since version 2.6.20. It was originally supporting x86 hardware with virtualization extensions (Intel VT or AMD-V), but later supporting of PowerPC and ARM were added. It supports a variety of different guest OSs, e.g., Windows family, Darwin (the core of MacOS), in addition to the different distros from various linux operating systems. The full supported guest list can be found at: http://www.linux-kvm.org/page/Guest_Support_Status

The full list of KVM features can be found here: http://www.linux-kvm.org/page/KVM_Features. Among them, some cool features include hot-plugging of hardware even CPU and PCI devices. It supports live migration of VMs too.

8.1.6.4.3.1 KVM vs QEMU

KVM includes a fork of the QEMU executable. The QEMU project focuses on hardware emulation and portability. KVM focus on the kernel module and interfacing with the rest of the userspace code. KVM comes with a `kvm-qemu` executable that just like QEMU manages the resources while allocating RAM, loading the code. However instead of recompiling the code it spawns a thread which calls the KVM kernel module to switch to guest mode. It then proceeds to

execute the VM code. When privileged instructions are found, it switches back to the KVM kernel module, and if necessary, signals the QEMU thread to handle most of the hardware emulation. This means that the guest code is emulated in a posix thread which can be managed with common Linux tools [64].

8.1.6.4.4 Xen

Xen is one of the most widely adopted hypervisors by IaaS cloud. It is supported by the earliest and still the most popular public cloud offering, i.e., Amazon Web Service (AWS). Eucalyptus, one open-source effort to replicate what AWS had to offer, and the then most popular private cloud software, supported Xen from the start. And later Openstack, the most popular open-source IaaS cloud software at present, also supports Xen.

Some notable features of Xen includes:

- Supporting x86-64 and ARM for host architecture.
- Supporting live migration of VMs between different physical hosts without losing the availability.

More detailed list can be found at https://wiki.xenproject.org/wiki/Xen_Project_Release_Features.

8.1.6.4.5 Hyper-V

Hyper-V is a product from Microsoft to support virtualization on systems running Windows. Hyper-V was originally released along with Windows Server 2008, with a separate free version with limited functionality. In later releases it adds more features, e.g., better support of Linux guest OS, live migration of VMs, etc.

Hyper-V is still getting a lot of popularity comparing to XEN and KVM which we attribute to the increasing presence of Microsoft's Azure cloud offering.

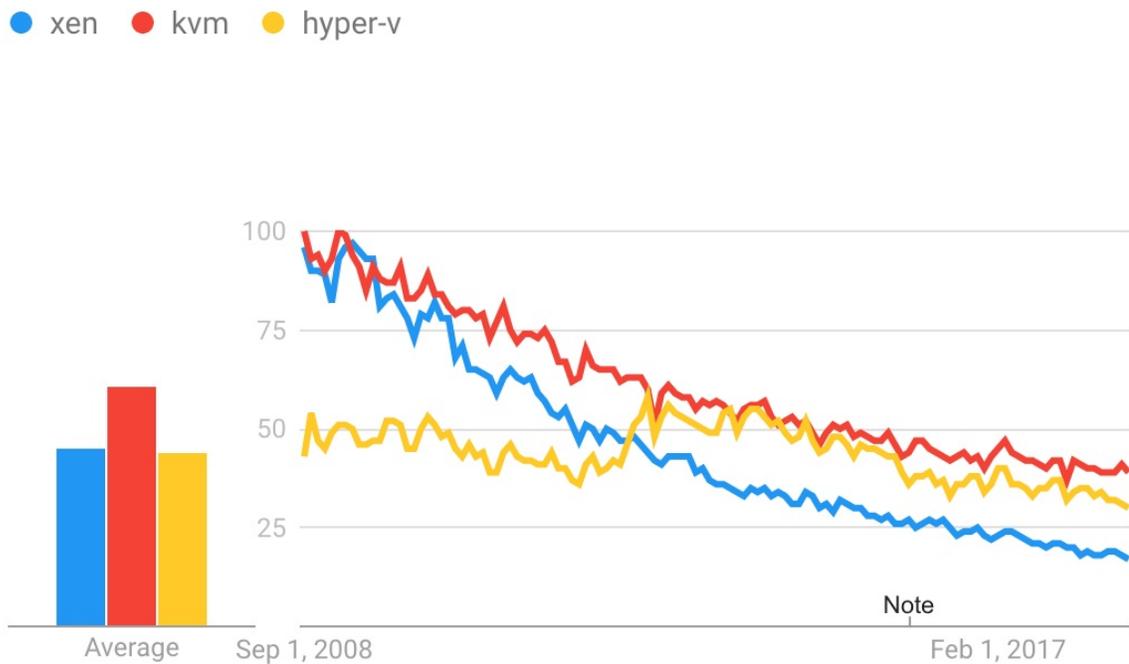


Figure 35: Popularity of KVM, Xen, and Hyper-V according to Google Trends [Source](#)

However overall the search popularity of hypervisors have been decreasing, as other lightweight virtualization solutions, i.e., container technologies become more main stream. We will covered them in a later chapter.

More detailed information about Hyper-V can be found at <https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture>

8.1.6.4.6 VMWare

VMware is well known for the company bringing hypervisors to the amss market. The company is now owned by Dell. It has developed the first type 2 hypervisor. Today VMWare offer type 1 hypervisors and type 2 hypervisors [65].

Because the initial software virtualized “hardware for a video adapter, a network adapter, and hard disk adapters” as well as “pass-through drivers for guest USB, serial, and parallel devices” [65] it provided an attractive solution for many to use it to run different OSs on their host computers. One important advantage is

that it does not rely on virtualization extensions to the x86 instruction set as it was developed before they became available. This means it can run on many other platforms. However this advantage is diminished with the ubiquitous available of these features in the hardware.

8.1.6.5 Parallels

Another interesting company offering hypervisors is Parallels. This company has two main products in that regards:

- Parallels Desktop for Mac, which for x86 machines allows users to run virtual machines independently using Windows, Linux, Solaris.
- Parallels Workstation for Microsoft Windows and Linux users which for x86 machines allows user to run virtual machines independently on the Windows host.

8.1.6.5.1 VirtualBox

VirtualBox is a free open-source hypervisor for x86 architectures. It is now owned by Oracle while transitioning from SUN which in turn acquired the original technology from Innotek.

One of the nice features for us is that VirtualBox is able to create and manage guest virtual machines such as Windows, Linux, BSD, OSx86 and even in part also macOS (on Apple hardware). Hence it makes it for us a very valuable tool while being able to run virtual machines on a local desktop or computer to simulate cloud resources without charging cost. In addition we find command line tools such as Vagrant that make the use convenient while not having to utilize the GUI or the more complex virtual box command interfaces. A guest additions package allows compatibility with the host OS, to for example allow window management between host and guest OS.

In Section [VirtualBox](#) we have provided a practical introduction to VirtualBox.

8.1.6.5.2 Wine – Wine is not an emulator

The next software that we introduce is actually not a hypervisor. However it is

very interesting as a contrast to the other approach. The term Wine has been originally introduced as an acronym for *Wine Is Not an Emulator*. In contrast to the other approaches Wine introduces a compatibility layer that allows to run Windows applications on a number of POSIX-compliant operating systems. This includes Linux, macOS and BSD. In contrast to using a virtual machine or emulator, Wine translates Windows API calls into POSIX calls [66]. Hence, it allows to pass the Windows API calls directly to operating system calls leading to good performance [66]. The disadvantage of this approach is that in the early days and till today some of the underlying calls may not be ported yet and may lead to applications not running. Those of us that wanted to run for example Microsoft Word on Linux or macOS, may remember that Wine was the tool we used to do so even before Word was released on macOS.

8.1.6.5.3 Comparison of some technologies

QEMU and KVM are better integrated in Linux and has a smaller footprint. This may result in better performance. VirtualBox is targeted as virtualization software and limited to x86 and amd64. As Xen uses QEMU it allows hardware virtualization. However, Xen can also use paravirtualization [67]. In the following table we summarize support for full- and paravirtualization

	XEN	KVM	VirtualBox	VMWare
Paravirtualization	yes	no	no	no
Full virtualization	yes	yes	yes	yes

8.1.6.6 Selected Storage Virtualization Software and Tools

Storage virtualization allows the system to integrate the logical view of the physical storage resources into a single pool of storage. Users are unaware while using virtual storage that it is not hosted on the same hardware resources, such as disks. Storage virtualization is done across the various layers that build state of the art storage infrastructures. This includes Storage devices, the Block aggregation layer, the File/record layer, and the Application layer. Most recently hosting files as part of the application layer in the cloud is changing how we approach data storage needs in the enterprise. A good example for a cloud based virtual storage is google drive. Other systems include Box, AWS3 and Azure.

8.1.6.7 Selected Network Virtualization Software and Tools

Network virtualization allows hardware and software network resources as well as network functionality to be combined into a single, software-defined administrative unit which is called a virtual network. We distinguish external network virtualization that combines many networks into a unifying network, and internal network virtualization that provides network functionality to the processes and containers running on a single server.

Note, that we will not cover this topic in this introductory class. However students can contribute a section or chapter

8.2 VIRTUAL MACHINE MANAGEMENT WITH QEMU

In this section we provide a short example on how to use QEMU. We will be starting with the installation, then create a virtual hard disk, install ubuntu on the disk and start the virtual machine. Next we will demonstrate how we can emulate a Raspberry Pi with QEMU. Lastly, we show how to use virsh.

8.2.1 Install QEMU

To install QEMU+KVM on Ubuntu/Linux Mint please use

```
$ sudo apt install qemu qemu-kvm libvirt-bin
```

On OSX QEMU can be installed with Homebrew

```
$ brew install qemu
```

8.2.2 Create a Virtual Hard Disk with QEMU

To create an image file with the size of 10GB and `qcow2` format (default format for QEMU images), run:

```
$ qemu-img create -f qcow2 testing-image.img 10G
```

Note that a new file called `testing-image.img` is now created at your home folder (or the place where you run the terminal). Note also that the size of this file is not 10 Gigabytes, it is around 150KB only; QEMU will not use any space unless

needed by the virtual operating system, but it will set the maximum allowed space for that image to 10 Gigabytes only.

8.2.3 Install Ubuntu on the Virtual Hard Disk

Now that we have created our image file, if we have an ISO file for a Linux distribution or any other operating system and we want to test it using QEMU and use the new image file we created as a hard drive, we can run:

```
qemu-system-x86_64 \  
-m 1024 \  
-boot d \  
-enable-kvm \  
-smp 3 \  
-net nic -net user \  
-hda testing-image.img \  
-cdrom ubuntu-16.04.iso
```

Explain the previous command part by part:

`-m 1024`:

Here we chose the RAM amount that we want to provide for QEMU when running the ISO file. We chose 1024MB here. You can change it if you like according to your needs.

`-boot -d`:

The boot option allows us to specify the boot order, which device should be booted first? `-d` means that the CD-ROM will be the first, then QEMU will boot normally to the hard drive image. We have used the `-cdrom` option as you can see at the end of the command. You can use `-c` if you want to boot the hard drive image first.

`-enable-kvm`:

This is a very important option. It allows us to use the KVM technology to emulate the architecture we want. Without it, QEMU will use software rendering which is very slow. That is why we must use this option, just make sure that the virtualization options are enabled from your computer BIOS.

`-smp 3`:

If we want to use more than 1 core for the emulated operating system, we can use this option. We chose to use 3 cores to run the virtual image which will make it faster. You should change this number according to your computer's CPU.

```
-net nic -net user:
```

By using these options, we will enable an Ethernet Internet connection to be available in the running virtual machine by default.

```
-hda testing-image.img:
```

Here we specified the path for the hard drive which will be used. In our case, it was the testing-image.img file which we created before.

```
-cdrom ubuntu-16.04.iso:
```

Finally we told QEMU that we want to boot our ISO file `ubuntu-16.04.iso`.

8.2.4 Start Ubuntu with QEMU

Now, if you want to just boot from the image file without the ISO file (for example if you have finished installing and now you always want to boot the installed system), you can just remove the `-cdrom` option:

```
$ qemu-system-x86_64 -m 1024 -boot d -enable-kvm -smp 3 -net nic -net user -hda testing-image.img
```

Please note QEMU `qemu-system-x86_64` emulates a 64-bit architecture.

8.2.5 Emulate Raspberry Pi with QEMU

First you have to download a pre-built kernel

```
$ wget https://raw.githubusercontent.com/dhruvvyas90/qemu-rpi-kernel/master/kernel-qemu-4.4.34-jessie
```

Next, you have to download the Raspbian image. Download a `.img` file from Raspbian website:

- <https://www.raspberrypi.org/downloads/raspbian/>

To start the emulator type in the following command to have QEMU emulate an ARM architecture:

```
$ qemu-system-arm -kernel ./kernel-qemu-4.4.34-jessie \  
-append "root=/dev/sda2 panic=1 rootfstype=ext4 rw" \  
-hda raspbian-stretch-lite.img \  
-cpu arm1176 -m 256 -machine versatilepb \  
-no-reboot -serial stdio
```

Please note that

- `kernel-qemu-4.4.34-jessie` is the pre-built kernel file.
- `raspbian-stretch-lite.img` is the Raspbian image file.

8.2.6 Resources

General

- Official website for `libvirt` is here: <https://libvirt.org/>
- Home page of KVM is here: https://www.linux-kvm.org/page/Main_Page
- QEMU home page: <https://www.qemu.org/>
- QEMU User Documentation: <https://qemu.weilnetz.de/doc/qemu-doc.html>
- Wikipedia page for QEMU: <https://en.wikipedia.org/wiki/QEMU>

Comparison

- <http://opensourceforu.com/2012/05/virtualisation-faceoff-qemu-virtualbox-vmware-player-parallels-workstation/>
- <https://stackoverflow.com/questions/43704856/what-is-the-difference-qemu-vs-virtualbox>

8.3 MANAGE VM GUESTS WITH VIRSH

`virsh` is a command line interface tool for managing guests and the hypervisor.

To initiate a hypervisor session with `virsh` :

```
virsh connect <name>
```

Where is the machine name of the hypervisor. If you want to initiate a read-only connection, append the previous command with `-readonly`.

To display the guest list and their current states with virsh:

```
virsh list [ --inactive | --all]
```

The `--inactive` option lists inactive domains (domains that have been defined but are not currently active). The `--all` option lists all domains, whether active or not.

A manual page can be found online at

- <https://linux.die.net/man/1/virsh>

A practical example of using virsh is provided at

- [Redhat Customer Portal: CHAPTER 26. MANAGING GUESTS WITH VIRSH](#)

9 IAAS

9.1 INTRODUCTION



Learning Objectives

- Review IaaS service by prominent cloud providers.
- Understand how to write Python programs on managing virtual machines with libcloud.
- Understand how to write Python programs on managing data services with libcloud.
- Experiment with cloud providers while practically testing them.



Be careful with your allocation.

[NIST](#) defines the term

Infrastructure as a Service (IaaS) as follows:

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

The key term is to *provision* fundamental computing resources. This means a user does not have to worry about managing the hardware allowing low level services such as the operating system or the network fabric to be the next higher interface for the user. The hardware fabric is managed by the cloud provider, while the operating system level and their connectivity with each other is

managed by the user.

We distinguish the following categories of infrastructure:

- compute resources
- network resources
- storage resources

We also like to remind you that such IaaS as parts of clouds can either be accessed public, private or in a hybrid fashion.

Within the next view section we will focus on some of the main providers of IaaS.

This includes

- Amazon Web Services
- Azure
- Google

Additionally, we also have

- Watson

which although provides IaaS focusses more on delivering AI platforms and related services to the community.

On the research side we will be focussing on

- Chameleon Cloud.

Some of the services listed provide a free small contingent of IaaS offerings that you can use. The use of this free tier will be sufficient to conduct this class.

A set of introductory slides is available at



[Virtual Machines \(21\)](#)

9.2 AMAZON WEB SERVICES

9.2.1 AWS Products

Amazon Web Services offers a large number of products that are centered around their cloud services. These services have grown considerably over the years from the core offering related to virtual machine (EC2) and datastorage (S3). An overview of them is provided by Amazon in the following document:

- <https://d0.awsstatic.com/whitepapers/aws-overview.pdf>

We list the product in screenshots from their Product Web page panel in Figure: [Figure 36](#), [Figure 37](#).

<p>Compute</p> <p>Amazon EC2 Amazon EC2 Auto Scaling Amazon Elastic Container Service Amazon Elastic Container Service for Kubernetes Amazon Elastic Container Registry Amazon Lightsail AWS Batch AWS Elastic Beanstalk AWS Fargate AWS Lambda AWS Serverless Application Repository Elastic Load Balancing VMware Cloud on AWS</p> <hr/> <p>Storage</p> <p>Amazon Simple Storage Service (S3) Amazon Elastic Block Storage (EBS) Amazon Elastic File System (EFS) Amazon Glacier AWS Storage Gateway AWS Snowball AWS Snowball Edge AWS Snowmobile</p> <hr/> <p>Database</p> <p>Amazon Aurora Amazon RDS Amazon DynamoDB Amazon ElastiCache Amazon Redshift Amazon Neptune AWS Database Migration Service</p> <hr/> <p>Migration</p> <p>AWS Migration Hub AWS Application Discovery Service AWS Database Migration Service AWS Server Migration Service</p>	<p>Networking & Content Delivery</p> <p>Amazon VPC Amazon CloudFront Amazon Route 53 Amazon API Gateway AWS Direct Connect Elastic Load Balancing</p> <hr/> <p>Developer Tools</p> <p>AWS CodeStar AWS CodeCommit AWS CodeBuild AWS CodeDeploy AWS CodePipeline AWS Cloud9 AWS X-Ray AWS Tools & SDKs</p> <hr/> <p>Management Tools</p> <p>Amazon CloudWatch AWS Auto Scaling AWS CloudFormation AWS CloudTrail AWS Config AWS OpsWorks AWS Service Catalog AWS Systems Manager AWS Trusted Advisor AWS Personal Health Dashboard AWS Command Line Interface AWS Management Console AWS Managed Services</p> <hr/> <p>Media Services</p> <p>Amazon Elastic Transcoder Amazon Kinesis Video Streams AWS Elemental MediaConvert AWS Elemental MediaLive AWS Elemental MediaPackage AWS Elemental MediaStore</p>	<p>Machine Learning</p> <p>Amazon SageMaker Amazon Comprehend Amazon Lex Amazon Polly Amazon Rekognition Amazon Machine Learning Amazon Translate Amazon Transcribe AWS DeepLens AWS Deep Learning AMIs Apache MXNet on AWS TensorFlow on AWS</p> <hr/> <p>Analytics</p> <p>Amazon Athena Amazon EMR Amazon CloudSearch Amazon Elasticsearch Service Amazon Kinesis Amazon Redshift Amazon QuickSight AWS Data Pipeline AWS Glue</p> <hr/> <p>Security, Identity & Compliance</p> <p>AWS Identity and Access Management (IAM) Amazon Cloud Directory Amazon Cognito Amazon GuardDuty Amazon Inspector Amazon Macie AWS Certificate Manager AWS CloudHSM AWS Directory Service AWS Key Management Service AWS Organizations AWS Single Sign-On AWS Shield</p>	<p>AR & VR</p> <p>Amazon Sumerian</p> <hr/> <p>Application Integration</p> <p>Amazon MQ Amazon Simple Queue Service (SQS) Amazon Simple Notification Service (SNS) AWS AppSync AWS Step Functions</p> <hr/> <p>Customer Engagement</p> <p>Amazon Connect Amazon Pinpoint Amazon Simple Email Service (SES)</p> <hr/> <p>Business Productivity</p> <p>Alexa for Business Amazon Chime Amazon WorkDocs Amazon WorkMail</p> <hr/> <p>Desktop & App Streaming</p> <p>Amazon WorkSpaces Amazon AppStream 2.0</p> <hr/> <p>Internet of Things</p> <p>AWS IoT Core Amazon FreeRTOS AWS Greengrass AWS IoT 1-Click AWS IoT Analytics AWS IoT Button AWS IoT Device Defender AWS IoT Device Management</p> <hr/> <p>Game Development</p> <p>Amazon GameLift Amazon Lumberyard</p> <hr/> <p>Software</p> <p>AWS WAF AWS Artifact</p> <hr/> <p>Mobile Services</p> <p>AWS Mobile Hub Amazon API Gateway Amazon Pinpoint AWS AppSync AWS Device Farm AWS Mobile SDK</p>
---	---	---	--

Figure 36: AWS Products 1 [Source](#)

<p>AWS Snowball AWS Snowball Edge AWS Snowmobile</p>	<p>AWS Elemental MediaTailor</p>	<p>AWS WAF AWS Artifact</p> <hr/> <p>Mobile Services</p> <p>AWS Mobile Hub Amazon API Gateway Amazon Pinpoint AWS AppSync AWS Device Farm AWS Mobile SDK</p>	<p>AWS Marketplace</p> <hr/> <p>AWS Cost Management</p> <p>AWS Cost Explorer AWS Budgets Reserved Instance Reporting AWS Cost and Usage Report</p>
--	----------------------------------	---	---

Figure 37: AWS Products 2 [Source](#)

Service offerings are grouped by categories:

- Compute
- Storage
- Database
- Migration
- Networking and Content Delivery
- Developer Tools
- Management Tools
- Media Services

- Machine Learning
- Analytics
- Security and Identity Compliance
- Mobile Services
- AR and VR
- Application Integration
- Customer Engagement
- Business Productivity
- Desktop and App Streaming
- Internet of Things
- Game Development
- Software
- Aws Core Management

Within each category you have several products. When choosing products from AWS it is best to start with the overview paper and identify products that can be of benefit to you. For our purpose we focus on the traditional Compute and Storage offerings. We list the products that are available In Sep. 2018 next.

9.2.1.1 Virtual Machine Infrastructure as a Services

Amazon offers a large number of services related to virtual machine management

- [Amazon EC2 \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/)
- [Amazon EC2 Auto Scaling \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/)

9.2.1.2 Container Infrastructure as a Service

Amazon offers the following container based services

- [Amazon Elastic Container Service \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/)
- [Amazon Elastic Container Service for Kubernetes \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/)
- [Amazon Elastic Container Registry \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/)

9.2.1.3 Serverless Compute using AWS Lambda

In addition to these services a number of additional compute services Serverless computing or FaaS is a new cloud computing paradigm that has are offered which you can find in the Appendix. This includes gained popularity recently. Serverless Computing with AWS Lambda. Serverless computing or Function as a Service (FaaS) is a new cloud computing paradigm that has gained popularity recently. AWS Lambda was one of the first serverless computing services that was made available to the public, Serverless computing allows users to run small functions in the cloud without having to worry about resource requirements. More information regarding AWS Lambda can be found in the following document

- <https://aws.amazon.com/lambda/>

9.2.1.4 Serverless Compute using AWS Lambda

Serverless computing or FaaS is a new cloud computing paradigm that has gained popularity recently.

9.2.1.5 Storage

AWS provides many storage services that users can leverage for developing applications and solutions. The next list showcases AWS storage services. Amazon offers the following storage services

- [Amazon Simple Storage Service \(S3\)](#)
- [Amazon Elastic Block Store \(EBS\)](#)
- [Amazon Elastic File System \(EFS\)](#)
- [Amazon Glacier](#)
- [AWS Storage Gateway](#)
- [AWS Snowball](#)
- [AWS Snowball Edge](#)
- [AWS Snowmobile](#)
- [AWS Marketplace](#)

9.2.1.6 Databases

AWS also provides many data base solutions. AWS has both SQL based

databases and NoSQL based databases. The next list shows the database services that AWS offers. And other database related services

- [Amazon Aurora](#)
- [Amazon RDS](#)
- [Amazon DynamoDB](#)
- [Amazon ElastiCache](#)
- [Amazon Redshift](#)
- [Amazon Neptune](#)
- [AWS Database Migration Service](#)
- [AWS Marketplace](#)

9.2.2 Locations

As the following figure shows: [Figure 38](#).

Global Network of Regions and Edge Locations

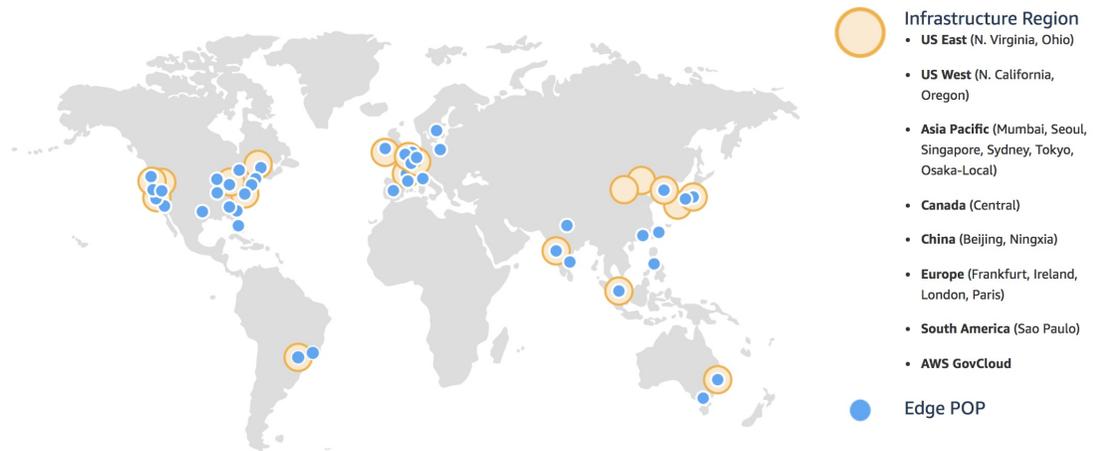


Figure 38: AWS-Locations [Source](#)

9.2.3 Creating an account

In order to create a AWS account you will need the following

- A valid email address
- A credit/debit card
- A valid phone number

First you need to visit the AWS [signup page \[Source\]\(https://aws.amazon.com/\)](https://aws.amazon.com/) and click “Create Free Account”. You will then be asked to provide some basic details including your email address as shown in the image: [Figure 39](#).

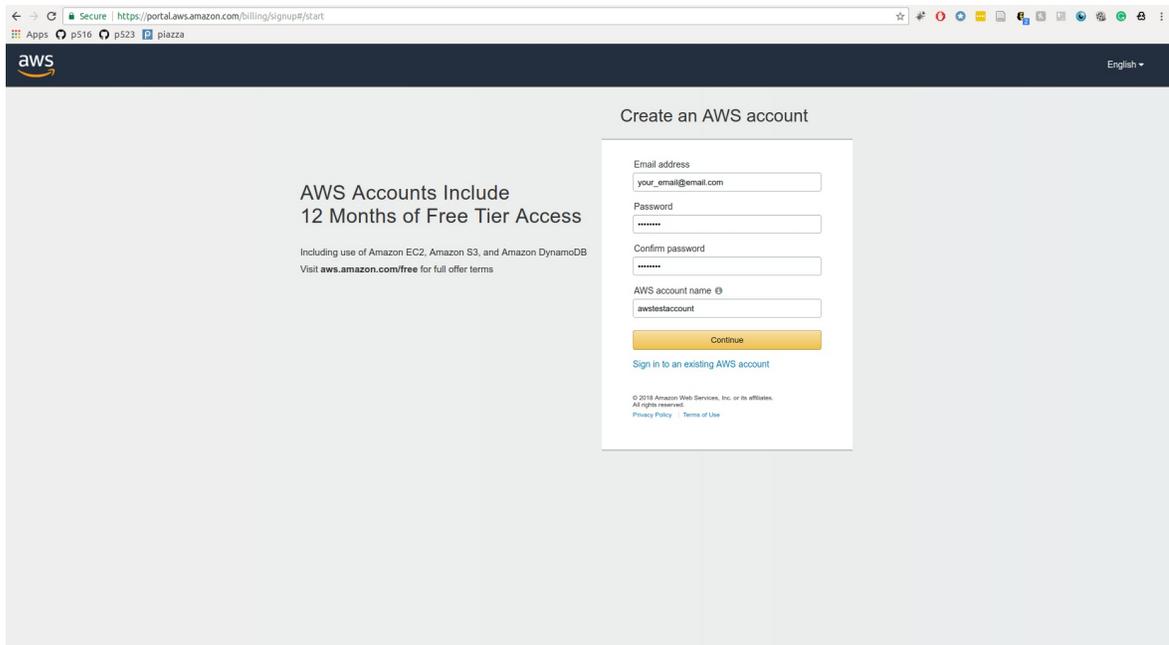


Figure 39: AWS Signup [Source](#)

Next you will be asked to provide further details such as your name, address and phone number. After the additional details have been provided. AWS will ask for credit/debit card details as shown: [Figure 40](#). They require this information to verify your identity and make sure they have a method to charge you if needed. However no charges will be applied to your credit/debit card unless you use the AWS services and exceed the free tier limits, which will be discussed in the next section.

The image shows a web browser window displaying the AWS Payment Information page. The browser's address bar shows the URL: <https://portal.aws.amazon.com/billing/signup#/paymentinformation>. The page features the AWS logo in the top left corner and a language dropdown menu in the top right corner set to "English".

The main content area is titled "Payment Information" and contains a form with the following fields and options:

- Credit/Debit card number:** A text input field.
- Expiration date:** Two dropdown menus, the first showing "08" and the second showing "2018".
- Cardholder's name:** A text input field.
- Billing address:** A section with two radio button options:
 - Use my contact address
2805 E 10th St Smith Research Center
Bloomington IN 47408
US
 - Use a new address
- Secure Submit:** A yellow button.

At the bottom of the form, there is a small copyright notice: "© 2018 Amazon Web Services, Inc. or its affiliates. All rights reserved." with links for "Privacy Policy", "Terms of Use", and "Sign Out".

Figure 40: Payment Information [Source](#)

After the credit/debit card has been verified AWS will use your phone number to verify your identity. Once you are logged into your account you will be able to sign into the console, from the link on the top right corner in your account. Once you are in the AWS console the services tab in the left top corner will allow you to access all the services that are available to you through AWS as shown in the image : [Figure 41](#).

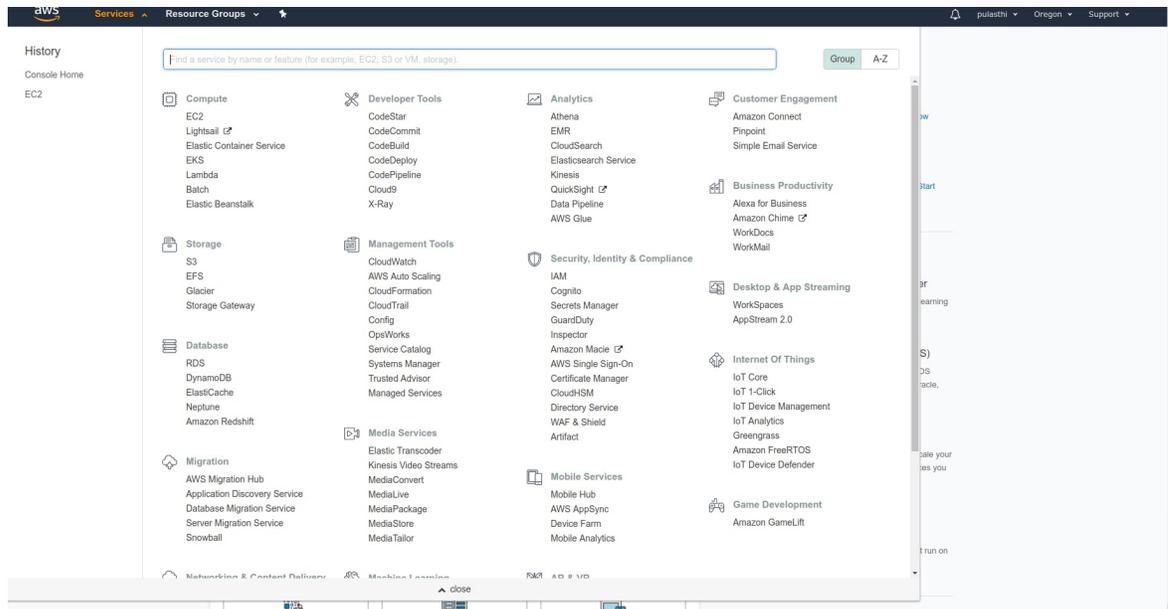


Figure 41: AWS Console [Source](#)

9.2.4 AWS Command Line Interface

9.2.4.1 Introduction

Amazon's CLI allows for programmatic interaction with AWS product through the command line. CLI provide many pre-built functions that allow for interaction with Amazon's Elastic Compute Cloud (EC2) instances and S3 storage.

9.2.4.2 Prerequisites

- [Linux](#)
- [Python](#)
- [PIP](#)
- [AWS Account](#)
- [AWS Key Pair](#)

9.2.4.2.1 Install CLI

Run the following code to install CLI.

```
pip install awscli
```

9.2.4.2.2 Configure CLI

Using the following code to configure AWS using. You will need to specify four parameters:

1. AWS Access Key ID
2. AWS Secret Access Key
3. Default region name (this is the default region that will be used when you create EC2 instances)
4. Default output format (the default format is json)

```
aws configure
```

9.2.5 AWS Admin Access

9.2.5.1 Introduction

In order to access various AWS functionality remotely (through command-line) you must enable administrative access.

9.2.5.2 Prerequisites

- [Set up AWS account](#)
- [Install and configure AWS CLI](#)
- [Linux environment](#)
- [AWS Key Pair](#)

9.2.5.3 Setting up admin access using AWS CLI

9.2.5.3.1 Create an admin security group

```
aws iam create-group --group-name Admins
```

9.2.5.3.2 Assign a security policy to the created group granting full admin access

```
aws iam attach-group-policy --group-name Admins --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

9.2.6 Understanding the free tier

AWS provides a set of services free of charge. These free services are to allow new users test and experiment with various AWS services without worrying about any cost. Free services are provided as a product that is free until a certain amount of usage, that is if you exceed those limits you will be charged for the additional usage. However the free quotas are typically more than sufficient for testing and learning purposes. For example under the free tier you are able to use 750 hours of EC2 resources per month for the first 12 months after account creation. However it is important to make note of important details that are included in the limits. For example for the 750 hours of free EC2 usage, you can only use “EC2 Micro” instances, using any other instance type for your EC2 machine will not fall under the free tier agreement and you will be charged for them, see picture: [Figure 42](#). To view all the AWS free tier details visit [AWS Free Tier](#)

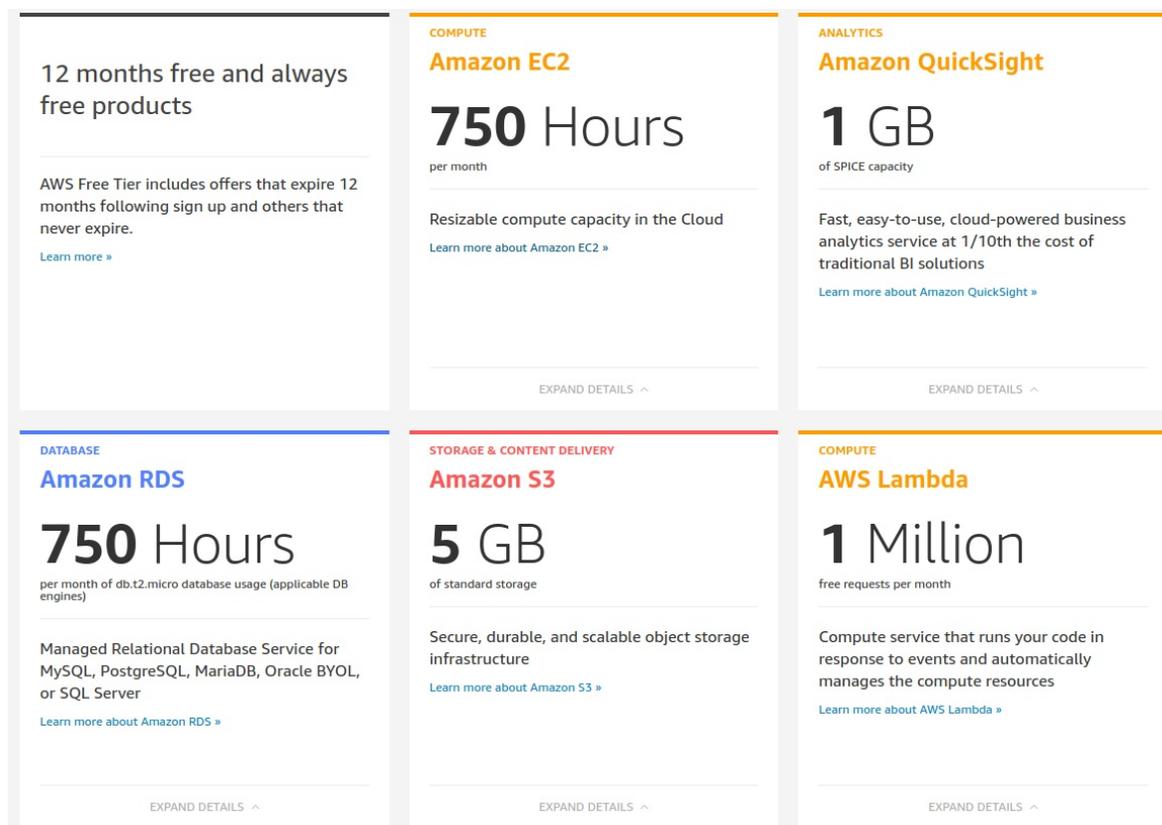


Figure 42: Free tier

Basically there are two categories in the free tier,

- 12 months free
- Always free

12 months free offer are only good for the first 12 months after you create your AWS account. The always free offer are valid even after the first 12 months.

9.2.7 Important Notes

 *When using AWS services make sure you understand how and when you will be charged for.*

For example if you are using an EC2 to run some application, usage of the instance will be calculated from the time you started the instance to the time you stop or terminate the instance. So even if you do not use the application itself, if you are have the instance in an active mode that will be added to the usage hours and you will be billed if you exceed the 750 hour limit. In EC2 even if you stop the instance you might be charged for data that is stored in the instance so terminating it would be the most safest option if you do not have any important data stored in the instance. You can look up other such tricky scenarios at [Avoiding Unexpected Charges](#) to make sure you will not get an unexpected bill

9.2.8 Introduction to the AWS console

As we discussed previously we can access all the service and product offerings that are provided by AWS from the AWS console. In the following section we will look into how we can start and stop a virtual machine using AWS EC2 service. Please keep in mind that this will reduce time from your free tier limit of 750 hours/month, So be careful when starting EC2 instances and make sure to terminate them after you are done.

9.2.8.1 Starting a VM

To go to the EC2 services you can click on the services link on the top left corner in the console and then click on EC2 which is listed under “Compute”. Then you will see a blue button labeled “Launch instance”. Click on the button and the console will take you to the page shown next: [Figure 43](#). Notice that the check box for “Free tier only” is clicked to make sure the instance type we

choose is eligible for the free tier hours. The instance type you select defines the properties of the virtual machine you are starting such as RAM, Storage, processing power. However since we are using instance that are free tier eligible we will only be able to use “EC2 Micro instances”. You can also select a OS that you want to be started as the virtual machine. We will select “Ubuntu Server 16.04 LTS” as our Operating system. press the blue select button to do so.

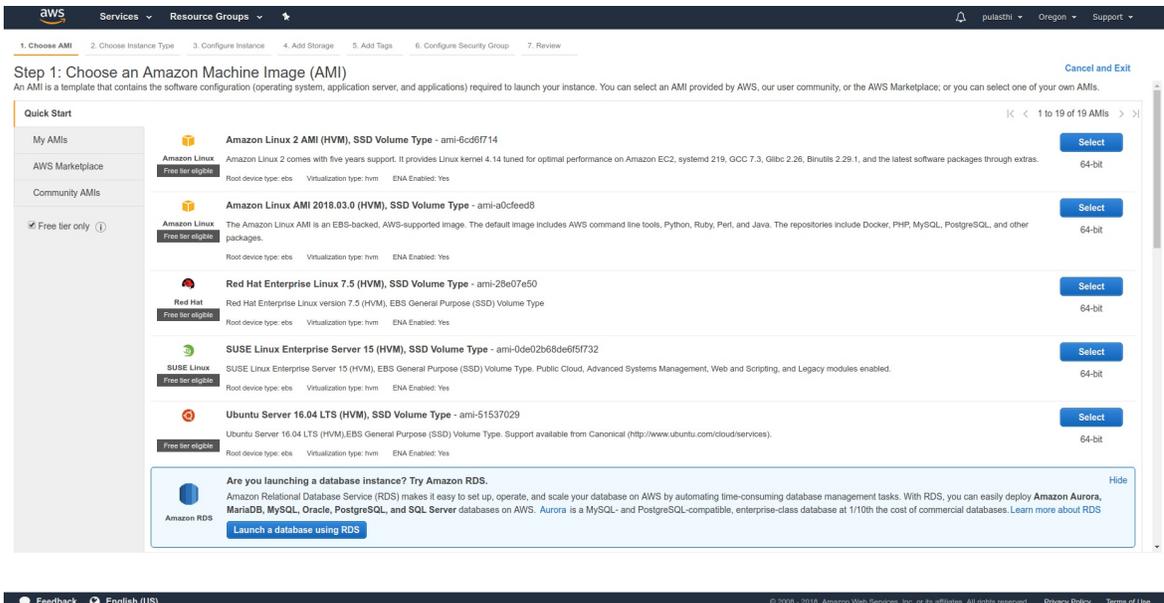


Figure 43: Launch Instance

Once you select the OS type you will be asked to select the instance type. You can notice that only the “t2.micro” is marked as free tier eligible as shown in the image: [Figure 44](#). Now that you have selected all the basic details press the “Review and Launch” button located in the button right corner. This will give you a summary of your current selections.

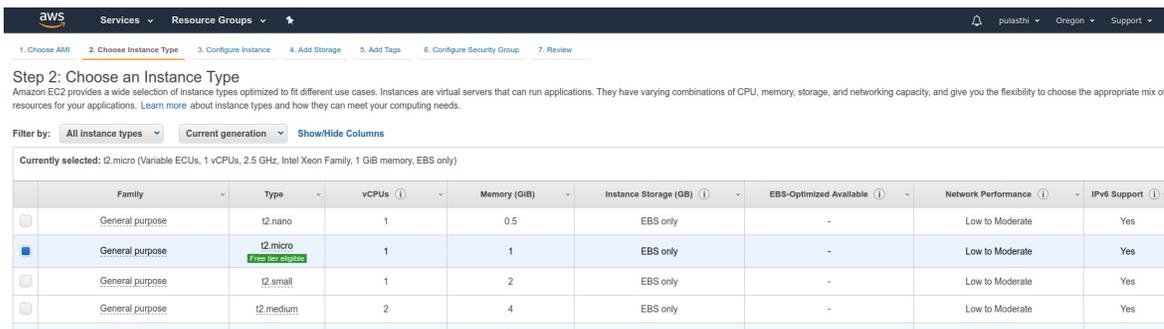


Figure 44: Instance Type

9.2.8.1.1 Setting up key pair

Before we can launch the VM we need to perform one more step. We need to setup a SSH key pair for the new VM. Creating this will allow us to access our VM through SSH. Once you click on the launch button, you will get the following dialog box: [Figure 45](#). If you already have a worked with SSH keys and if you already have a key pair you can use it, otherwise you can create a new

key pair as we will do. To create a new key pair select the “Create a new key pair” in the first drop down box and enter a name of your choosing as the name. Next you need to download and save the private key, Keep the private key in a safe place and do not delete it since you will need it when you are accessing the VM (This tutorial will not cover accessing the VM through SSH but you need to keep the private key so you can use the same key value pair later). Once you have downloaded the private key, the “Launch Instance” button will activate. Press this button to start the VM.

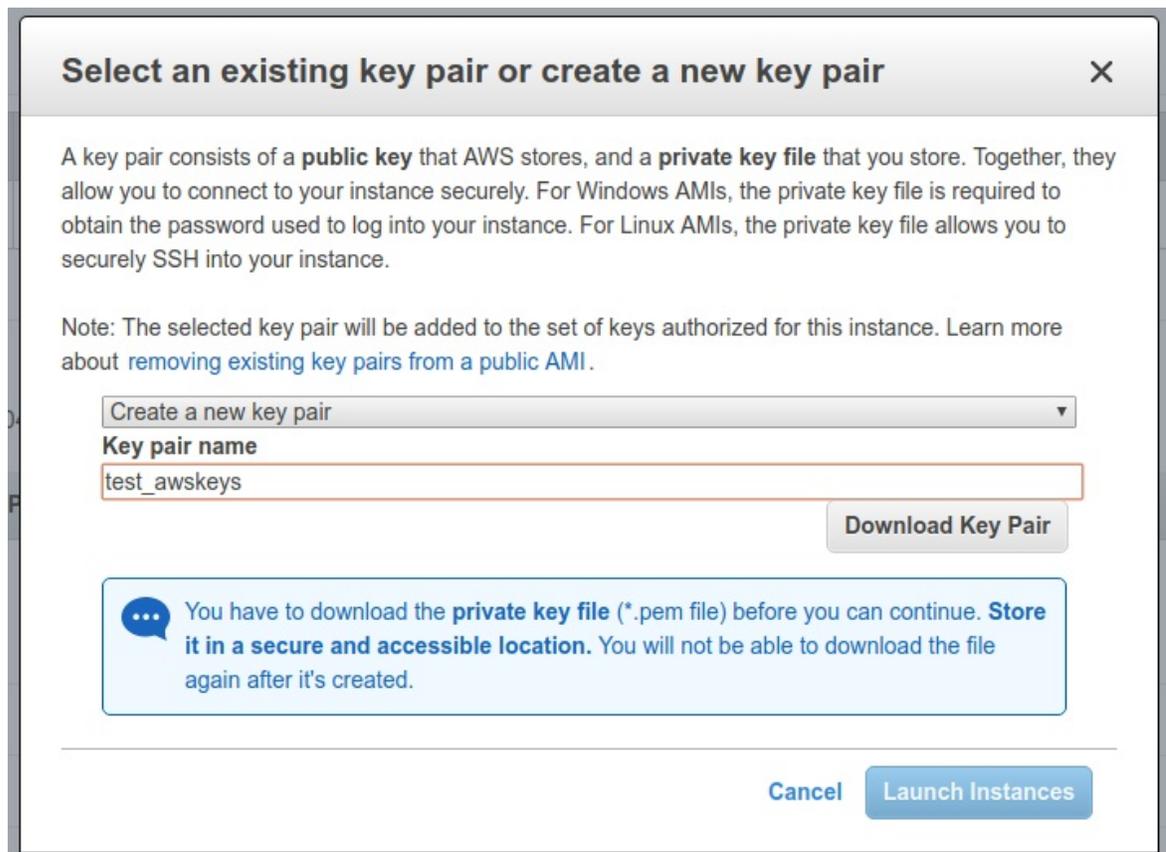


Figure 45: Key Pair

After starting the instance go back to the EC2 dashboard (Services -> EC2). Now the dashboard will show the number of running instance as shown in the image: [Figure 46](#). If you do not see is initially, refresh the page after a little while, starting the VM may take a little time so the dashboard will not be updated until the VM starts.

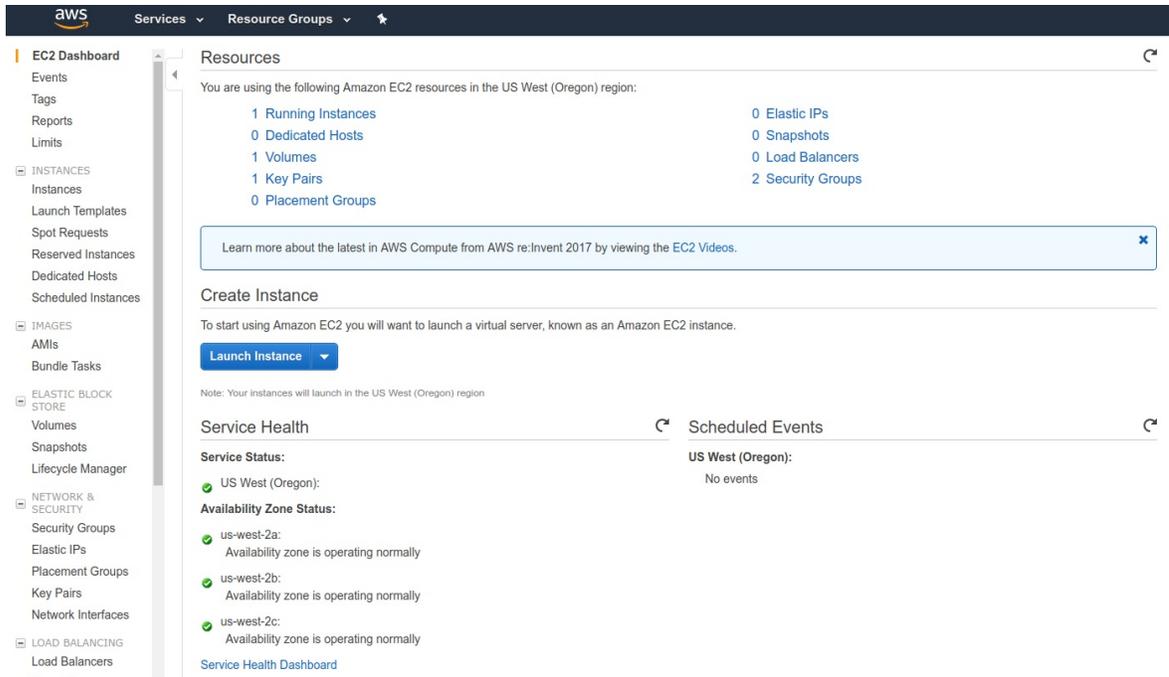


Figure 46: Running Instance1

Now to get a more detailed view click on the “Running Instances” link. This will give you the following view: [Figure 47](#). It shows the current instance that you are running

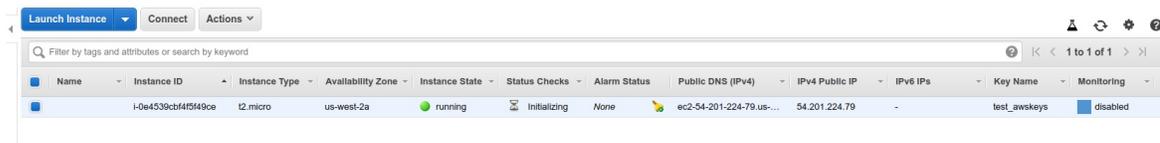


Figure 47: Running Instance2

9.2.8.2 Stopping a VM

In AWS EC2 you can either stop a VM or terminate it. If you terminate it you will lose all the data that was stored in the VM as well, simply stopping will save the data for future use if you restart the instance again. In order to stop the VM you can select the VM machines you want to stop from the GUI and go to “Actions -> Instance status” and click on stop: [Figure 48](#). This will stop your VM machine.

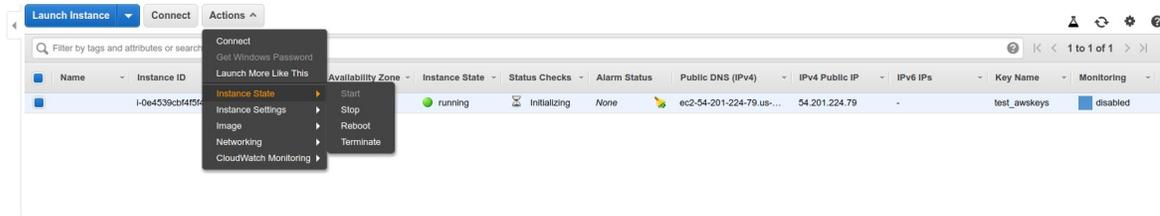


Figure 48: Instance Stop

After a little while the dashboard will show the instance as stopped as the following: [Figure 49](#). If you want to go further and terminate the instance you can again go to “Actions -> Instance status” and select terminate, which will terminate the VM.

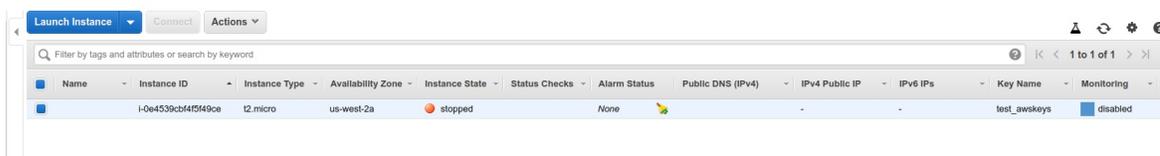


Figure 49: Stopped Instance

9.2.9 Access from the Command Line

AWS also provides a command line interface that can be used to manage all the AWS services through simple commands. Next are two example commands.

```
aws s3 <Command> [<Arg> ...]
aws ec2 <Command> [<Arg> ...]
```

You can find more information regarding the AWS CLI in the following documents.

- AWS Command Line: <https://aws.amazon.com/cli/>
- AWS Command Line reference: <https://docs.aws.amazon.com/cli/latest/reference/>
- EC2: <https://docs.aws.amazon.com/cli/latest/reference/ec2/index.html>
- S3: <https://docs.aws.amazon.com/cli/latest/reference/s3/index.html>

Amazon Web Services (AWS) is a cloud platform that provides a large number of services for individuals and enterprises. You can get an overview of the AWS offering at [Amazon Web Services Overview](#). This section will guide through the processes of creating an AWS account and explain the free tier details so that you can leverage the tools and products available in AWS for your work and

research.

9.2.10 Access from Python

9.2.11 Boto

Boto is a Python software development kit specifically targeting Amazon Web Services (AWS). It allows access to services such as S3 and EC2. It is using object oriented programming paradigms to access the lower level services. The advantage is that it is written just for Amazon and thus we assume it will be developed with high quality due to its specialization. However this is also its limitation as in contrast to libcloud it does not support other cloud providers. Hence it bears the risk of vendor lockin. Boto is maintained in github.

Documentation about boto can be found at

- <https://boto3.readthedocs.io/en/latest/>
- <https://github.com/boto/boto3>

9.2.12 libcloud

“Libcloud is a Python library for interacting with many of the popular cloud service providers using a unified API. It was created to make it easy for developers to build products that work between any of the services that it supports.” A more detailed description on Libcloud and how you can use it to connect with AWS is provided in the Section [Python libcloud](#).

For more information about the features and supported providers, please refer to the [documentation](#)

9.3 MICROSOFT AZURE

Microsoft Azure is a cloud computing service created by Microsoft. It includes computing services and products for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming

languages, tools and frameworks, including both Microsoft-specific and third-party software and systems.

9.3.1 Products

Microsoft offers a large number of services. We included the from Microsoft a number of services in the appendix, with convenient links to them.

The services are organized in the following categories:

- AI + Machine Learning
- Analytics
- Compute
- Containers
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- Management Tools
- Media
- Migration
- Mobile
- Networking
- Security
- Storage
- Web

We focus next on the

- compute,
- container, and
- data resources

For a more elaborate list please consult our Appendix. To see the complete list please visit the Microsoft Web page via this [link](#).

9.3.1.1 Virtual Machine Infrastructure as a Services

Source: <https://support.microsoft.com/en-us/allproducts>

Microsoft offers core IaaS [Compute](#) compute resources. This includes the following services:

- [Virtual Machines](#) to provision Windows and Linux virtual machines
- [Virtual Machine Scale Sets](#) to manage and scale thousands of Linux and Windows virtual machines

9.3.1.2 Container Infrastructure as a Service

Microsoft offers [Containers](#) to allow for the development of containerized applications. This includes:

- [Azure Kubernetes Service \(AKS\)](#) to provide access to Kubernetes as a Service so that deployment, management, and operations of Kubernetes can be conducted on the cloud resources offered.
- [Service Fabric](#) to develop microservices and orchestrate containers on Windows or Linux as part of the infrastructure
- [Container Instances](#) to run containers on Azure without managing servers which seems unrelated to kubernetes
- [Container Registry](#) to store and manage container images for deployments

9.3.1.3 Databases

Storage is offered through a variety of [Database](#) services to provide access to enterprise-grade, and fully managed services.

- [Azure Cosmos DB](#) is a globally distributed, multi-model database for any scale
- [Azure SQL Database](#) is a managed relational SQL database as a service
- [Azure Database for MySQL](#) is a managed MySQL database as a service
- [Azure Database for PostgreSQL](#) is a managed PostgreSQL database service
- [SQL Server on Virtual Machines](#) allowing to host enterprise SQL Server apps in the cloud

- [SQL Data Warehouse](#) is an elastic data warehouse as a service with enterprise-class features
- [Azure Database Migration Service](#) simplifies on-premises database migration to the cloud
- [Redis Cache](#) which provides a Redis Cache as a service to support high-throughput and low-latency data access
- [SQL Server Stretch Database](#) which supports dynamically stretch on-premises SQL server databases to Azure

9.3.1.4 Networking

We will not go much into the network offerings at this time

9.3.2 Registration

In order for you to register in Azure and start your free account, you will need to go to

- <https://azure.microsoft.com/en-us/free/>

You will see an image such as: [Figure 50](#).

Create your Azure free account today

Get started with 12 months of free services

Start free >

Or buy now >

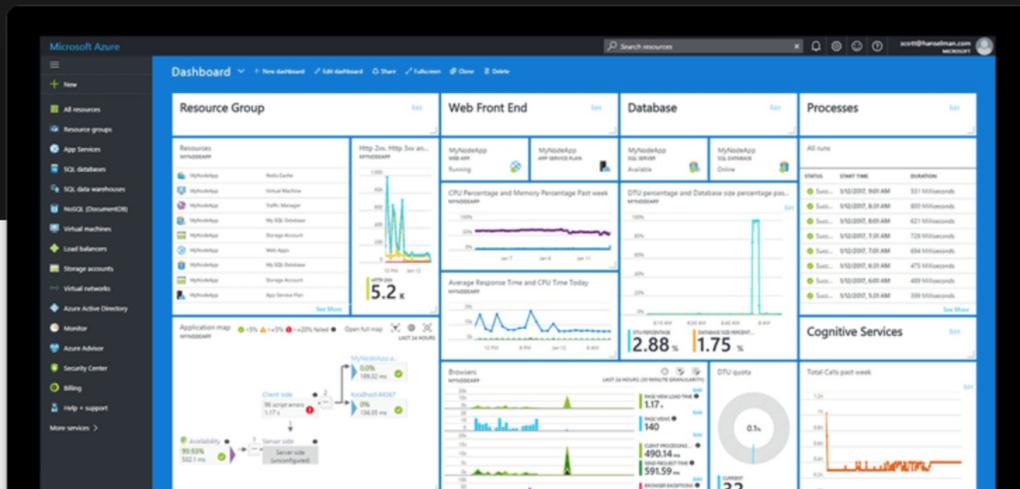


Figure 50: Registration

On that image you click the `start free` button to obtain a free one year account. You will have to either create a new Microsoft account or use the one from Indiana University which will be your IU id followed by the `???` domain. You will be redirected to the single sign on from IU to proceed. If you use another e-mail you can certainly do that and your free account is not associated with the IU account. This could be your Skype account or some other e-mail. After registration you will be provided with 12 months of free usage of a few selected services and \$200 credits for 30 days. At the end of 30 days, Azure disables your subscription. Your subscription is disabled to protect you from accidentally incurring charges for usage beyond the credit and free services included with your subscription. To continue using Azure services, you must upgrade your subscription to a Pay-As-You-Go subscription. After you upgrade, your subscription still has access to free services for 12 months. You only get charged for usage beyond the free services and quantities. The Azure Student Account requires that you to activate the account after 30 days of use. If you do not activate, you will lose access to your Azure Student Account and can not use the

services.

The Azure student account FAQ will likely answer questions you might have pertaining to an Azure Student Account, what you will have access to, how long you will enjoy access, and additional general overview information including terms of the account. <https://azure.microsoft.com/en-us/free/free-account-students-faq>

Once you have set up the Azure Student account, you will gain access the Azure environment through the Azure Portal <https://portal.azure.com>. To log in, please use the credentials you determined during the set up.

The services that you have access to include:

- Linux Virtual Machines (750 Hours)
- Windows Virtual Machines (750 Hours)
- Managed Disks (64 GB X 2)
- Blob Storage (5 GB)
- File Storage (5 GB)
- SQL Database (250 GB)
- Azure Cosmos DB (5 GB)
- Bandwidth (Data Transfer 15 GB)
- In case Azure changes the product list, please refer to the official page for a full list of free products: <https://azure.microsoft.com/en-us/free/>

9.3.3 Introduction to the Azure Portal

Azure can be accessed via a portal. An introductory video from Microsoft provides you with some elementary information:



[Introduction to Azure Portal](#)

9.3.4 Creating a VM

Choose `create a resource` in the upper left-hand corner of the Azure portal. Select a VM name, and the disk type as SSD, then provide a username. The password must be at least 12 characters long and meet the defined complexity

requirements. As the following: [Figure 51](#).

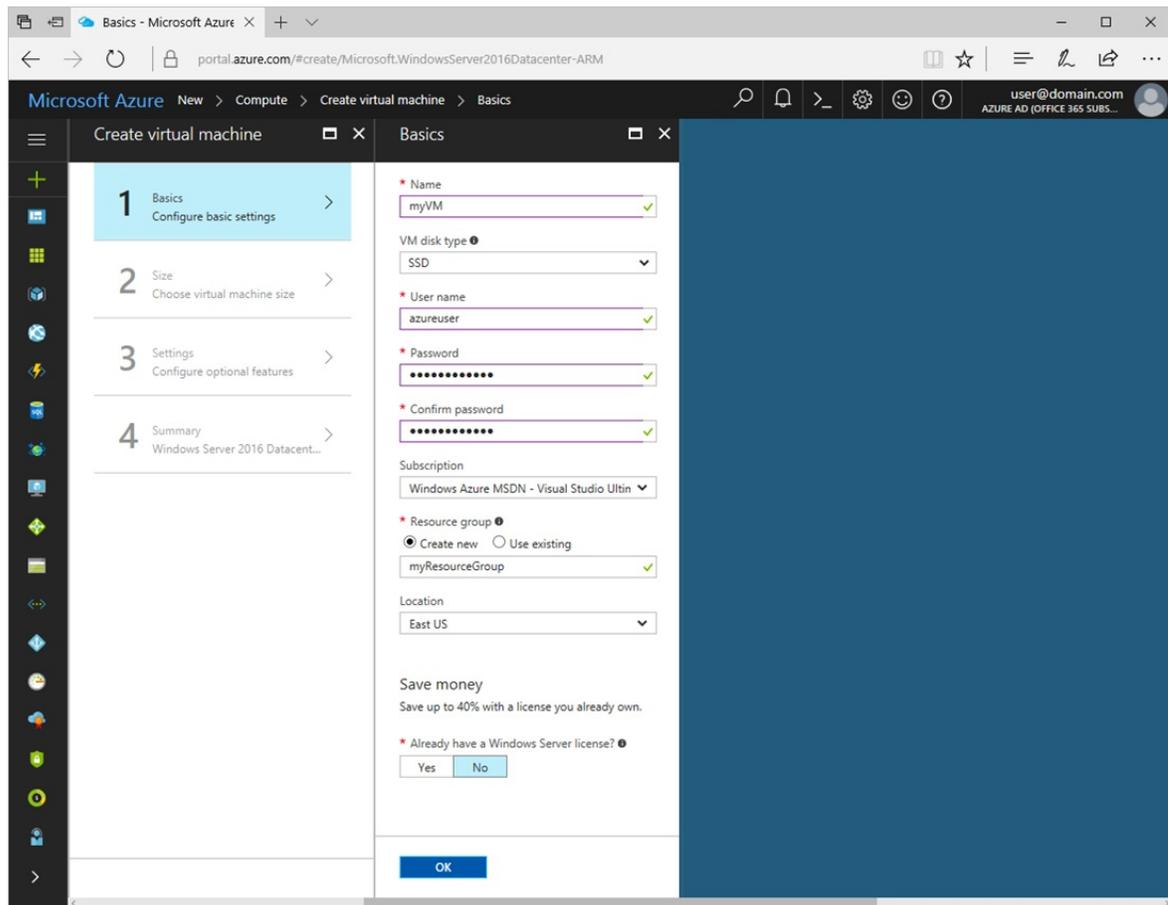


Figure 51: Creating a VM

Source: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/media/quick-create-portal/create-windows-vm-portal-basic-blade.png>

9.3.5 Create a Ubuntu Server 18.04 LTS Virtual Machine in Azure

Here are the steps to create a Ubuntu Server 18.04 LTS Virtual Machine in Azure.

To start, go to the Azure Portal <https://portal.azure.com>.

Next, locate the `virtual machines` option and select it:(see [Figure 52](#)).

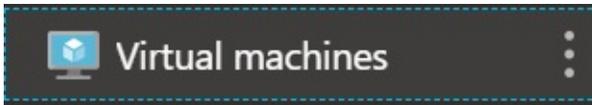


Figure 52: virtualmachines

Then to create a new virtual machine, select **Add**:(see [Figure 53](#)).

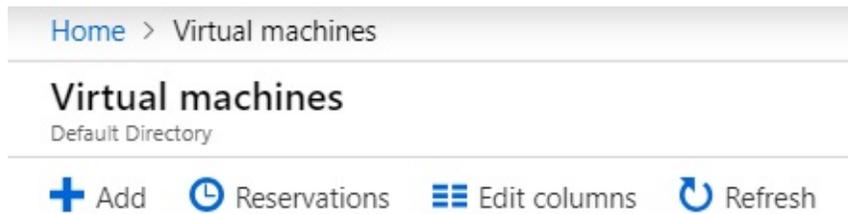


Figure 53: addvirtualmachines

This will present you the configuration options needed to create a new virtual machine:(see [Figure 54](#)).

Create a virtual machine

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription ⓘ ✓

* Resource group ⓘ [Create new](#)

INSTANCE DETAILS

* Virtual machine name ⓘ ✓

* Region ⓘ ✓

Availability options ⓘ ✓

* Image ⓘ ✓
[Browse all images](#)

* Size ⓘ **Standard B1ls**
1 vcpu, 0.5 GB memory
[Change size](#)

ADMINISTRATOR ACCOUNT

Authentication type ⓘ Password SSH public key

* Username ⓘ ✓

* Password ⓘ ✓

* Confirm password ⓘ ✓

Login with Azure Active Directory (Preview) On Off ⓘ

INBOUND PORT RULES

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

* Public inbound ports ⓘ None Allow selected ports

* Select inbound ports ✓

⚠ These ports will be exposed to the internet. Use the Advanced controls to limit inbound traffic to known IP addresses. You can also update inbound traffic rules later.

[Review + create](#) [Previous](#) [Next : Disks >](#)

Figure 54: createavirtualmachine

To configure the virtual machine, choose the following options or modify to

your situational needs.

Subscription: `Azure for Students` (default)

Resource Group: `YourResourceGroupHere` (Create a new one if you do not have an available option.)

Virtual Machine Name: `EfiveothreeTest`

Region: `central us` (default)

Availability Options: `No infrastructure redundancy required` (default)

Image: `Ubuntu Server 18.04 LTS`

Size: `Standard D2s v3, 2 vcpus, 8GB memory`

Authentication type: `password` (Choose a username and a password that meet the requirements).

The next configuration section is `disks`:(see [Figure 55](#)).

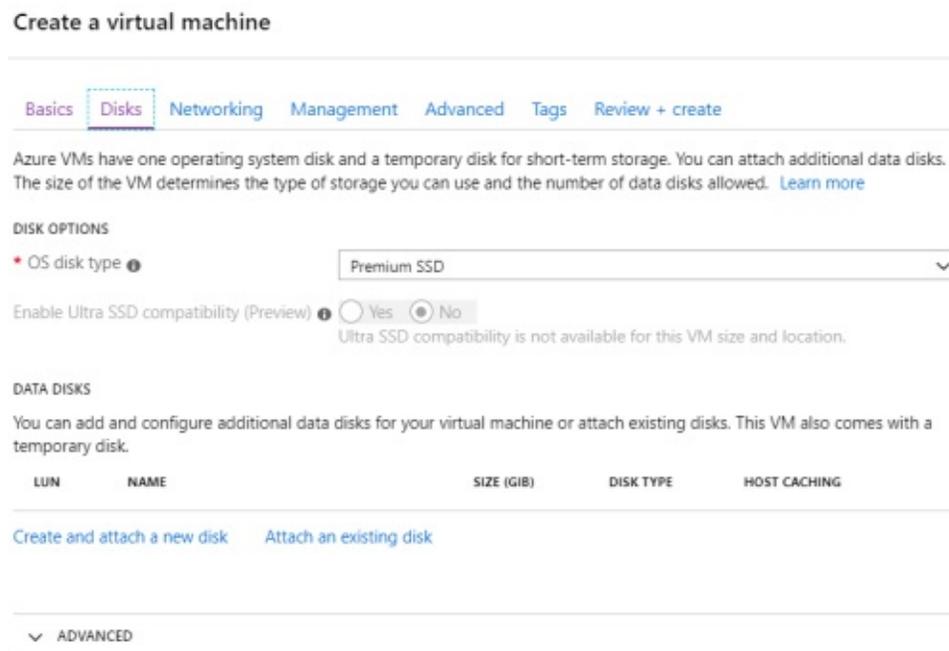


Figure 55: disks

Choose the default configurations settings or modify to your liking the `os disk` type. This example uses the `Standard SSD` option.

For `networking` you can choose all the default configuration settings or modify to your liking:(see [Figure 56](#)).

Create a virtual machine

[Basics](#) [Disks](#) [Networking](#) [Management](#) [Advanced](#) [Tags](#) [Review + create](#)

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution. [Learn more](#)

NETWORK INTERFACE

When creating a virtual machine, a network interface will be created for you.

CONFIGURE VIRTUAL NETWORKS

* Virtual network  [Create new](#)

* Subnet  [Manage subnet configuration](#)

Public IP  [Create new](#)

NIC network security group  None Basic Advanced

* Public inbound ports  None Allow selected ports

* Select inbound ports [Create new](#)

 These ports will be exposed to the internet. Use the Advanced controls to limit inbound traffic to known IP addresses. You can also update inbound traffic rules later.

Accelerated networking  On Off The selected VM size does not support accelerated networking.

LOAD BALANCING

You can place this virtual machine in the backend pool of an existing Azure load balancing solution. [Learn more](#)

Place this virtual machine behind an existing load balancing solution? Yes No

Figure 56: networking

For `management` you can choose all the default configuration settings or modify to your liking:(see [Figure 57](#)).

Create a virtual machine

Basics Disks Networking **Management** Advanced Tags Review + create

Configure monitoring and management options for your VM.

AZURE SECURITY CENTER

Azure Security Center provides unified security management and advanced threat protection across hybrid cloud workloads. [Learn more](#)

✔ Your subscription is protected by Azure Security Center basic plan.

MONITORING

Boot diagnostics ⓘ On Off

OS guest diagnostics ⓘ On Off

* Diagnostics storage account ⓘ [Create new](#)

IDENTITY

System assigned managed identity ⓘ On Off

AUTO-SHUTDOWN

Enable auto-shutdown ⓘ On Off

BACKUP

Enable backup ⓘ On Off

Figure 57: management

Last, create the virtual machine:(see [Figure 58](#)).

Create a virtual machine

✓ Validation passed

Basics Disks Networking Management Advanced Tags Review + create

BASICS

Subscription: Azure for Students
Resource group: [REDACTED]
Virtual machine name: EfiveothreeTest
Region: Central US
Availability options: No infrastructure redundancy required
Authentication type: Password
Username: [REDACTED]
Public inbound ports: HTTP, HTTPS, SSH, RDP

DISKS

OS disk type: Premium SSD
Use managed disks: Yes

NETWORKING

Virtual network: AndrewCloudServicetest-vnet
Subnet: default (10 [REDACTED])
Public IP: (new) [REDACTED]
Accelerated networking: Off
Place this virtual machine behind an existing load balancing solution?: No

MANAGEMENT

Boot diagnostics: On
OS guest diagnostics: Off
Azure Security Center: Basic (free)
Diagnostics storage account: [REDACTED]
System assigned managed identity: Off
Auto-shutdown: Off
Backup: Disabled

ADVANCED

Extensions: None
Cloud init: No

Figure 58: createvmvalidation

Once the new VM has been created, Naviagate back to the `virtual machines` and now discover your Virtual Machine:(see [Figure 59](#)).

NAME	TYPE	STATUS	RESOURCE GROUP	LOCATION	MAINTENANCE STATUS	SUBSCRIPTION
EfiveothreeTest	Virtual machine	Running	[REDACTED]	Central US	-	Azure for Students

Figure 59: newvmaftercreation

After creation the virtual machine will be in a `running` status. You will want to decide if you want your virtual machine in a `running` status, else stop the VM so

that you do not waste resources.

9.3.6 Remote access the Virtual Machine

To remote access a virtual machine, you can use a client application like Putty: <https://www.putty.org> .

To use Putty and access the virtual machine, you can configure DNS name in Azure instead of using an IP. This is performed in the Virtual Machine configuration under `DNS name`:(see [Figure 60](#)).

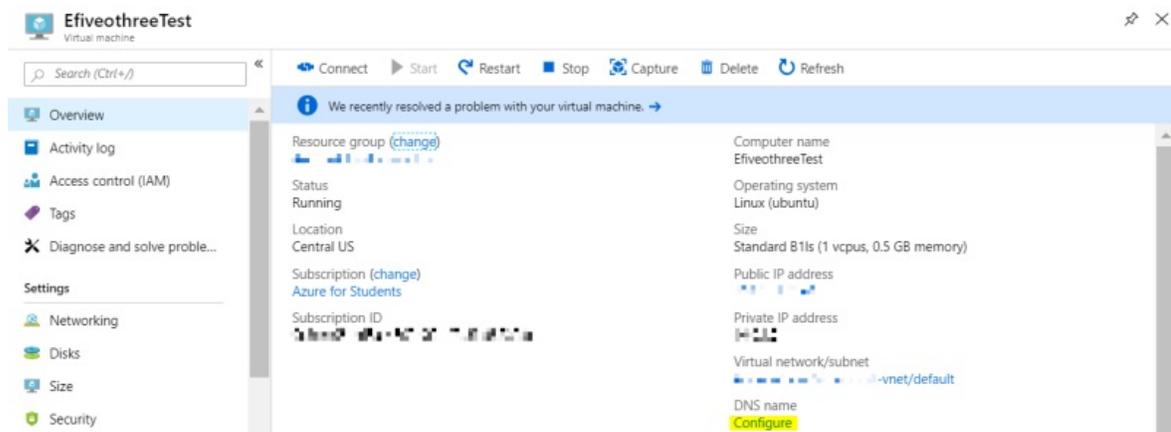


Figure 60: dns

Click `Configure`. You can choose a `static` IP setting or a `dynamic` IP (This example uses a static IP setting):(see [Figure 61](#)).



Figure 61: static

To apply the setting, click `save`.

Note: If you have not configured the `port` that connection will use, then connection will not be successful.

In your Virtual machine settings click `connect` and review the connection settings. This example shows the designated `port 22` to be the port that will remote connect to the virtual machine:(see [Figure 62](#)).

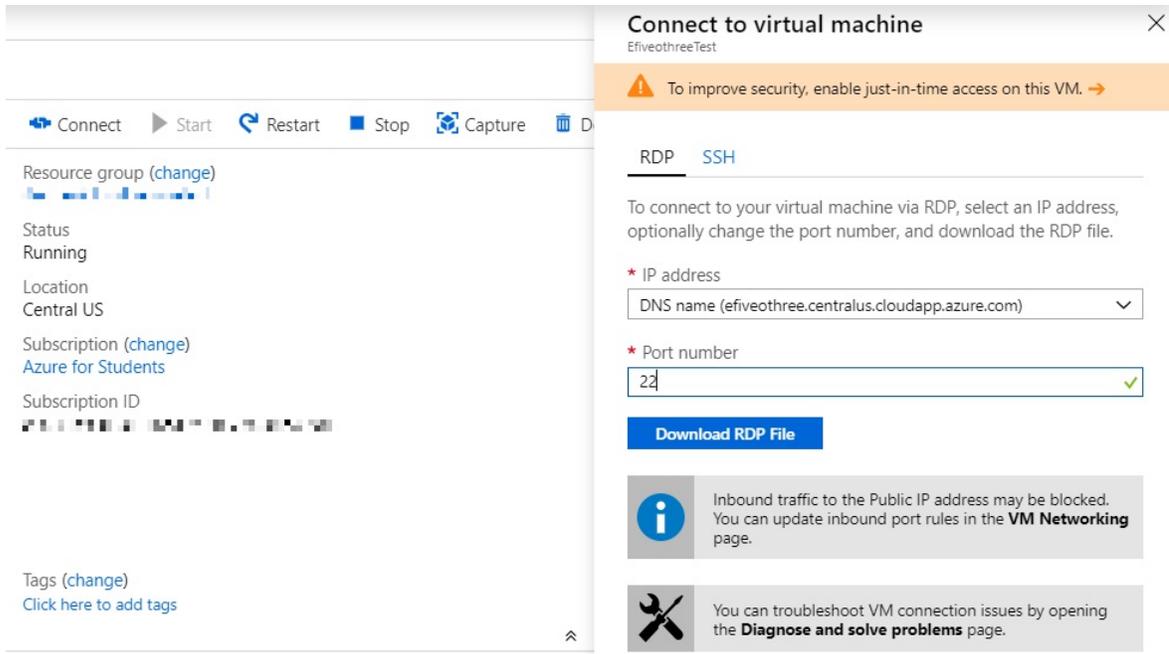


Figure 62: connectandport

To learn more about working with ports you can review the following: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/nsg-quickstart-portal>

Next, to connect to the virtual machine, launch the Putty client and enter the DNS name of the virtual machine to connect to the virtual machine:(see [Figure 63](#)).

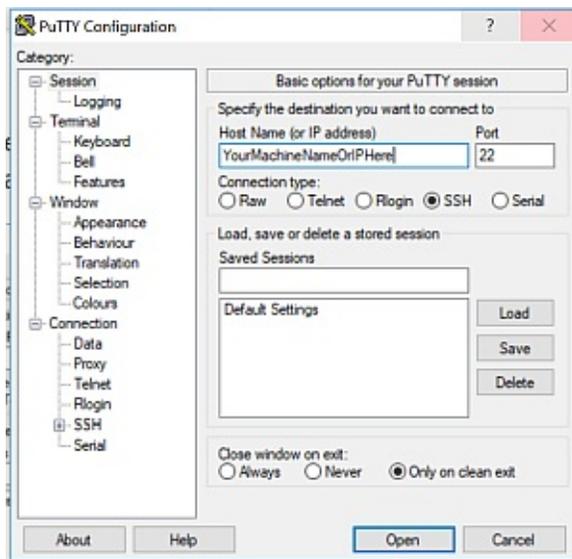


Figure 63: putty

The first time the environment is accessed Putty, Putty will prompt to cache your servers host key. Select `yes` when prompted:(see [Figure 64](#)).

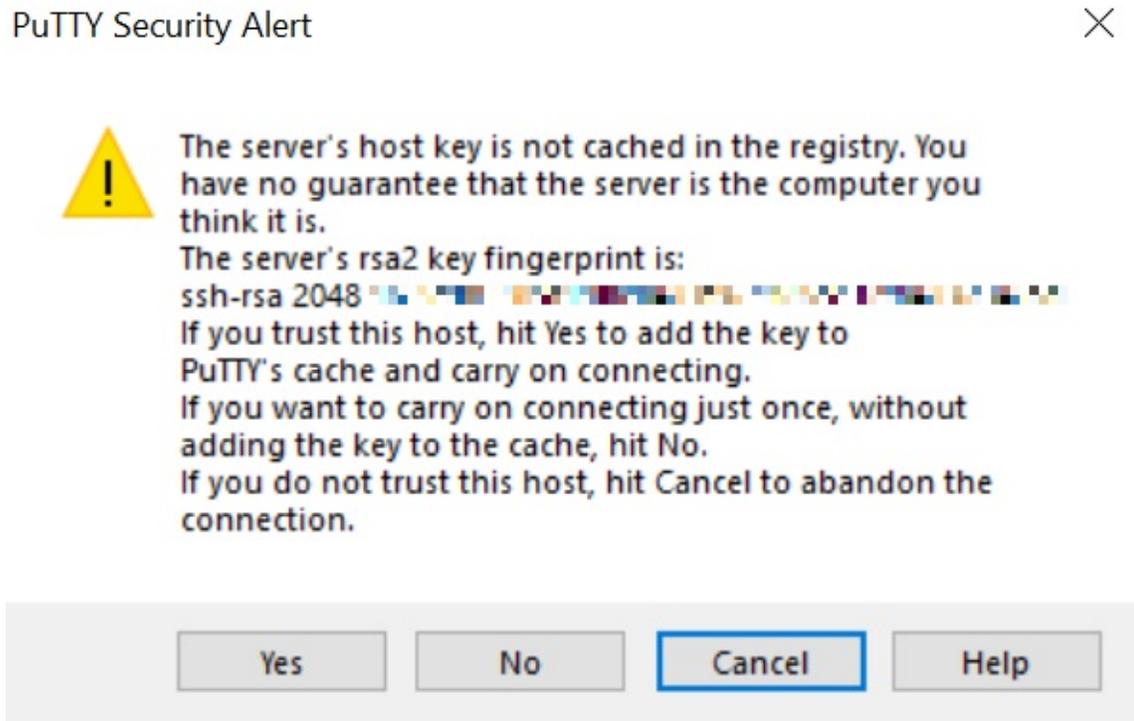
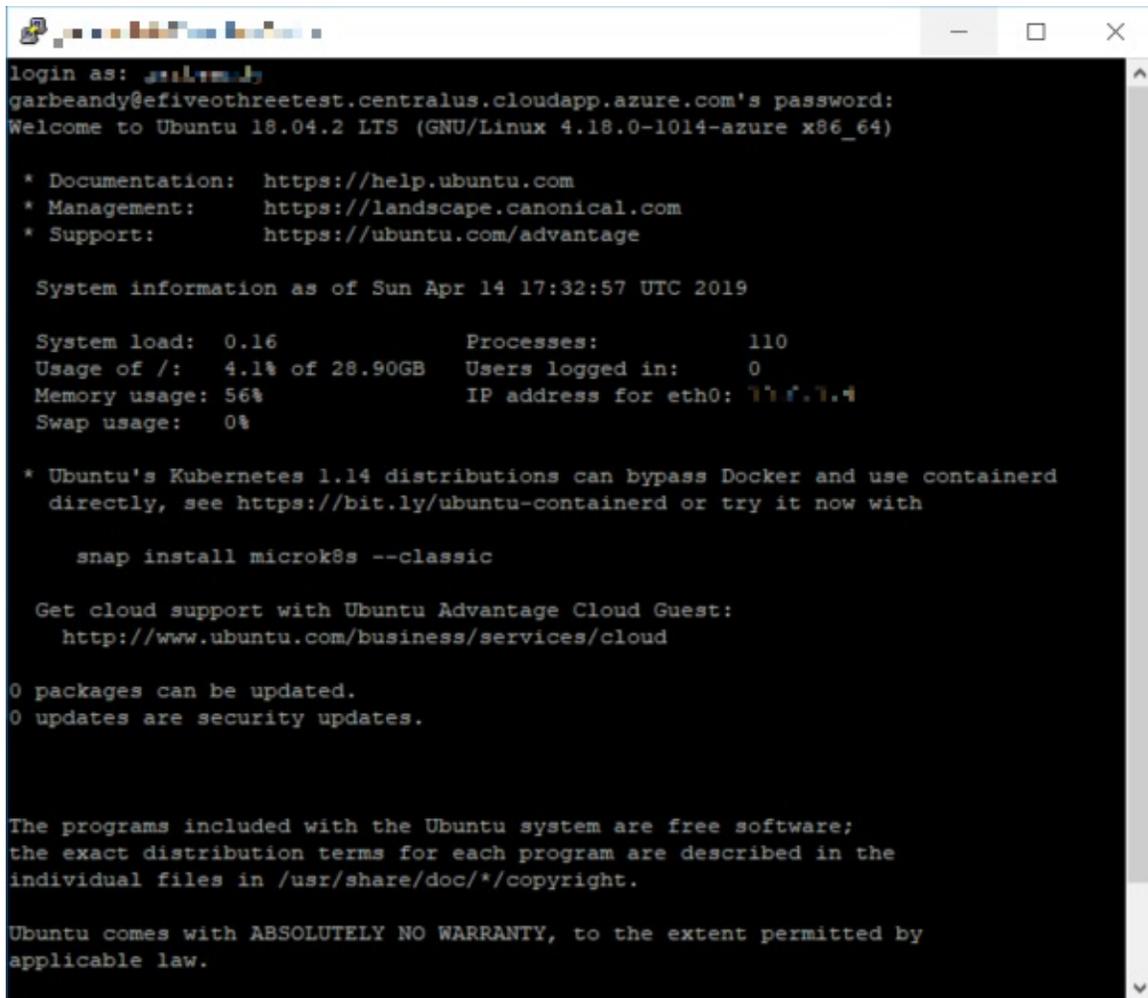


Figure 64: cacheputtykey

After the key is cached, it will be remember the next time you access the VM with the Putty client. After the VM is successfully accessed in Puty, you will be prompted to enter your server credentials as specified in the virtual machine setup.

Once credentials are provided, you will be logged into your virtual machine:(see [Figure 65](#)).

A terminal window showing the login process for an Ubuntu 18.04.2 LTS virtual machine. The user 'garbeandy' has successfully logged in. The terminal displays system information as of Sun Apr 14 17:32:57 UTC 2019, including system load (0.16), memory usage (56%), and IP address (10.0.0.4). It also provides links for documentation, management, and support, and offers instructions for installing microk8s via snap. The terminal concludes with a notice about Ubuntu's warranty and copyright information.

```
login as: garbeandy
garbeandy@efiveothreetest.centralus.cloudapp.azure.com's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-1014-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun Apr 14 17:32:57 UTC 2019

System load:  0.16          Processes:    110
Usage of /:   4.1% of 28.90GB Users logged in:  0
Memory usage: 56%          IP address for eth0: 10.0.0.4
Swap usage:   0%

 * Ubuntu's Kubernetes 1.14 distributions can bypass Docker and use containerd
   directly, see https://bit.ly/ubuntu-containerd or try it now with

   snap install microk8s --classic

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

Figure 65: loggedinviaputty

To learn more about connecting to Azure virtual machines you can visit: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/connect-logon>

9.3.7 Starting a VM

Now we like to introduce you how to start a VM. Please note that VMS do cost and reduce your free hours on Azure. Hence you need to make sure you carefully review the charging rates and chose VM sizes and types that minimize your charges.

A VM can be started through the Portal as follows: [Figure 66](#).

- On the overview tab, a VM can be started by clicking the `start` button.

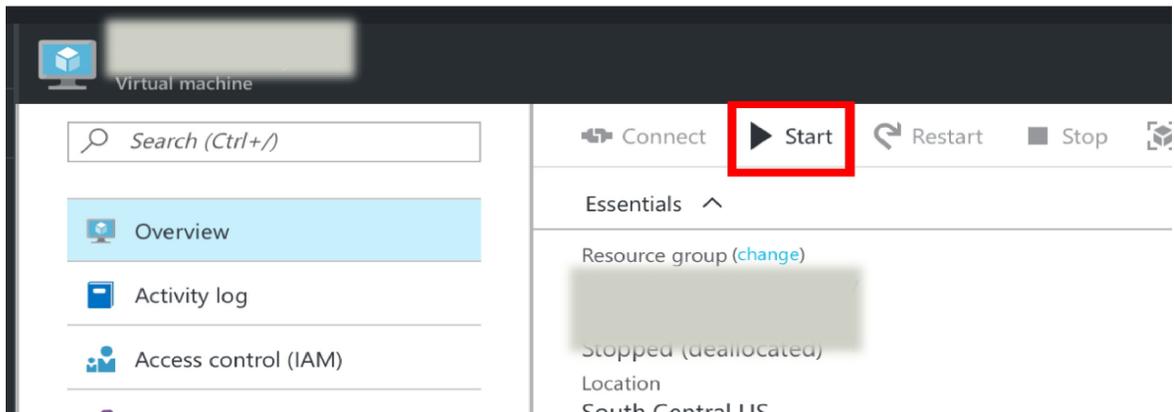


Figure 66: Start button

9.3.8 Stopping the VM

It is the most important to stop your VMS once they are not in used, or you get continuously charged. The portal allows you to see the list of VM that you run as follows

To shut a VM down, please do the following: see [Figure 67](#).

- On the overview tab, a VM can be started by clicking the `stop` button.

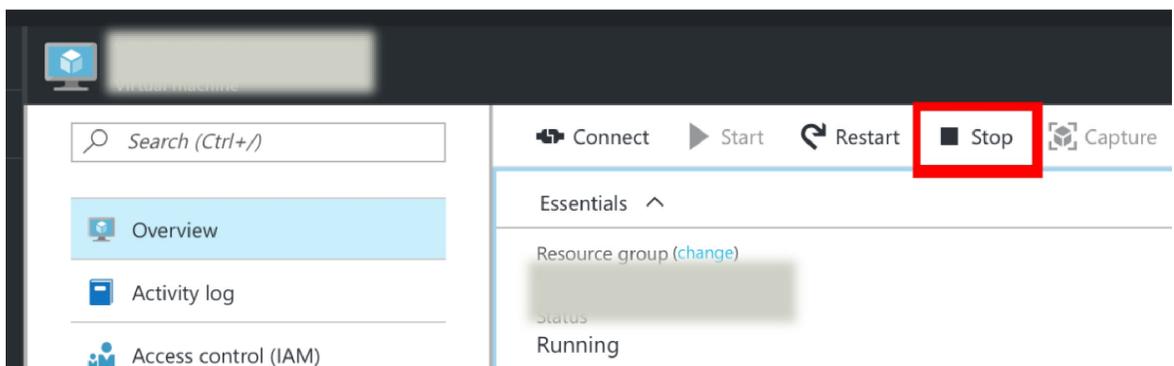


Figure 67: Stop button

9.3.9 Exercises

E.Azure.0:

Identify all products related to IaaS service offerings and mark them with `after` after the bullet. Do copy the `aws.md` file into your own repository, do not yet create a pull request. Confirm in a team your findings and agree with each other.

E.Azure.1:

What is the difference between terminating, shutting down, and suspension?

E.Azure.2:

Do I get charged when the VM is suspended, terminated, shutdown?

E.Azure.3:

How do I resume a VM if it is suspended?

9.4 WHAT IS IBM WATSON AND WHY IS IT IMPORTANT?

In years past we traditionally typed our questions into a query based search engine and it would return relevant content. As we start interacting with our devices more in conversation through natural language processing, we need an answer to a question - not a list of relevant information. We need the power of question answering (QA) technology.

IBM's Watson's is well known for its ability to play and successfully win the popular gameshow, jeopardy! IBM startled the world in 2011 when Watson beat Jeopardy! pros Ken Jennings and Brad Rutter over several rounds. Watson completed the formidable task by combining 15 terabytes of human knowledge with a variety of computer disciplines including automated reasoning, natural language processing, knowledge representation, information retrieval and machine learning.

IBM's goal of having computers understand the questions humans ask while providing answers in a similar fashion is not unique to them. In recent years, products like Amazon's Alexa and Google Home have brought the awareness of this capability mainstream for millions of households. In short, it has become a

race to serve up relevant content with the least amount of effort in the most consumable format.

9.4.1 How can we use Watson?

IBM's Watson has a rich set of Developer Services (<https://www.ibm.com/watson/developer/>) that allow users to stand on the shoulders of the IBM developers using their AI framework to “bolt on” new or improved applications that sit on top.

There are a breadth of services available. Watson Discovery is used to mine through data to find trends and surface patterns. Watson Visual Recognition is used to classify content using machine learning. Watson Assistant provides a framework for chatbots and virtual agents.

While The next section walks through how to create a free account, let's continue with an example of leveraging a foundation and building on-top with Watson Assistant Basic. Please see steps: [Figure 68](#), [Figure 69](#), [Figure 70](#), [Figure 71](#), [Figure 72](#), [Figure 73](#)

Instead of starting with a blank page IBM steps are put in place and working examples can be customized.

Watson Assistant Basic

Web App • Lite

Overview

This starter kit is a simple application to demonstrate the Watson Assistant (formerly Conversation) service in a chat interface simulating a car dashboard. After creating the project, use the getting started material to launch the Tool and learn about how to use Watson Assistant (formerly Conversation). Included is a zip file you can download with everything you need to easily deploy the project locally and to the IBM Cloud.

This starter kit will help you

- Available in Node.JS, Python and Java
- Deploy Locally and to the IBM Cloud
- Access to Watson Assistant (formerly Conversation)

Ready to get started?

Use this starter kit to accelerate your development process.

[Create app](#) [View Demo](#)

Figure 68: Step 1

The screenshot shows the IBM Cloud console interface for the 'Watson Assistant Basic ASKOI' application. The top navigation bar includes 'IBM Cloud', 'Catalog', 'Docs', 'Support', and 'Manage'. The main content area is divided into several sections:

- Resources (1):** A table listing the 'Watson Assistant (formerly Conversation)' resource. It includes links for 'Documentation' and 'API reference', and a 'Launch tool' button.
- Deploy your App:** A section with a 'Deploy to Cloud' button and a note: 'Continuous delivery is not enabled for this app. Enable continuous delivery to automate builds, tests, and deployments through the Delivery Pipeline, GitLab, and more.'
- Credentials:** A section with a 'Show' button and a code block containing the Watson Assistant configuration:


```

{
  "conversation": [
    {
      "name": "watson-assistant-bas-
conversation-1648649829876",
      "credentials": {
        "apikey": "
.....",
        "url": "https://gateway-
wdc-watsonolaefom.net/assistant/api"
      }
    }
  ]
}
      
```
- Knowledge Guide:** A section titled 'Getting started quickly' with a list of steps:
 1. Click "Download Code" to download a .zip file of your generated app
 2. Extract the .zip file
 3. Install Node.js (LTS supported versions).
 4. Open a terminal and go to your starter kit directory
 5. Install the dependencies
npm install
 6. Start the application:
npm start
 7. Go to `localhost:3000` in a web browser to view the application
 An 'Optional' section follows with:
 1. Click on "Launch tool" to train the Assistant service

Figure 69: Step 2

The screenshot shows the 'Introduction' page for IBM Watson Assistant. At the top, there are navigation links for 'Home' and 'Workspaces'. The main heading is 'Introducing IBM Watson Assistant', with a subtext: 'Watson Conversation is evolving to simplify how you build and scale virtual assistants. See what's new'. Below this, a section titled 'Three easy steps' provides a guide:

- 1 Create intents and entities**
Determine what your virtual assistant will understand by providing training examples so Watson can learn.
[Learn more](#)
- 2 Build your dialog**
Utilize the intents and entities you created, plus context from the application, so your virtual assistant responds appropriately.
[Learn more](#)
- 3 Test your dialog**
Try out the virtual assistant in the tool to see how it recognizes the intents and entities and how it responds firsthand.
[Learn more](#)

At the bottom, a blue banner contains the text 'Get started now' and a button labeled 'Create a Workspace'.

Figure 70: Step 3

The screenshot shows the 'Workspaces' page in the IBM Watson Assistant interface. The top navigation bar includes 'Home' and 'Workspaces'. An information banner at the top right states: 'Info: Creating workspace from sample "Customer Service - Sample"'. The main content area is titled 'Workspaces' and features a dashed box highlighting the 'Create a new workspace' section. This section explains that workspaces allow for separate intents, user examples, entities, and dialogs for different uses, and notes that the user is using 1 of 5 available workspaces. A 'Create' button with a plus sign is visible. To the right, a card for the 'Customer Service - Sample' workspace is shown, including a 'Get started' button and a 'Last modified: just now' timestamp.

Figure 71: Step 4

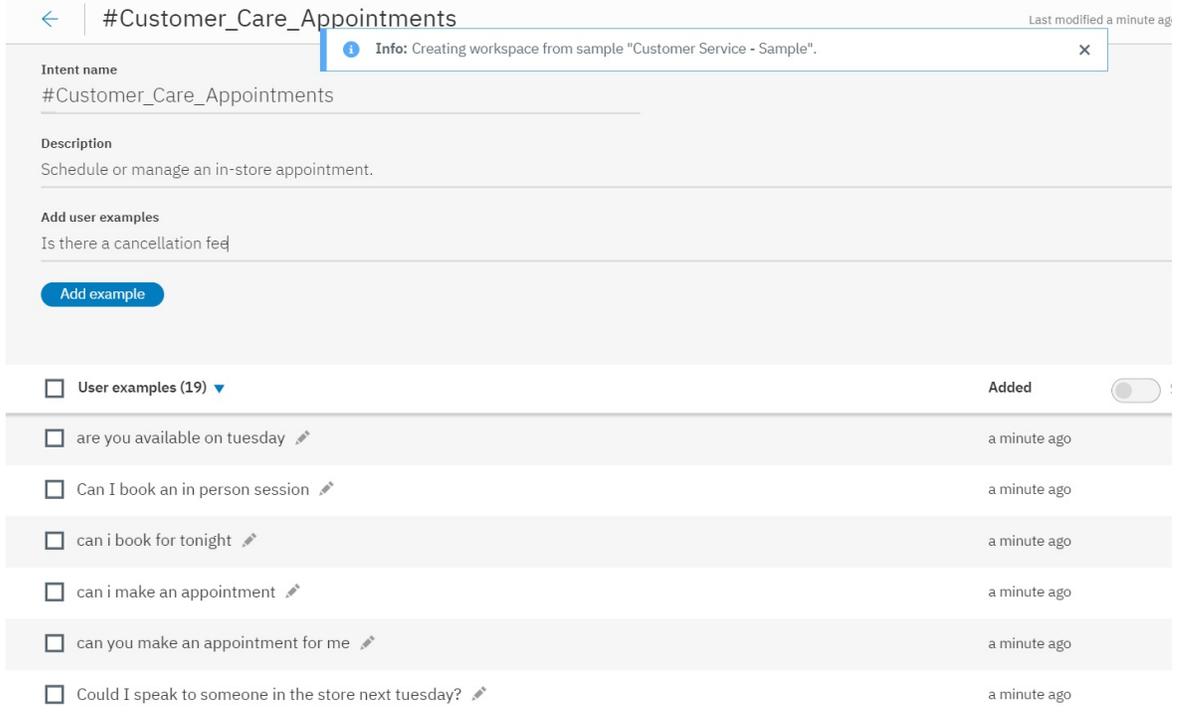


Figure 72: Step 5

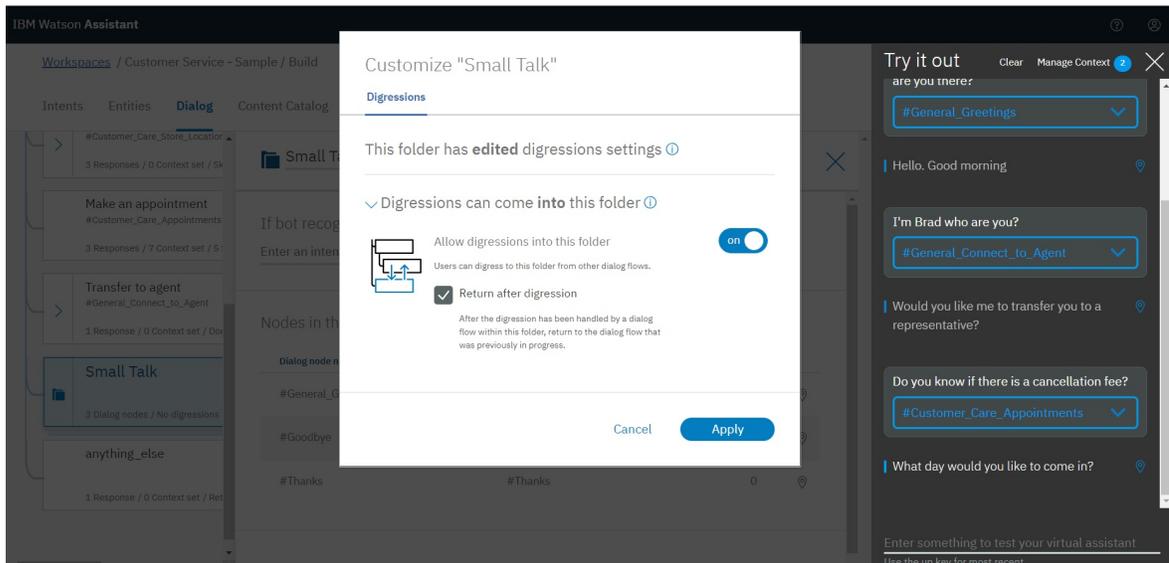


Figure 73: Step 6

In the previous case when I was trying the Watson Assistant It was not personal when I asked the Assistant's name so it can be modified to digress!

In addition to using these modules to help build there is also a variety of APIs and services that can be used:

The list of APIs and services include: * Watson Assistant * Watson Discovery * Natural Language Understanding * Discovery News * Knowledge Studio * Language Translator * Natural Language Classifier * Personality Insights * Tone Analyzer * Visual Recognition * Speech to Text * Text to Speech

9.4.2 Creating an account

This section will guide through the processes of creating an IBM Watson account and explain the free tier details so that you can leverage the tools and products available in AWS for your work and research.

- A valid email address

First you need to visit the [IBM Watson home page](#) and click in the “Get Started Free” link on the top right corner. You will then be asked to provide some basic details including your email address as shown in the image [Figure 74](#).

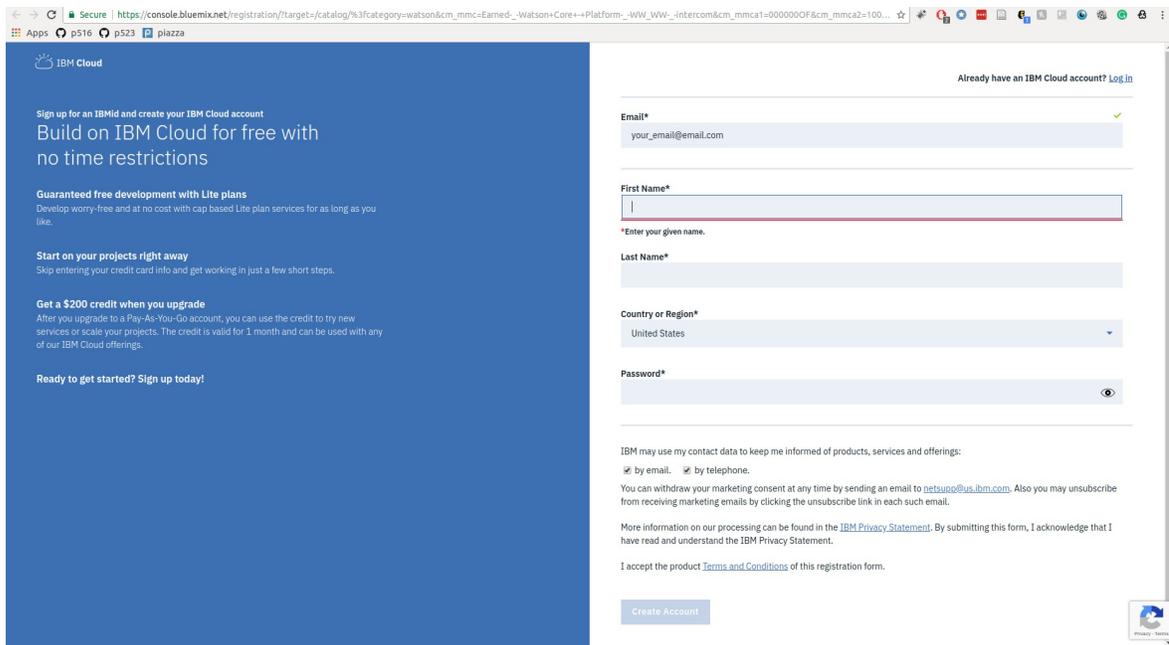


Figure 74: Watson Signup

Once you have submitted the signup form a confirmation email will be sent to your email account, check your inbox and click on the confirm account link in the email you receive. This will activate your IBM Watson account. Once you have accepted the terms and conditions you will be taken to the product and service catalog of IBM Watson as shown in the image [Figure 75](#).

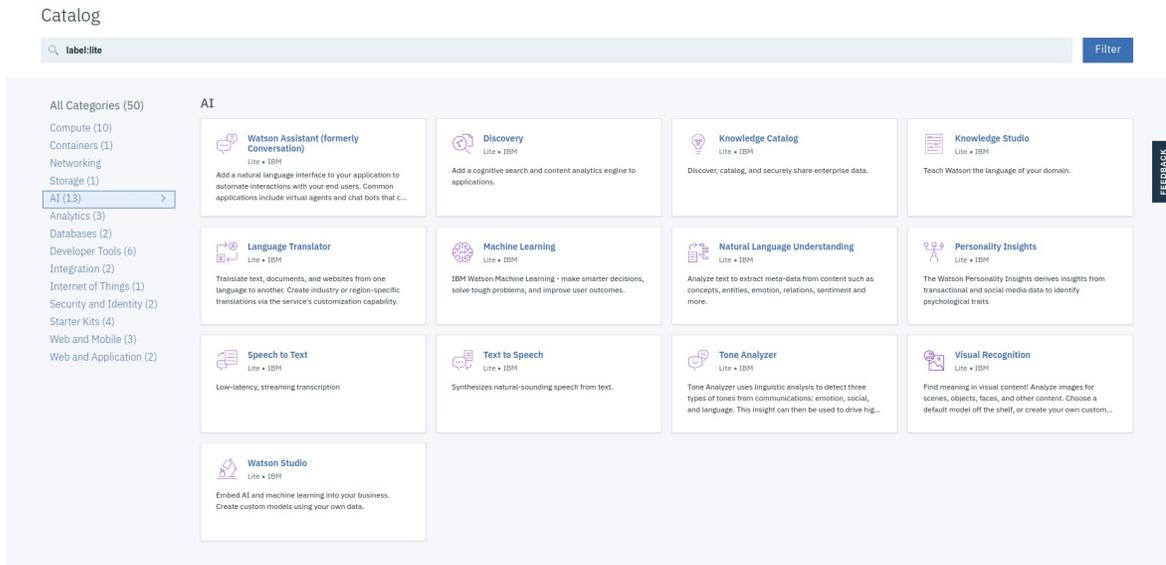


Figure 75: Watson Catalog [Source](#)

9.4.3 Understanding the free tier

IBM Watson provides a set of services for free with their Lite account. Since you did not provide any credit/debit card information when creating the account, by default you will have a Lite account. The lite plan does apply usage caps for services offered under the plan. If you need to expand and remove such limits you would have to upgrade to a paid account. However, the free quotas are typically more than sufficient for testing and learning purposes. For example, under the Lite plan, you can use the “Watson Assistant” service with caps such as 10K API calls per month.

9.5 GOOGLE IAAS CLOUD SERVICES

Google Cloud, offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search and YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics, and machine learning. Registration requires a credit card or bank account details. Pricing is on a pay-as-you-go per second basis, and discounts are offered for certain services that run for extended periods. A free trial of \$300 worth of services is available for the first 12 months. Many services are always free up to a certain amount of use.

Google Cloud Platform provides Infrastructure as a Service, Platform as a Service, and Server-less Computing environments.

Google Cloud Platform is a part of Google Cloud, which includes the Google Cloud Platform public cloud infrastructure, as well as G Suite, Cloud Identity, Apigee, Firebase, enterprise versions of Android and Chrome OS, and Google Maps Platform. The platform and all its offerings can be managed via a customizable dashboard [Figure 76](#).

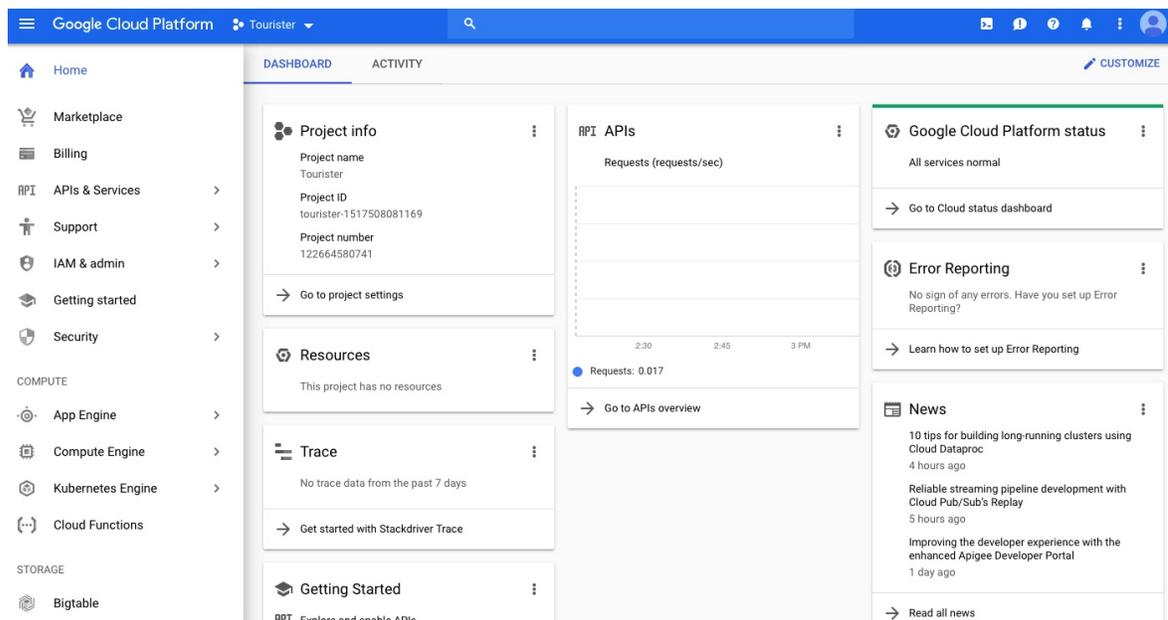


Figure 76: Gcloud dashboard [Source](#)

9.5.1 Cloud Computing Services and Products

All products here and their links can be found on

- <https://cloud.google.com>

We have copied the information from that location and made them conveniently available in this section.

9.5.1.1 Overview

A list of the Google Cloud Services is shown in [Figure 77](#).



Google Cloud

DEVELOPER'S CHEAT SHEET

2019.7.17
Created by the Google Developer Relations Team
Maintained at <https://github.com/progmanblings/google-cloud-4-words>
Feedback? [@progmanblings](https://twitter.com/progmanblings)

- COMPUTE**
 - App Engine: Managed app platform
 - Cloud Functions: Event-driven serverless functions
 - Cloud Run: Serverless for containerized applications
 - Compute Engine: VMs, GPUs, TPUs, Disks
 - Kubernetes Engine (GKE): Managed Kubernetes/containers
 - Autos: Enterprise hybrid/multi-cloud platform
- STORAGE**
 - Cloud Storage: Object storage and serving
 - Nearline: Archival occasional access storage
 - Coldline: Archival rare access storage
 - Persistent Disk: VM-attached disks
 - Cloud Filestore: Managed NFS server
- DATABASES**
 - Cloud Bigtable: Petabyte-scale, low-latency, non-relational
 - Cloud Datastore: Horizontally scalable document DB
 - Cloud Firestore: Strongly-consistent serverless document DB
 - Cloud Memorystore: Managed Redis
 - Cloud Spanner: Horizontally scalable relational DB
 - Cloud SQL: Managed MySQL and PostgreSQL
- DATA AND ANALYTICS**
 - BigQuery: Data warehouse/analytics
 - BigQuery BI Engine: In-memory analytics engine
 - BigQuery ML: BigQuery model training/serving
 - Cloud Composer: Managed workflow orchestration service
 - Cloud Data Fusion: Graphically manage data pipelines
 - Cloud Dataflow: Stream/batch data processing
 - Cloud Datacatalog: Managed Jupyter notebook
 - Cloud Dataprep: Visual data wrangling
 - Cloud Dataplex: Managed Spark and Hadoop
 - Cloud Pub/Sub: Global real-time messaging
 - Data Catalog: Metadata management service
 - Data Studio: Collaborative data exploration/dashboarding
 - Genomics: Managed genomics platform
- AI/ML**
 - AI Hub: Hosted AI component sharing
 - AI Platform: Managed platform for ML
 - AI Platform Data Labeling: Data labeling by humans
 - AI Platform Deep Learning VMs: Preconfigured VMs for deep learning
 - AI Platform Notebooks: Managed JupyterLab notebook instances
 - AI Platform Training: Parallel and distributed training
- AI Platform Predictions**
 - AutoML Natural Language: Custom text models
 - AutoML Tables: Custom structured data models
 - AutoML Translation: Custom domain-specific translation
 - AutoML Video Intelligence: Custom video annotation models
 - AutoML Vision: Custom image models
 - Cloud AI Building Blocks: Hosted AI component repository
 - Cloud Natural Language API: Text parsing and analysis
 - Cloud Speech-to-Text API: Convert audio to text
 - Cloud Text-to-Speech API: Job search with ML
 - Cloud Translation API: Convert text to audio
 - Cloud Video Intelligence API: Language detection and translation
 - Cloud Vision API: Scene-level video annotation
 - Cloud TPU: Hardware acceleration for ML
 - Dialogflow Enterprise Edition: Create conversational interfaces
 - Document Understanding AI: Analyze, classify, search documents
 - Recommendations AI: Create custom recommendations
 - Vision Product Search: Visual search for products
- NETWORKING**
 - Carrier Peering: Peer through a carrier
 - Direct Peering: Peer with ISP
 - Dedicated Interconnect: Dedicated private network connection
 - Partner Interconnect: Connect on-prem network to VPC
 - Cloud Armor: DDoS protection and WAF
 - Cloud CDN: Content delivery network
 - Cloud DNS: Programmable DNS serving
 - Cloud Load Balancing: Multi-region load distribution
 - Cloud NAT: Network address translation service
 - Firewall: Virtual private network correction
 - Network Service Tiers: Price vs performance tiering
 - Network Telemetry: Network telemetry service
 - Traffic Director: Service mesh traffic management
 - Google Cloud Service Mesh: Service-aware network management
 - Virtual Private Cloud: Software defined networking
- INTERNET OF THINGS (IOT)**
 - Cloud IoT Core: Device management and ingest data
- IDENTITY AND SECURITY**
 - Access Transparency: Audit cloud provider access
 - Binary Authorization: Kubernetes deploy-time security
 - Cloud Audit Log: Audit trails for GCP
 - Cloud Data Loss Prevention API: Classify and redact sensitive data
 - Cloud IAM: Hardware security module service
 - Cloud IAP: Resource access control
 - Cloud Identity: Manage users, devices & apps
 - Cloud Identity-Aware Proxy: Identity-based app sign in
 - Cloud Key Management Service: Hosted key management service
 - Cloud Resource Manager: Cloud project/metadata management
 - Cloud Security Scanner: App engine security scanner
 - Cloud Security Command Center: Asset inventory/discovery/search, management
 - Content-aware Access: End-user attribute-based access control
 - Event Threat Detection: Scans for suspicious activity
 - Security Key Enforcement: Two-factor authentication
 - Shielded VMs: Hardened VMs
 - Titan Security Key: Two-factor authentication (2FA) device
 - VPC Service Controls: VPC constraint data
- MANAGEMENT TOOLS**
 - Cloud Aps: APIs for cloud services
 - Cloud Billing: Billing and cost management tools
 - Cloud Billing API: Programmatically manage GCP billing
 - Cloud Console: Web-based management console
 - Cloud Deployment Manager: Templated infrastructure deployment
 - Cloud Mobile App: iOS/Android GCP manager app
- Cloud Shell**
 - Stackdriver Debugger: Live production debugging
 - Stackdriver Error Reporting: App error reporting
 - Stackdriver Logging: Centralized logging
 - Stackdriver Monitoring: Infrastructure and application monitoring
 - Stackdriver Profiler: CPU and heap profiling
 - Stackdriver Trace: Monitor GCP services
 - Stackdriver Trace: App performance insights
- DEVELOPER TOOLS**
 - Cloud SDK: CLI for GCP
 - Cloud Build: Continuous integration/delivery platform
 - Cloud Code: Cloud native IDE extensions
 - Cloud Source Repositories: Hosted private git repos
 - Cloud Scheduler: Managed cron job service
 - Cloud Tails: Asynchronous task execution
 - Cloud Tools for IntelliJ: IntelliJ GCP tool
 - Cloud Tools for PowerShell: PowerShell GCP tools
 - Cloud Tools for Visual Studio: Visual Studio GCP tools
 - Cloud Tools for Eclipse: Eclipse GCP tools
 - Container Registry: Private container registry/storage
 - Gradle App Engine Plugin: Gradle App Engine plugin
 - Maven App Engine plugin: Maven App Engine plugin
- MIGRATION TO GCP**
 - Cloud Data Transfer: Data migration tools/CLI
 - Google Transfer Appliance: Reliable data transport box
 - Cloud Storage Transfer Service: Cloud to cloud transfers
 - BigQuery Data Transfer Service: Bulk import analytics data
 - Migrate from Amazon Redshift: Migrate from Redshift to BigQuery
 - Migrate from Teradata: Migrate from Teradata to BigQuery
 - Migrate from Aerospike: Migrate VMs to GKE containers
 - Migrate for Compute Engine: Compute engine migration tools
 - VM Migration: VM migration tools
- API PLATFORM AND ECOSYSTEMS**
 - API Analytics: API metrics
 - API Monetization: Monetize APIs
 - Apigee API Platform: Develop, secure, monitor APIs
 - Apigee Sense: API protection from attacks
 - Apigee Hybrid: Manage hybrid/multi-cloud API environments
 - Cloud API Gateway: Cloud API gateway
 - Cloud Healthcare API: Healthcare system GCP interoperability
 - Developer Portal: API management portal
 - GCP Marketplace: Partner & open source marketplace
- GOOGLE MAPS PLATFORM**
 - Directions API: Get directions between locations
 - Distances Matrix API: Calculate travel times
 - Geocoding API: Convert address to/from coordinates
 - Geolocation API: Derive location without GPS
 - Maps Embed API: Web embedded maps
 - Maps JavaScript API: Dynamic web maps
 - Maps SDK for Android: Maps SDK for Android
 - Maps SDK for iOS: Maps SDK for iOS
 - Maps Static API: Web static maps
 - Maps Unity SDK: Unity SDK for games
 - Maps URLs: URL scheme for maps
 - Places API: Metadata about places (REST)
 - Places Library, Maps JS API: Metadata about places (JavaScript)
 - Places SDK for Android: Places SDK for Android
 - Places SDK for iOS: Places SDK for iOS
 - Roads API: Metadata about roads
 - Street View Static API: Static street view images
 - Street View Service: Interactive street view images
 - Time Zone API: Convert coordinates to timezone
- G SUITE PLATFORM**
 - App Maker: Assistive app building
 - Apps Script: Extend and automate everything
 - Editor Add-ons: Extend Docs, Sheets, Slides
 - Google Add-ons: Contextual apps in Gmail
 - Hangouts Chat Bots: Conversational bots in chat
 - Calendar API: Create and manage calendars
 - Classroom API: Provision and manage classrooms
 - Docs API: Create and edit documents
 - Drive API: Read and write files
 - Gmail API: Enhance Gmail
 - Sheets API: Read and write spreadsheets
 - Slides API: Create and edit presentations
 - Drive Picker: Drive file selection widget
 - Cloud Search: Unify search for enterprise
 - Admin SDK: Manage G Suite resources
 - Email Markup: Interactive email using schema.org
 - G Suite Marketplace: Storefront for integrated applications
 - Other G Suite APIs/SDKs: Contacts, Google+, Talk, Vault, ...
- MOBILE (FIREBASE)**
 - Cloud Firestore: Document store and sync
 - Cloud Functions for Firebase: Event-driven serverless applications
 - Cloud Storage for Firebase: Object storage and serving
 - Crashlytics: Crash reporting and analytics
 - Analytics: Create A/B test experiments
 - App-Check: App-Check search integration
 - Drop-in authentication: Drop-in authentication
 - Send message: Send device notifications
 - Link to app content: Link to app content
 - Web hosting with CDN/SSL: Web hosting with CDN/SSL
 - Send in-app: Send in-app contextual messages
 - App performance monitoring: App performance monitoring
 - Product user targeting: Product user targeting
 - Realtime database: Realtime data synchronization
 - Remote config: Remotely configure installed apps
 - Mobile testing: Mobile testing device farm
 - Mobile analytics: Mobile app analytics
 - ML Kit for mobile: ML APIs for mobile
- GCP FOUNDATIONAL OPEN SOURCE PROJECTS**
 - Apache Beam: Batch/streaming data processing
 - gRPC: RPC framework
 - Istio: Service container runtime
 - Knative: Serverless framework for Kubernetes
 - Kubernetes: ML Tool kit for Kubernetes
 - OpenCensus: Management of containerized applications
 - TensorFlow: Cloud native observability framework
 - ML framework: ML framework
- ADDITIONAL RESOURCES**
 - Google Cloud Home Page: cloud.google.com
 - Google Cloud Blog: cloud.google.com/blog
 - GCP Medium Publication: medium.com/google-cloud
 - Apigee Blog: apigee.com/about/blog
 - Firestore Blog: firebase.googleblog.com
 - G Suite Developers Blog: gsuite-developers.googleblog.com
 - Google Cloud System Status: status.cloud.google.com
 - Google Cloud Training: cloud.google.com/training
 - Google Developers Blog: developers.googleblog.com
 - Google Maps Platform Blog: mapsplatform.googleblog.com
 - Google Open Source Blog: opensource.googleblog.com
 - Google Security Blog: security.googleblog.com
 - Kaggle Home Page: www.kaggle.com
 - Kubernetes Blog: kubernetes.io/blog
 - Regions and Network Map: cloud.google.com/about/locations

Figure 77: Google Cloud Services [Source](#)

9.5.1.2 AI and Machine Learning

Google offers many machine learning and artificial intelligence tools, including tools for text-to-speech, speech-to-text, translation, and image and video analysis as well as various tools for making models and predictions and deploying pipelines and out-of-the-box algorithms.

- [AI Hub \(alpha\)](#): Discover, share, and deploy AI on Google Cloud.
- [Cloud AutoML \(beta\)](#): Easily train high-quality, custom ML models.
- [Cloud TPU](#) Train and run ML models faster than ever.
- [Cloud Machine Learning Engine](#): Build superior models and deploy them into production.
- [Cloud Talent Solution](#): Put AI to work on your hiring needs.
- [Dialogflow Enterprise Edition](#): Create conversational experiences across devices and platforms.
- [Cloud Natural Language](#): Derive insights from unstructured text.
- [Cloud Speech-to-Text](#): Speech-to-text conversion powered by ML.
- [Cloud Text-to-Speech](#): Text-to-speech conversion powered by ML.

- [Cloud Translation](#): Dynamically translate between languages.
- [Cloud Vision](#): Derive insight from images powered by ML
- [Cloud Video Intelligence](#): Extract metadata from videos.
- [Cloud Inference API \(alpha\)](#): Quickly run large-scale correlations over typed time-series datasets.
- [Firebase Predictions](#): Smart user segmentation based on predicted behavior.
- [Cloud Deep Learning VM Image](#): Preconfigured VMs for deep learning applications.

9.5.1.3 API management

API tools include monetization and analytic tools as well as deployment tools. Apigee Edge integrates these tools together into a platform for managing APIs through the use of API proxies, which are combined together with a service plan into an API product.

- [Apigee API Platform](#): Develop, secure, deploy, and monitor your APIs everywhere.
- [API Analytics](#): Insight into operational and business metrics for APIs.
- [API Monetization](#): Flexible, easy-to-use solution to realize value from APIs.
- [Apigee Sense](#): Intelligent behavior detection to protect APIs from attacks.
- [Cloud Endpoints](#): Develop, deploy, and manage APIs on GCP.
- [DeveloperPortal](#): Enable developers and API teams with a turnkey self-service platform.
- [Apigee healthcare APIx](#): Accelerate building new FHIR API-based digital services.
- [Apigee Open Banking APIx](#): Accelerate open banking and PSD2 compliance.
- [Cloud Healthcare API](#): Secure APIs powering actionable healthcare insights.

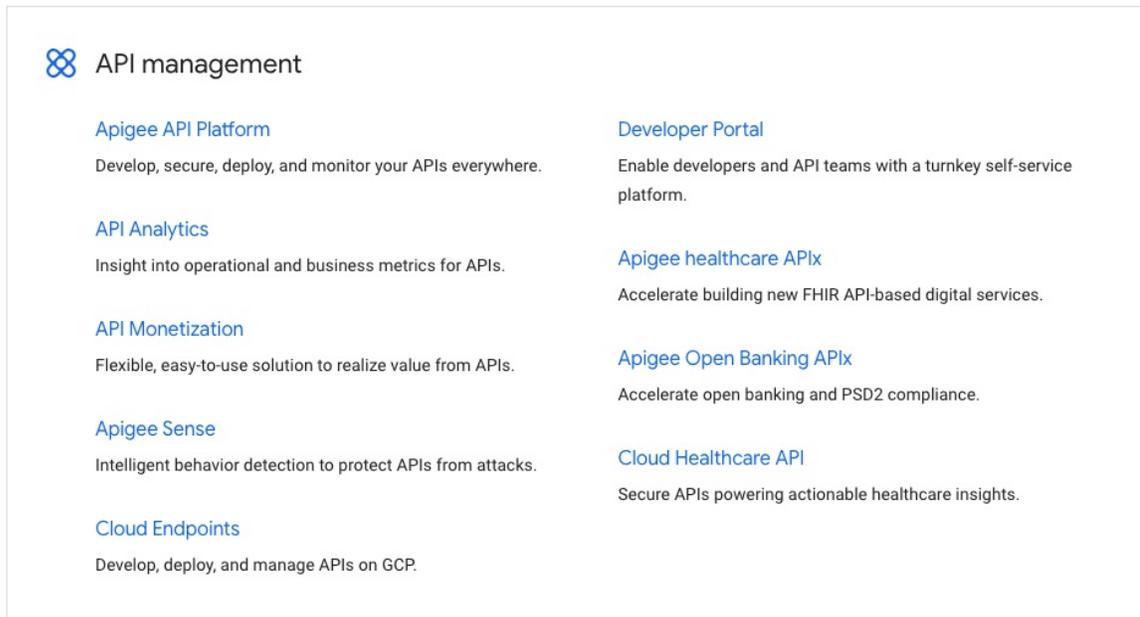


Figure 78: Google API management [Source](#)

9.5.1.4 Compute

Google Cloud Compute services offer infrastructure as a service tools including virtual machines, containers, as well as an app engine for deploying web, mobile and IoT apps.

- [Compute Engine](#): Scalable, high-performance VMs.
- [App Engine](#): Serverless application platform for apps and backends.
- [Google Kubernetes Engine](#): Run containerized applications.
- [GKE On-Prem \(alpha\)](#): Make apps “cloud-ready” and move them to the cloud at your own pace.
- [Cloud Functions](#): Event-driven serverless compute platform.
- [Cloud Functions for Firebase](#): Run mobile backend code without managing servers.
- [Knative](#): Components to create modern, Kubernetes-native cloud-based software.
- [Shielded VMs \(beta\)](#): Hardened virtual machines on GCP.
- [Container security](#): Secure your container environment on GCP.
- [Graphics Processing Unit \(GPU\)](#): Leverage GPUs on Google Cloud for machine learning, scientific computing, and 3D visualization.

9.5.1.5 Data Analytics

Google Cloud's data analytics services include serverless data warehousing, tools for running Hadoop and Spark clusters, data preparation and processing, creating dashboards and reports, NoSQL databases, and a tool that lets you experiment with transforming, analyzing, modeling, and predicting data.

- [BigQuery](#): A fully managed, highly scalable data warehouse with built-in ML.
- [Cloud Dataflow](#): Real-time batch and stream data processing.
- [Cloud Dataproc](#): Managed Spark and Hadoop service.
- [Cloud Datalab](#): Explore, analyze, and visualize large datasets.
- [Cloud Dataprep](#): Cloud data service to explore, clean, and prepare data for analysis.
- [Cloud Pub/Sub](#): Ingest event streams from anywhere, at any scale.
- [Cloud Composer](#): A fully managed workflow orchestration service built on Apache Airflow.* [Genomics](#): Power your science with Google Genomics.
- [Google Marketing Platform](#): Enterprise analytics for better customer experiences.
- [Google Data Studio](#): Tell great data stories to support better business decisions.
- [Firebase Performance Monitoring](#): Gain insight into your app's performance.

9.5.1.6 Databases

Google offers a range of databases including NoSQL, managed file system storage, and VM and container storage.

- [Cloud SQL](#): MySQL and PostgreSQL database service.
- [Cloud Bigtable](#): NoSQL wide-column database service.
- [Cloud Spanner](#): Mission-critical, scalable, relational database service.
- [Cloud Memorystore](#): Fully managed in-memory data store service.
- [Cloud Firestore](#): Store mobile and web app data at global scale.
- [Firebase Realtime Database](#): Store and sync data in real time.

A flow chart is even provided for helping determine the best service for your

needs [Figure 79](#).

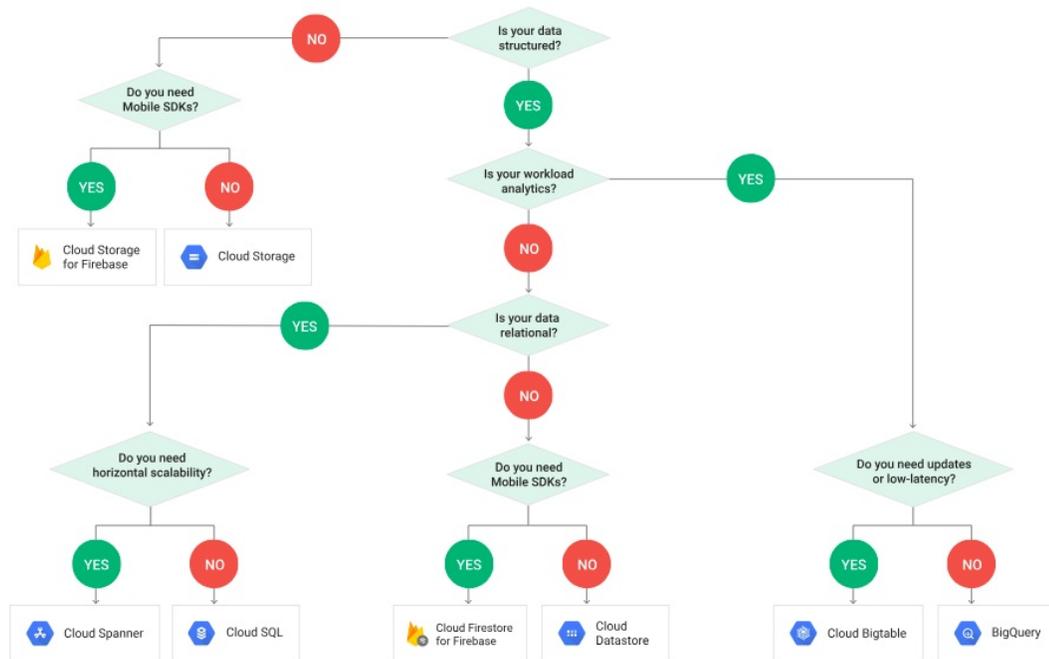


Figure 79: Gcloud db flow

9.5.1.7 Developer Tools

Google's developer tools include tools for Visual Studio, IntelliJ, Google Cloud, and Powershell, cloud-hosted git repositories, a infrastructure for testing mobile apps, and a deployment management tool.

- [Cloud SDK](#): CLI for GCP products and services.
- [Container Registry](#): Store, manage, and secure your Docker container images.
- [Cloud Build](#): Continuously build, test, and deploy.
- [Cloud Source Repositories](#): A single place for your team to store, manage, and track code.
- [Cloud Scheduler \(beta\)](#): Fully managed cron job service.
- [Cloud Tasks \(beta\)](#): Asynchronous task execution.
- [Cloud Tools for IntelliJ](#): Debug production cloud apps inside IntelliJ.
- [Cloud Tools for PowerShell](#): Full cloud control from Windows PowerShell.
- [Cloud Tools for Visual Studio](#): Deploy Visual Studio applications to GCP.
- [Cloud Tools for Eclipse](#): Deploy Eclipse projects to GCP.

- [Gradle App Engine Plugin](#): Use Gradle for your App Engine projects.
- [Maven App EnginePlugin](#): Use Maven for your App Engine projects.
- [Cloud Test Lab](#): On-demand testing infrastructure for Android apps.
- [Firebase Crashlytics](#): Prioritize and fix stability issues faster.

9.5.1.8 Internet of Things

Google's internet of things offerings include a tool for device connection management and two tools for edge computing, one of which is in beta and the other is only accessible by request currently.

- [Cloud IoT Core](#): Secure device connection and management.
- [Edge TPU \(early access\)](#): Purpose-built ASIC designed to run inference at the edge.
- [Cloud IoT Edge \(alpha\)](#): Deliver Google AI capabilities at the edge.

9.5.1.9 Management Tools

Management tools provide a variety of services for managing cloud applications. Google's Stackdriver has components for managing logs, monitoring exceptions, latency information, and overall health and also has a debugger component. Google also offers a web UI, a mobile app, and a command line interface for monitoring and managing cloud applications. Their cost management service includes components for monitoring and reporting, controlling spending and billing, and sizing recommendations for virtual machines.

- [Stackdriver](#): Monitoring and management for services, containers, applications, and infrastructure.
- [Monitoring](#): Monitoring for applications on GCP and AWS.
- [Service Monitoring \(early access\)](#): Stackdriver Service monitoring for Istio and Google App Engine services.
- [Logging](#): Logging for applications on GCP and AWS.
- [Error Reporting](#): Identifies and helps you understand application errors.
- [Trace](#): Find performance bottlenecks in production.
- [Debugger](#): Investigate code behavior in production.
- [Profiler \(beta\)](#): Low-impact CPU and heap profiling to reduce latency.
- [Transparent Service Level Indicators](#): Monitor Google Cloud services and

their effects on your workloads.

- [Cloud Deployment Manager](#): Manage cloud resources with simple templates.
- [Cloud Console](#): GCP's integrated management console.
- [Cloud Shell](#): Command-line management from any browser.
- [Cloud Mobile App](#): Manage GCP services from your mobile device.
- [Cost management](#): Tools for monitoring, controlling, and optimizing your costs.
- [Cloud APIs](#): Programmatic interfaces for all GCP services.

9.5.1.10 Media and Migration

Google currently offers two media tools, Anvato for live-streaming videos, and Zync Render for rendering videos.

- [Anvato](#): Stream live and on-demand video to any device.
- [Zync Render](#): Render directly from your 3D modeling tools, quickly and cost efficiently.

9.5.2 Migration

Google's migration tools are geared towards transferring data or applications fully or partially to the cloud.

- [Cloud Data Transfer](#): Command-line tools for developers to transfer data over the network.
- [Transfer Appliance](#): Rackable storage server for shipping large volumes of data to Google Cloud.
- [Cloud Storage Transfer Service](#): Transfer data between cloud storage services such as AWS S3 and Google Cloud Storage.
- [BigQuery Data Transfer Service](#): Fully managed data import service for BigQuery.
- [Velostrata](#): Purpose-built, enterprise-grade migration to Google Cloud.
- [VM Migration](#): Migrating VMs is a fast, effective way to get started in Google Cloud.

9.5.2.1 Networking

The Google Virtual Private Cloud Network is Google's own world-wide network where you can host your applications and services. Google also has load balancing, DNS, CDN, and connectivity tools for working with this network.

- [Virtual Private Cloud \(VPC\)](#): VPC networking for GCP resources.
- [Cloud Load Balancing](#): High-performance, scalable load balancing.
- [Cloud Armor](#): Protect your services against DoS and web attacks.
- [Cloud CDN](#): Content delivery on Google's global network.
- [Cloud NAT](#): GCP-managed high-performance Network Address Translation.
- [Cloud Interconnect](#): Connect directly to GCP's network edge.
- [Cloud VPN](#): Securely connect to your GCP VPC via the public internet.
- [Cloud DNS](#): Reliable, resilient, low-latency DNS serving.
- [Network Service Tiers](#): Optimize your network for performance or cost.
- [Network Telemetry](#): In-depth network telemetry to keep your services secure.

9.5.2.2 Security

Google's security offerings are aimed at protection from phishing, ransomware, and DoS attacks, controlling the transfer of data, controlling access to applications and resources, and monitoring and controlling vulnerabilities and incidents.

- [Cloud IAM](#): Fine-grained identity and access management.
- [Cloud Identity for Customers and Partners \(beta\)](#): Add Google-grade identity and access management to your apps.
- [Firebase Authentication](#): Simple, free multi-platform sign-in.
- [Cloud Identity-Aware Proxy](#): Use identity and context to guard access to your applications and VMs.
- [Cloud Data Loss Prevention](#): Discover and redact sensitive data.
- [Security Key Enforcement](#): Enforce the use of security keys to help prevent phishing.
- [Titan Security Key](#): Defend against account takeovers from phishing attacks.
- [Cloud HSM](#): Protect cryptographic keys with a fully managed hardware security module service.

- [VPC Service Controls \(beta\)](#): Define security perimeters for sensitive data in Google Cloud Platform services.
- [Cloud Key Management Service](#): Manage encryption keys on GCP and encrypt secrets in GKE.
- [Resource Manager](#): Hierarchically manage resources on GCP.
- [Cloud Security Command Center \(beta\)](#): Comprehensive security and data risk platform for GCP.
- [Cloud Security Scanner](#): Automatically scan your App Engine apps.
- [Access Transparency](#): Get visibility over your cloud provider through near real-time logs.
- [Binary Authorization \(beta\)](#): Deploy only trusted containers on Kubernetes Engine.

9.5.2.3 Storage

Google's storage services include all the services mentioned in its database services as well as Persistent Disk, which offers block service for virtual machines and containers.

- [Cloud Storage](#): Object storage with global edge-caching.
- [Persistent Disk](#): Block storage for VM instances.
- [Cloud Storage for Firebase](#): Store and serve content with ease.
- [Cloud Filestore](#): High-performance file storage.
- [Drive Enterprise](#): Cloud-based content collaboration and storage.

9.5.2.4 Google IaaS Example

To demonstrate an example of what Google IaaS solutions are available, Google has provided these options: <https://cloud.google.com/solutions/>

Locate the Try for Free button option on the top right portion of the webpage. The free trial allows a person access to all Cloud Platform Products. You get everything you need to build and run your apps, websites and services, including Firebase and the Google Maps API. Note, you will be asked you for your credit card to make sure you are not a robot. You will not be charged unless you manually upgrade to a paid account. Disclaimer: Please be aware that you pay for this service only after you accrue costs, via an automatic charge when you

reach your billing threshold or 30 days after your last automatic payment, whichever comes first. You will be presented the option to agree and continue. Once you satisfy all the formalities you will be granted the 12-month free trial.

9.5.2.5 Google Cloud Console Overview

This material was obtained from Google on the [Cloud Console Tour](#). This information covers the core features of Cloud Console to get you ready to build and manage your applications on Google Cloud Platform. You will learn about the following concepts: * GCP projects and resources * High-level resource overview and activity logs * Console navigation and search * User and permissions management * Technical support * GCP's browser-based command line

9.5.2.6 Use GCP Resources

GCP resources are the fundamental components that make up all Google Cloud services. Resources are organized hierarchically and help organize your work on GCP. Projects are the first level of the resource hierarchy, and they contain other low-level resources like Cloud Storage buckets and Compute Engine instances. Project navigation Easily navigate across your GCP projects using the scopicker in Cloud Console. Switching projects will tailor the view to that project and all of its child resources.

9.5.2.7 Project navigation

Easily navigate across your GCP projects using the scopicker in Cloud Console. Switching projects will tailor the view to that project and all of its child resources.

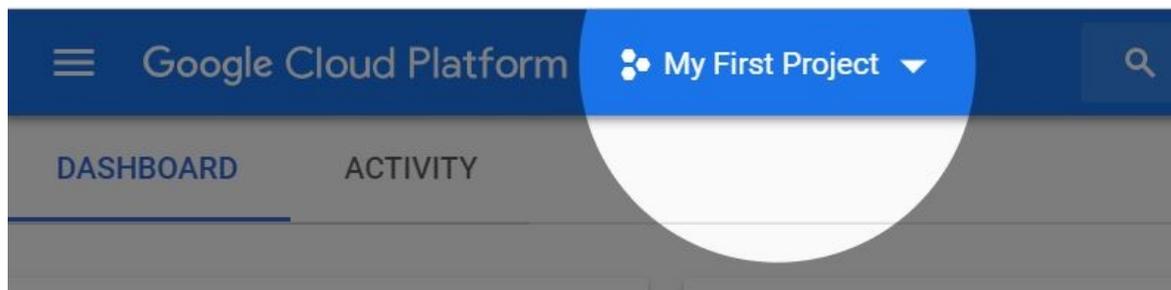
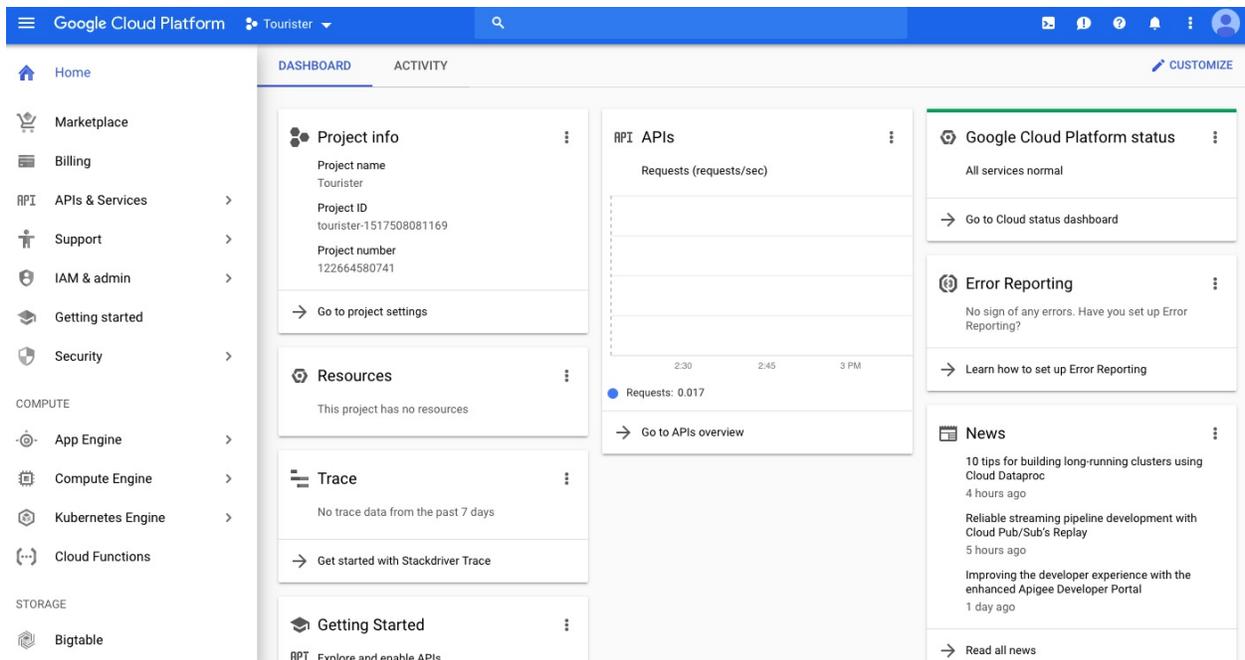


Figure 80: Scope Picker Example

More detail regarding resources can be found at: <https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy>

9.5.2.8 Navigate Google Cloud Services

Service navigation Google Cloud services are accessible in the left-hand navigation menu organized by product area including Big Data, Compute, Networking, etc.

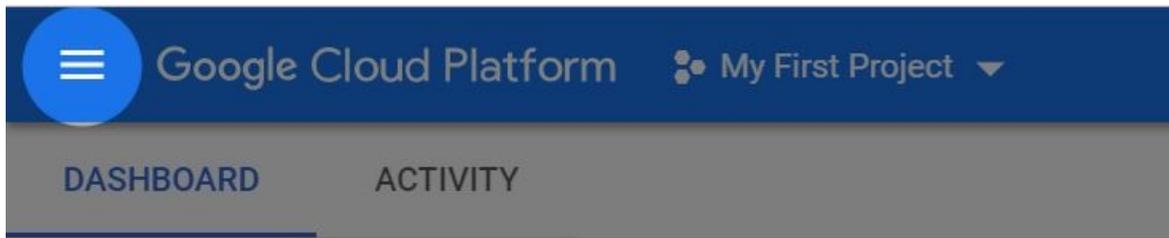


Figure 81: Left-Hand Navigation Example

9.5.2.9 Section pinning

For any service that you visit regularly, pin the section to the top of the navigation menu by hovering over the section item and clicking the pin icon. See a high-level overview of any project * Home dashboard The Home dashboard provides a high-level overview of the selected GCP project, highlighting key metrics, billing, and other useful information. * Customization You can customize your dashboard by clicking Customize. Any card can be hidden, shown, and reordered on the page. Each card also has custom options accessible from the overflow menu when hovering a card. Customize Figure:

9.5.2.10 View activity across your GCP resources

With Activity Stream, you will be able to understand all the activities that occur across your GCP resources in one place. See what your teammates are updating in any project to track down issues and audit access. Easily filter through the feed to find exactly what you need.

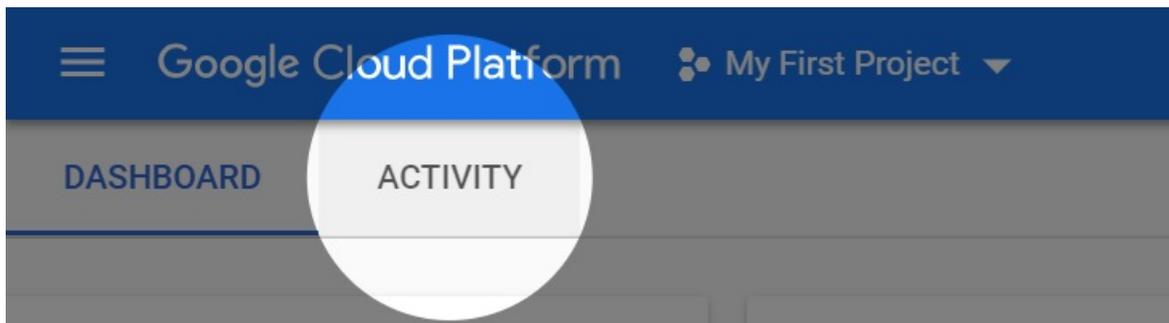


Figure 82: Activity Example

9.5.2.11 Search across Cloud Console

The search bar in Cloud Console allows you to quickly access Google Cloud products and any of your resources across GCP. Try running a search for App Engine or the name of one of your projects.



Figure 83: Searchbar Example

9.5.2.12 Get support anytime

If you ever get stuck, or need help navigating the world of GCP, the Google support team is here to help. Access support from the navigation menu.

More information regarding support options and details can be found at: <https://cloud.google.com/support>

9.5.2.13 Manage users and permissions

Google Cloud Identity and Access Management (Cloud IAM) enables you to manage and create permissions for your GCP resources. As your team continues to grow, you can grant access to teammates using Cloud IAM in the [IAM & Admin](#) section. Add users, groups, or service accounts and assign them any number of roles to grant them the permissions they need.

Additional resources for Google Cloud Identity and Access Management documentation: <https://cloud.google.com/iam/docs>

9.5.2.14 Access the command line from your browser

Google Cloud Shell provides you with command-line access to your cloud resources directly from your browser. You can easily manage your projects and resources without having to install the Google Cloud SDK or other tools on your system. With Cloud Shell, the Cloud SDK `gcloud` command-line tool and other utilities you need are always available, up to date and fully authenticated when you need them.



Figure 84: Cloudshell Example

Reference to more documented detail about the Cloudshell:
<https://cloud.google.com/shell>

9.5.3 Create a VM Example

Since we have been exploring virtual machines in this class, I thought that I would provide an additional example explaining how to create a Linux virtual machine instance in Compute Engine using the Google Cloud Platform Console. Navigate to Compute Engine Open the menu on the left side of the console. Then, select the `Compute Engine` section.



Figure 85: Menu Example

9.5.3.1 Create a virtual machine instance

Click the Create instance button. * Select a name and zone for this instance. * In the Firewall selector, select Allow HTTP traffic. This opens port 80 (HTTP) to access the app. * Click the Create button to create the instance. Note: Once the instance is created your billing account will start being charged according to the GCE pricing. You will remove the instance later to avoid extra charges.

9.5.3.2 VM instances page

While the instance is being created take your time to explore the VM instances page. * At the bottom you can see the list of your VMs * At the top you can see a control panel allowing you to * Create a new VM instance or an instance group * Start, stop, reset and delete instances.

Compute Engine lets you use virtual machines that run on Google's infrastructure. Create micro-VMs or larger instances running Debian, Windows, or other standard images. Create your first VM instance, import it using a migration service, or try a quickstart to build a sample app. More detail can be found at the following link regarding VM instances:
<https://cloud.google.com/compute/?>

[hl=en_US&_ga=2.98598104.-779866669.1550427921](#)

9.5.3.3 Connect to your instance

When the VM instance is created, you'll run a web server on the virtual machine. The SSH buttons in the table will open up an SSH session to your instance in a separate window.

For this tutorial you will connect using Cloud Shell. Cloud Shell is a built-in command line tool for the console.

Open the Cloud Shell Open Cloud Shell by clicking the `Activate Cloud Shell` button in the navigation bar in the upper-right corner of the console. Wait for the instance creation to finish. The instance creation needs to finish before the tutorial can proceed. The activity can be tracked by clicking the notification menu from the navigation bar at the top.

To Connect to the instance, enter the following command to SSH into the VM. If this is your first time using SSH from Cloud Shell, you will need to create a private key. Enter the zone and name of the instance you created.

```
$ gcloud compute --project \"regal-buckeye-232200\" ssh --zone <vm-zone> <vm-name>
```

9.5.3.4 Run a simple web server

Create a simple index.html file with the following command inside the parenthesis and double quotes: `echo Hello World index.html`

Then, enter this command to run a simple Python webserver:

```
$ sudo python -m SimpleHTTPServer 80
```

9.5.3.5 Visit your application

Visit your webserver at the IP address listed in the External IP column.

9.5.3.6 Cleanup

To remove your instance, select the checkbox next to your instance name and click the Delete button.

It is recommended to review the Google Cloud Platform on Github for additional examples. Here is the link to the GCP on GitHub: <https://github.com/GoogleCloudPlatform>.

9.6 OPENSTACK

9.6.1 Introduction

OpenStack can be described as a cloud operating system. OpenStack can be used on private or public clouds to manage large amounts of compute, storage and network resources. OpenStack is built up using a large number of small software components, which will be described in more detail in the next couple of sections. Another important aspect of OpenStack is that it is completely OpenSource, which means that anyone can access and use the product without having to pay any licensing or any other fee. And since the source code is publicly available under the Apache-2.0 License developers can modify and use OpenStack as needed.

OpenStack is managed and maintained by the “The OpenStack Foundation”, which is a non-profit organization which organizes the development of OpenStack and keeps the OpenStack community running.

9.6.2 OpenStack Architecture

OpenStack consists of a large number of small components. Which components to use for your OpenStack deployment depends on your usecases and requirements. The existing components can be integrated to achieve the desired deployment by carefully examining and understanding each component. [Figure 86](#) shows a high-level architecture diagram of OpenStack which gives a clear understanding of how the overall framework is organized.

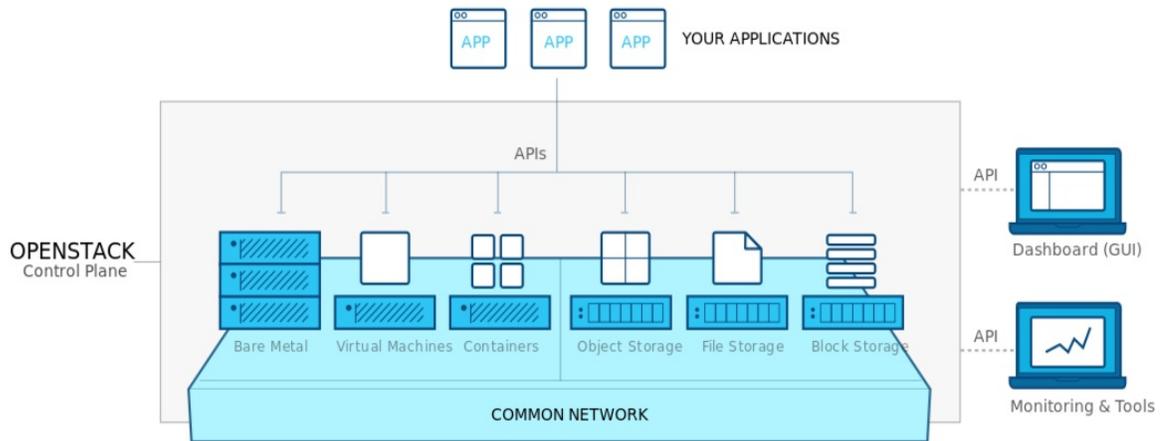


Figure 86: OpenStack Overview

Image reference - <https://www.openstack.org/software>

However each of the high-level components are in constructed using several small components. [Figure 87](#) shows one such popular deployment and its architecture.

- IRONIC - Bare Metal Provisioning Service
- CYBORG - Accelerators resource management

Storage

- SWIFT - Object store
- CINDER - Block Storage
- MANILA - Shared filesystems

Networking

- NEUTRON - Networking
- OCTAVIA - Load balancer
- DESIGNATE - DNS service

Shared Services

- KEYSTONE - Identity service
- GLANCE - Image service
- BARBICAN - Key management
- KARBOR - Application Data Protection as a Service
- SEARCHLIGHT - Indexing and Search

Orchestration

- HEAT - Orchestration
- SENLIN - Clustering service
- MISTRAL - Workflow service
- ZAQAR - Messaging Service
- BLAZAR - Resource reservation service
- AODH - Alarming Service

Workload Provisioning

- MAGNUM - Container Orchestration Engine Provisioning
- SAHARA - Big Data Processing Framework Provisioning
- TROVE - Database as a Service

Application Lifecycle

- MASAKARI - Instances High Availability Service
- MURANO - Application Catalog
- SOLUM - Software Development Lifecycle Automation
- FREEZER - Backup, Restore, and Disaster Recovery

API Proxies

- EC2API - EC2 API proxy

Web Frontend

- HORIZON - Dashboard

This list just includes the open stack services. Additionally, there are several other major component groups such as Operational Services, Add-Ons to Services and Bridges for Adjacent Tech listed in the services page at - [OpenStack Services](#).

9.6.4 Core Services

Among all the service components that are available for OpenStack, there are 9 services that are considered to be *Core* services, these services are essential to any OpenStack deployment.

9.6.4.1 Nova - Compute

Nova provides services to manage virtual machines in cloud environments. It is also capable of handling other compute resources such as containers and is highly scalable.

9.6.4.2 Glance - Image Services

Glance adds image services capabilities to OpenStack, this allows open stack users to manage virtual machine images and provides services such as image registration and discovery.

9.6.4.3 Swift - Object Storage

Swift allows developers refer to files and other data similar to object references. All actual storage and management is handled by Swift so the developers do not need to worry about where to store data and files

9.6.4.4 Cinder - Block Storage

Cinder provides block storage management capabilities to OpenStack, Cinder supports several block storage devices underneath and provides an unified API so that developers do not need to worry or think about what device is been used underneath.

9.6.4.5 Neutron - Networking

Neutron provides networking capabilities to OpenStack. It allows the creation and management of various networks that are used as the communication medium for OpenStack deployments. Neutron supports multi-tenancy and scale to large deployments with ease. Extension frameworks for Neutron allow users to deploy more advance network features such as VPN's, firewalls, load-balancer, etc.

9.6.4.6 Horizon - Dashboard

Horizon is the graphical user interface(GUI) for OpenStack, which developers can use to manage and monitor their OpenStack deployment.

9.6.4.7 Keystone - Identity Service

Keystone is the identity management services in OpenStack, it keeps a list of users and maps all the access rights for each user for all the cloud services that are available in the OpenStack deployment. Keystone supports several authentication mechanisms such as classical user name password based authentication and token based systems

9.6.4.8 Ceilometer - Telemetry

Ceilometer provides developers with billing and usage services that allow

developers to bill end users based on each individuals usage amounts. It also records and saves usage values of the cloud for each user so that anything that needs verification can be done.

9.6.4.9 Heat - Orchestration

Heat is the orchestration component of OpenStack. It allows developers to use a requirement files that define the resources requirements for cloud application, which can later be referenced when needed.

9.6.5 Access from Python and Scripts

9.6.5.1 Libcloud

Libcloud provides for some selected functionality a reasonable interface to OpenStack. More information is provided in Section [Python libcloud](#). More advanced resources are exposed through REST interfaces that may not be available in Libcloud. To access them new client libraroes that are not included in libcloud need to be developed. Such functionality was exposed for example in FutureGrid to access cloud metric data.

9.6.5.2 DevStack

It is very convenient to be able to set up an OpenSTack depoyment for development purposes on our own single computer.

DevStack is a set of scripts that can be used by developers to manage and maintain their OpenStack development. DevStack was developed to increase the ease of use for developers. It is very useful to setup a developer environment where you can test your deployment. More detailed information regarding DevStack can be found in their official documentaion - [DevStack documentation](#)

9.7 PYTHON LIBCLOUD



Construction

❶ This section has some features missing and does not yet use cloudmesh v4

With all the cloud providers and cloud services that are currently available, it becomes hard to manage and maintain services that work with several services. Therefore it is good to have a unified service that allows developers to access many of the cloud services through a single abstraction. Apache Libcloud is a python library that was developed for this purpose. Apache Libcloud provides a unified API for many of the popular cloud providers and services.

Apache Libcloud currently supports many providers, the complete list of providers that are supported can be found at [Supported Providers](#)

However, it is good to keep in mind that the Libcloud API might not support some of the advanced features that are provided by some cloud services or some of the most recent features that are yet to be integrated into Libcloud

9.7.1 Service categories

Libcloud provides many services and defines several categories to distinguish between the main types of services. The list of categories is as bellow, More details about this list can be found at [Categories](#) The list is extracted from the LibCloud documentation [LibCloud Docs](#)

- Cloud Servers and Block Storage - services such as Amazon EC2 and Rackspace CloudServers
- Cloud Object Storage and CDN - services such as Amazon S3 and Rackspace CloudFiles
- Load Balancers as a Service - services such as Amazon Elastic Load Balancer and GoGrid LoadBalancers
- DNS as a Service - services such as Amazon Route 53 and Zerigo
- Container Services - container virtualization like Docker and Rkt as well as container based services
- Backup as a Service - services such as Amazon EBS and OpenStack Freezer

each category has a set of terms that represent various constructs and services. For example the following list is the list of terms used in for Compute related services, this list is extracted from the [Compute docs](#)

9.7.1.0.1 Compute

- Node - represents a cloud or virtual server.
- NodeSize - represents node hardware configuration. Usually this is amount of the available RAM, bandwidth, CPU speed and disk size. Most of the drivers also expose an hourly price (in dollars) for the Node of this size.
- NodeImage - represents an operating system image.
- NodeLocation - represents a physical location where a server can be.
- NodeState - represents a node state. Standard states are: running, rebooting, terminated, pending, stopped, suspended, paused, erro, unknown.

9.7.1.0.2 Key Pair Management

- KeyPair - represents an SSH key pair object.

9.7.1.0.3 Block Storage

- StorageVolume - represents a block storage volume
- VolumeSnapshot - represents a point in time snapshot of a StorageVolume

You can find more complete information on Libcloud in the official documentations, this article will only provide a brief summary in most part: [Apache Libcloud Documentaions](#)

9.7.2 Installation

Libcloud can be installed via pip. Execute the following command in order to install Libcloud

```
pip install apache-libcloud
```

9.7.3 Quick Example

The following basic example shows you how the Python Libcloud library can be used to access information in a cloud provider

```
from pprint import pprint
import libcloud

cls = libcloud.get_driver(
    libcloud.DriverType.COMPUTE,
    libcloud.DriverType.COMPUTE.OPENSTACK)
```

```
driver = cls('username', 'api key')
pprint(driver.list_sizes())
pprint(driver.list_nodes())
```

9.7.4 Managing your cloud credentials

Often you will need as part of your code to access cloud credentials. These could be read in interactively, from environment variables, or from a configuration file. To make things easy for now, we assume the credentials are stored in a yaml file that is stored in `~/.cloudmesh/cloudmesh.4.yaml`. An example is listed at

- <https://raw.githubusercontent.com/cloudmesh/cloudmesh-configuration/master/cloudmesh/configuration/etc/cloudmesh.yaml>

With the help of this yaml file it is now easy to manage credentials for multiple clouds. We provide next a simple example on how to get the credentials for the cloud called aws.

```
from cloudmesh.common.util import path_expand
from cloudmesh.management.configuration.config import Config

name="aws"

credentials = Config()["cloudmesh"]["cloud"][name]["credentials"]
```

The last function can also be called via

```
credentials = Config().credentials("cloud", name)
```

Which is a convenient method to access the credentials for a named cloud.

Certainly you should be encrypting this file and an extension could be developed by you to manage the encryption and decryption for example while using your password protected public/private keypair or other methods.

9.7.5 Working with cloud services

In the following section we will look into how Libcloud can be used to perform various functions in specific cloud providers. One of the main aspects that change between different cloud providers is how authentication is done. Because of the unified API most of the other features are executed in the same manner.

9.7.5.1 Authenticating with cloud providers

Depending on the cloud provider, how Libcloud is granted access to your cloud account may differ, next we will look at some such examples

There are two main steps that are common to all providers

1. Using the `get_driver()` method to obtain a reference to the cloud provider driver
2. Instantiating the driver with the credentials to access the cloud

After you obtain the connection, it can be used to invoke various services

9.7.5.1.1 Amazon AWS

o get a driver via libcloud for AWS you first have to set up the `cloudmesh.yaml` file and install the convenience methods from cloudmesh as documented in

- <https://cloudmesh-community.github.io/cm/install.html#installation-via-pip-development>

This will provide you with a convenient config method that reads the Azure configuration parameters from the `cloudmesh.yaml` file which you need to place in `~/.cloudmesh`

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
from cloudmesh.common.util import path_expand
from cloudmesh.management.configuration.config import Config
from pprint import pprint

# Azure related variables

name = "azure"
# This assumes the cloudname for azure to be *azure*

credentials = Config().credentials("cloud", name)

pprint(credentials)

#AZURE_MANAGEMENT_CERT_PATH = path_expand('~/.cloudmesh/azure_cert.pem')

driver = get_driver(Provider.AZURE)
connection = self.driver(
    credentials["EC2_ACCESS_ID"],
    credentials["EC2_SECRET_KEY"],
    region=credentials["region"])

pprint(connection.__dict__)
```

9.7.5.1.2 Azure

9.7.5.1.2.1 Azure Classic Driver

Please note that libcloud has multiple drivers to interact with Azure. The following is an example using the classic method.

To get a driver via libcloud for azure you first have to set up the cloudmesh.yaml file and install the convenience methods from cloudmesh as documented in

- <https://cloudmesh-community.github.io/cm/install.html#installation-via-pip-development>

This will provide you with a convenient config method that reads the Azure configuration parameters from the cloudmesh.yaml file which you need to place in `~/.cloudmesh`

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
from cloudmesh.common.util import path_expand
from cloudmesh.management.configuration.config import Config
from pprint import pprint

# Azure related variables

name = "azure"
# This assumes the cloudname for azure to be *azure*

credentials = Config().credentials("cloud", name)

pprint(credentials)

#AZURE_MANAGEMENT_CERT_PATH = path_expand('~/.cloudmesh/azure_cert.pem')

driver = get_driver(Provider.AZURE)
connection = driver(
    subscription_id=credentials["AZURE_SUBSCRIPTION_ID"],
    key_file=path_expand(credentials["AZURE_KEY_FILE"])
)

pprint(connection.__dict__)
```

9.7.5.1.2.2 Azure New Driver

The following is an example using the Azure Resource Management (ARM) method.

- <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview>

To connect to Azure you need your tenant ID and subscription ID. Using the Azure cross platform CLI, use `azure account list` to get these values.

```
azure ad app create --name "<Your Application Display Name>" --home-page "<https://YourApplicationHomePage>" --identifier
azure ad sp create "<Application_Id>"
azure role assignment create --objectId "<Object_Id>" -o Owner -c /subscriptions/{subscriptionId}/
```

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
from cloudmesh.management.configuration.config import Config
from pprint import pprint

# Azure related variables

name = "azure_arm"
# This assumes the cloudname for azure to be *azure_arm*

credentials = Config().credentials("cloud", name)

pprint(credentials)

driver = get_driver(Provider.AZURE_ARM)
connection = driver(
    tenant_id=credentials["TENANT_ID"],
    subscription_id=credentials["AZURE_SUBSCRIPTION_ID"],
    key=credentials["APPLICATION_ID"],
    secret=credentials["PASSWORD"]
)

pprint(connection.__dict__)
```

9.7.5.1.3 OpenStack

To get a driver via libcloud for OpenStack you first have to set up the cloudmesh.yaml file and install the convenience methods from cloudmesh as documented in

- <https://cloudmesh-community.github.io/cm/install.html#installation-via-pip-development>

This will provide you with a convenient config method that reads the Azure configuration parameters from the cloudmesh.yaml file which you need to place in `~/.cloudmesh`

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
from cloudmesh.common.util import path_expand
from cloudmesh.management.configuration.config import Config
from pprint import pprint

# Azure related variables

name = "chameleon"
# This assumes the cloudname for azure to be *azure*

credentials = Config().credentials("cloud", name)

pprint(credentials)

#AZURE_MANAGEMENT_CERT_PATH = path_expand('~/.cloudmesh/azure_cert.pem')

driver = get_driver(Provider.AZURE)
```

```

connection = self.driver(
    credentials["OS_USERNAME"],
    credentials["OS_PASSWORD"],
    ex_force_auth_url=credentials['OS_AUTH_URL'],
    ex_force_auth_version='2.0_password',
    ex_tenant_name=credentials['OS_TENANT_NAME'])

pprint(connection.__dict__)

```

9.7.5.1.4 Google

Google cloud and account setup:

1. Go to the Google Developers Console
2. Select your project
3. In the left sidebar, go to “APIs & auth”
4. Click on “Credentials” then “Create New Client ID”
5. Select “Installed application” and “Other” then click “Create Client ID”
6. For authentication, you will need the “Client ID” and the “Client Secret”
7. You will also need your “Project ID” (a string, not a numerical value) that can be found by clicking on the “Overview” link on the left sidebar.

- <https://libcloud.readthedocs.io/en/latest/compute/drivers/gce.html>

```

from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
from cloudmesh.common.util import path_expand
from cloudmesh.management.configuration.config import Config
from pprint import pprint

credentials = Config().credentials("cloud", "GCE")

pprint(credentials)

driver = get_driver(Provider.GCE)
# Datacenter is set to 'us-central1-a' as an example, but can be set to any
# zone, like 'us-central1-b' or 'europe-west1-a'
connection = driver(
    credentials['SERVICE_ACCOUNT_EMAIL'],
    credentials['PATH_TO_PEM_FILE'],
    datacenter=credentials['DATA_CENTER'],
    project=credentials['PROJECT_ID'])

pprint(connection.__dict__)

```

9.7.5.2 Invoking services

In this section we will look into how we can use the connection created as previously instructed to perform various services such as creating nodes, listing nodes, starting nodes and stopping nodes.

Appropriate authentication code as described in the previous section is assumed.

This will give us an variable named conn which we will use for invoking Services. It is in the next sections not explicitly listed. It is indicated by our ... at the beginning

9.7.5.2.1 Creating Nodes

In this section we will look at the code that can be used to create a node in the provider a node which represents a virtual server

```
...
# retrieve available images and sizes
images = conn.list_images()

sizes = conn.list_sizes()

# create node with first image and first size
node = conn.create_node(
    name='yourservername',
    image=images[0],
    size=sizes[0])
```

9.7.5.2.2 Listing Nodes

In this section we will look at the code that can be used to list the nodes that have been created in the provider

```
...
nodes = connection.list_nodes()
print (nodes)
```

9.7.5.2.3 Starting Nodes

After the node (Virtual server) has been created the following code can be used to start the node

```
...
nodes = connection.list_nodes()
node = [n for n in nodes if 'yourservername' in n.name][0]
connection.ex_start(node=node)
```

9.7.5.2.4 Stopping Nodes

When needed the following command can be used to stop a node that has been started

```
...
nodes = connection.list_nodes()
node = [n for n in nodes if 'yourservername' in n.name][0]
connection.ex_stop(node=node)
```

9.7.6 Cloudmesh Community Program to Manage Clouds

As you have noticed since the authentication can change from cloud services to cloud service it would be much easier to use a simple python script to automatically handle the differences in the code.

We have provided such a python script which you can leverage to manage different cloud providers. You can find the python script and the corresponding .yaml file in the cloudmesh-community github repository.

- Python Script - <https://github.com/cloudmesh-community/cm/blob/master/cm.py>
- Yaml File - <https://github.com/cloudmesh-community/cm/blob/master/cloudmesh.yaml>

When using the script and yaml file please keep in mind the following steps to make sure you do not share your private keys and passwords on your publicly accessible Github account.

1. Create a folder in your computer that is not within a git clone that you have made. For example maybe you can use a new directory on your desktop
2. Copy the cm.py and cloudmesh.yaml files into this folder. Just to make sure you are not working with the files under the git repo you should delete the cloudmesh.yaml file in that is in your local git repo.
3. change the needed fields in the yaml file and use the python script to access the cloud services using libcloud.

To illustrate how simple the program is and that it significantly improves your management of credentials we provide the following code:

NOTE: This is to be implemented by you

```
from cm import cloudmesh

cm = cloudmesh()
cm.config()
driver = cm.get_driver("aws")
print("driver=", driver)
```

To switch to a different cloud, you just have to create it in the yaml file and use

that name.

It will be your task to add more providers to it.

We intent to host the code sometime soon on pypi so you can issue the command

```
$ pip install cm-community
```

and this library will be installed for you.

9.7.7 Amazon Simple Storage Service S3 via libcloud



No

Next we explain how to use Amazon Web Services (AWS) S3 via libcloud. Apache libcloud is a python library that provides abstraction layer and hides the complexities of directly integrating with AWS API's, for that matter it allows you to do so for different cloud providers. In the next sections more detailed steps are shown to install and use libcloud for AWS S3.

9.7.7.1 Access key

To be able to access AWS S3 from libcloud we need the access key to be specified in the call. Access key can be setup on AWS console by navigating to

`My Security credentials->Encryption Keys->Access Keys.`

9.7.7.2 Create a new bucket on AWS S3

In S3 you first need to create a bucket which is nothing but a container where you store your data in the form of files. This is where you can also define access controls.

- Click on S3 link on the AWS console under storage section, this will bring you to the create bucket window.
- Click on “Create Bucket” button, this opens up a wizard.
- Answer all mandatory questions on each page.
- Important point here is to note the “Bucket Name” and the “Region” you

are creating this bucket in, as this information will be used while calling the API.

9.7.7.3 List Containers

List Containers function list all the containers of buckets available for the user in that particular region.

 TODO change this example to use the cloudmesh.yaml file

```
from libcloud.storage.types import Provider
from libcloud.storage.providers import get_driver

cls = get_driver(Provider.S3_US_EAST2)
driver = cls('api key', 'api secret key')

d = driver.list_containers()

print (d)
```

9.7.7.4 List container objects

List container objects function shows the list of all objects in that container. Please note the output could be large depending on the files present in the bucket.

 TODO change this example to use the cloudmesh.yaml file

```
from libcloud.storage.types import Provider
from libcloud.storage.providers import get_driver

# Note I have used S3_US_EAST2 as this is the
# "region" where my S3 bucket is located.

cls = get_driver(Provider.S3_US_EAST2)
driver = cls('api key', 'api secret key')

container = driver.get_container(
    container_name='<bucket name>')

d = driver.list_container_objects(container)

print(d)
```

9.7.7.5 Upload a file

Upload a file helps in uploading a local file to S3 bucket.



TODO change this example to use the cloudmesh.yaml file

```

from libcloud.storage.types import Provider
from libcloud.storage.providers import get_driver

FILE_PATH = '<file path>/<filename>'

# Note I have used S3_US_EAST2 as this is
# the "region" where my S3 bucket is located.

cls = get_driver(Provider.S3_US_EAST2)
driver = cls('api key', 'api secret key')

container = driver.get_container(
    container_name='<bucket name>')

extra = {
    'meta_data': {
        'owner': '<owner name>',
        'created': '2018-03-24'
    }
}

with open(FILE_PATH, 'rb') as iterator:
    obj = driver.upload_object_via_stream(
        iterator=iterator,
        container=container,
        object_name='backup.tar.gz',
        extra=extra)

```

9.7.7.6 References

- <https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html>
- Documentation about libcloud can be found at <https://libcloud.readthedocs.org>
 - storage driver http://libcloud.readthedocs.io/en/latest/_modules/libcloud/storage/driver
 - Examples: <https://libcloud.readthedocs.io/en/latest/storage/examples.html>
 - API docs <http://libcloud.apache.org/apidocs/0.6.1/libcloud.storage.base.StorageD>

9.8 AWS BOTO



Construction

 This section has some features missing and does not yet use cloudmesh v4

Boto is a software development kit (SDK) that provides AWS interface for Python applications. It enables to write applications in Python that make use of Amazon Web Services.

Boto supports different AWS services such as, Elastic Compute Cloud (EC2), DynamoDB, AWS Config, CloudWatch and Simple Storage Service (S3).

In contrast to libcloud it only focusses to support AWS.

9.8.1 Boto versions

The current version of Boto is Boto3 and is available from:

- <https://github.com/boto/boto3>

The documentation from amazon is provided here:

- <http://aws.amazon.com/sdk-for-python>

It supports Python versions 2.6.5, 2.7 and 3.3+.

9.8.2 Boto Installation

To install boto with its latest release, use

```
$ pip install boto3
```

To install boto from source, use

```
$ git clone https://github.com/boto/boto3.git  
$ cd boto3
```

Before you install it we suggest that you either use pyenv or venv.

```
$ python setup.py install
```

To install additional modules to use boto.cloudsearch, boto.manage, boto.mashups and to get all modules required for test suite, than the run command

```
$ python setup.py install
```

9.8.3 Access key

An initial setup is required to be able to access AWS EC2 from BOTO wherein you provide the key and region details. You can find the key details from IAM console on AWS.

9.8.4 Boto configuration

BOTO can be configured in two ways, either by using the `aws configure` command if you have AWS Command line interface installed or simply by manually creating and editing the `~/.aws/credentials` file to include important parameters highlighted next.

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY>
aws_secret_access_key = <YOUR_SECRET_KEY>
```

Similar to libcloud, BOTO also requires the region where you would create your EC2 instance, the same can be maintained by creating a config file.

```
$ emacs ~/.aws/config
[default]
region=<region name> # for example us-east
```

9.8.5 Boto configuration with cloudmesh



please also document here how to use the `cloudmesh.yaml` file

9.8.6 EC2 interface of Boto

9.8.6.0.1 Create connection

To access EC2 instance, first import the required package.

```
import boto3.ec2
```

Make a connection to from application by specifying AWS region in which the user account is created, aws access key and secret key. AWS provides access key and secret key when a new user is created. Access key and secret key helps to identify the user.



TODO use the cloudmesh config file here

```
connection = boto3.ec2.connect_to_region(  
    '<region name>',  
    aws_access_key_id='<access key>',  
    aws_secret_access_key='<secret key>')
```

connection object now points to EC2Connection object returned by the function

`connect_to_region`.

9.8.7 List EC2 instances

The code to list the running instances (if you have some) is very simple:

```
import boto3  
  
ec2 = boto3.client('ec2')  
response = ec2.describe_instances()  
print(response)
```

9.8.7.0.1 Launch a new instance

To launch a new instance with default properties

```
connection.run_instances('<ami-id>')
```

Additional parameters can be specified to create instance of specific type and security group.

```
connection.run_instances('<ami-id>', key_name='<key>', instance_type='<type>',  
security_groups=['<security group list>'])
```

Instance type specifies the storage and type of platform. Security groups are required to provide access rights such as access to SSH into the instance.

9.8.7.0.2 Check running instances

The `get_all_reservations` function of EC2Connection object will return list of running instances.

```
reservations = connection.get_all_reservations()  
instances = reservations[0].instances
```

9.8.7.0.3 Stop instance

Up and running instances can be stopped. The `stop_instances` function of connection

object enables multiple instances to be stopped in one command.

```
connection.stop_instances(instance_ids=['<id1>', '<id2>', ...])
```

9.8.7.0.4 Terminate instance

To terminate one or more instances simultaneously, use the `terminate_instances` function.

```
connection.terminate_instances(instance_ids=['<id1>', '<id2>', ..])
```

9.8.7.1 Reboot instances

The next example showcases how to reboot an instance, which is copied from <http://boto3.readthedocs.io/en/latest/guide/ec2-example-managing-instances.html>

```
# Code copied from
# http://boto3.readthedocs.io/en/latest/guide/ec2-example-managing-instances.html
import boto3
from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

try:
    ec2.reboot_instances(InstanceIds=['INSTANCE_ID'], DryRun=True)
except ClientError as e:
    if 'DryRunOperation' not in str(e):
        print("You don't have permission to reboot instances.")
        raise
try:
    response = ec2.reboot_instances(InstanceIds=['INSTANCE_ID'], DryRun=False)
    print('Success', response)
except ClientError as e:
    print('Error', e)
```

9.8.8 Amazon S3 interface of Boto

9.8.8.0.1 Create connection

Import required packages

```
import boto3.s3
from boto3.s3.key import Key
```

Create a connection

```
connection = boto.connect_s3('<access-key>', '<secret-key>')
```

9.8.8.0.2 Create new bucket in S3

Amazon S3 stores all its data in Bucket. There is no limitation specified by AWS about number of data files allowed per bucket.

Bucket name has to be unique name accross all the AWS regions and hence globally unique.

```
bucket = conn.create_bucket('<bucket_name>')
```

If bucket name is unique, a new bucket of specified name will get created. If bucket name is not unique, application will throw error as

```
boto.exception.S3CreateError: S3Error[409]: Conflict
```

9.8.8.0.3 Upload data

To upload a file in the S3 bucket, first create a key object from `new_key()` function of bucket.

```
key = bucket.new_key('hello2.txt')
key.set_contents_from_string('Hello World!')
```

This will create hello.txt file with content Hello World! in the text file. This file can be found inside the bucket in which new key is created.

9.8.8.0.4 List all buckets

One account can have maximum 100 buckets in which data objects can be stored.

```
result = connection.get_all_buckets()
```

The `get_all_buckets` function of `S3Connection` lists all the buckets within account. It returns `ResultSet` object which has list of all buckets.

9.8.8.0.5 List all objects in a bucket

Data objects stored in a bucket has a metadata associated with it such as `LastModified` date and time. This information can also be captured.

```
# To list files in selected bucket
for key in bucket.list():
    print ("{name}".format(**key))
    print ("{size}".format(**key))
    print ("{last_modified}".format(**key))
```

9.8.8.0.6 Delete object

To delete any data object from bucket, `delete_key` function of bucket is used.

```
k = Key(<bucket-name>, <file-name>)  
k.delete()
```

9.8.8.0.7 Delete bucket

To delete a bucket, provide a bucket name and call the `delete_bucket` function of `S3Connection` object.

```
connection.delete_bucket('<bucket-name>')
```

9.8.9 References

- <https://github.com/boto/boto3>
- <https://boto3.readthedocs.io/en/latest/guide/quickstart.html#installation>
- <http://boto3.readthedocs.io/en/latest/guide/ec2-example-managing-instances.html>

9.8.10 Excercises

E.boto.cloudmesh.1:

will will nw create a cloudmesh tool that manages virtual machines on the commandline. For that we copy the code published at

- <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/ec2-example-managing-instances.html>.

Modify this code using docopts and look at samples in

- <https://github.com/cloudmesh-community/cm>

where we use libcloud. The code from Amazon is.

```
import sys  
import boto3  
from botocore.exceptions import ClientError  
  
instance_id = sys.argv[2]  
action = sys.argv[1].upper()
```

```

ec2 = boto3.client('ec2')

if action == 'ON':
    # Do a dryrun first to verify permissions
    try:
        ec2.start_instances(InstanceIds=[instance_id], DryRun=True)
    except ClientError as e:
        if 'DryRunOperation' not in str(e):
            raise
    # Dry run succeeded, run start_instances without dryrun
    try:
        response = ec2.start_instances(InstanceIds=[instance_id], DryRun=False)
        print(response)
    except ClientError as e:
        print(e)
else:
    # Do a dryrun first to verify permissions
    try:
        ec2.stop_instances(InstanceIds=[instance_id], DryRun=True)
    except ClientError as e:
        if 'DryRunOperation' not in str(e):
            raise
    # Dry run succeeded, call stop_instances without dryrun
    try:
        response = ec2.stop_instances(InstanceIds=[instance_id], DryRun=False)
        print(response)
    except ClientError as e:
        print(e)

```

E.boto.cloudmesh.2:

Integrate, start, stop, reboot, and other useful functions

E.boto.cloudmesh.3:

Discuss the advantages of docopts.

10 MAPREDUCE

10.1 INTRODUCTION TO MAPREDUCE

In this section we discuss about the background of Mapreduce along with Hadoop and core components of Hadoop.

We start out our section with a review of the python lambda expression as well as the map function. Understanding these concepts is helpful for our overall understanding of map reduce.

So before you watch the video, we encourage you to learn Sections {#s-python-lambda} and {#s-python-map}.

Now that you have a basic understanding of the map function we recommend to watch our videos about mapreduce, hadoop and spark which we provide within this chapter.



[Map Reduce, Hadoop, and Spark \(19:02\) Hadoop A](#)

MapReduce is a programming technique or processing capability which operates in a cluster or a grid on a massive data set and brings out reliable output. It works on essentially two main functions – map() and reduce(). MapReduce processes large chunks of data so its highly beneficial to operate in multi-threaded fashion meaning parallel processing. MapReduce can also take advantage of data locality so that we do not loose much on communication of data from place to another.

10.1.1 MapReduce Algorithm

MapReduce can operate on a filesystem, which is an unstructured data or a database, a structured data and these are the following three stages of its operation (see [Figure 88](#)):

1. **Map:** This method processes the very initial data set. Generally, the data is in file format which can be stored in HDFS (Hadoop File System). Map

function reads the data line by line and creates several chunks of data and that is again stored in HDFS. This broken set of data is in key/value pairs. So in multi-threaded environment, there will be many worker nodes operating on the data using this map() function and write this intermediate data in form of key/value to temporary data storage.

2. **Shuffle:** In this stage, worker nodes will shuffle or redistribute the data in such a way that there is only one copy for each key.
3. **Reduce:** This function always comes at last and it works on the data produced by map and shuffle stages and produces even smaller chunk of data which is used to calculate output.

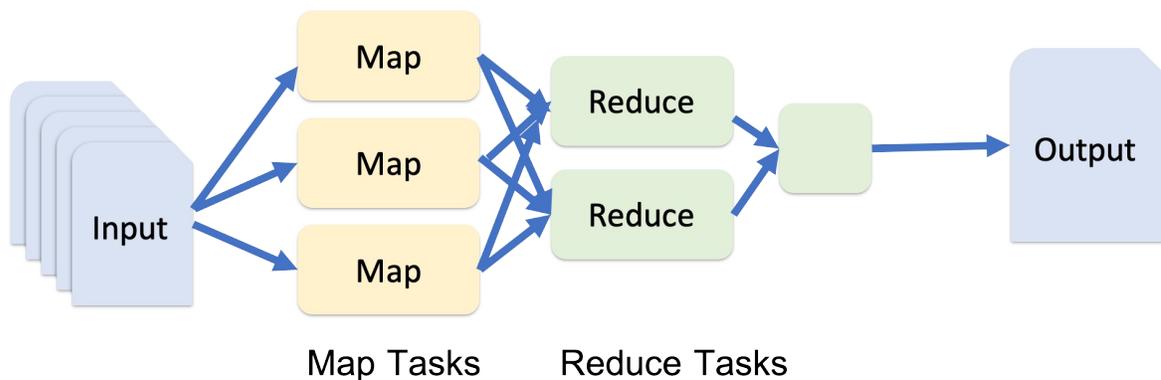


Figure 88: MapReduce Conceptual diagram

The Shuffle operation is very important here as that is mainly responsible for reducing the communication cost. The main advantage of using MapReduce algorithm is that it becomes very easy to scale up data processing just by adding some extra computing nodes. Building up map and reduce methods are sometimes nontrivial but once done, scaling up the applications is so easy that it is just a matter of changing configuration. Scalability is really big advantage of MapReduce model. In the traditional way of data processing, data was moved from nodes to the master and then the processing happens in master machine. In this approach, we lose bandwidth and time on moving data to master and parallel operation cannot happen. Also master can get over-burdened and fail. In MapReduce approach, Master node distributes the data to the worker machines which are in themselves a processing unit. So all worker process the data in parallel and the time taken to process the data is reduced tremendously. (see [Figure 89](#))

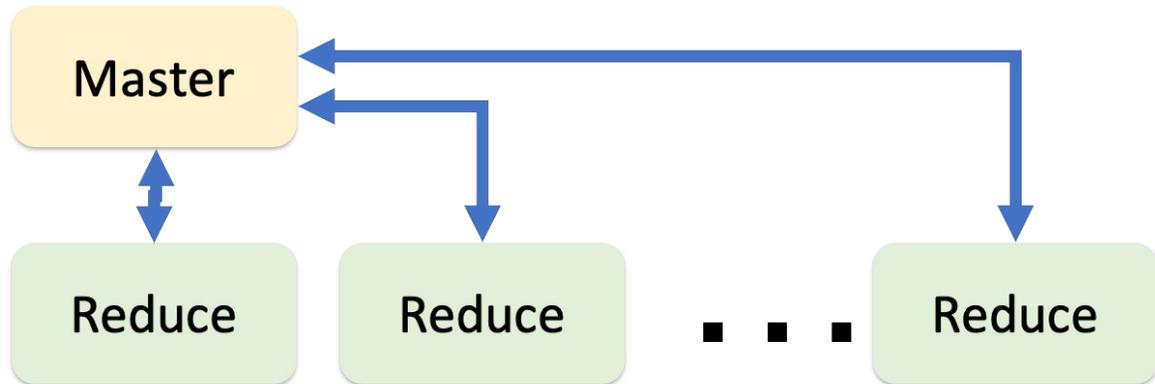


Figure 89: MapReduce Master worker diagram

10.1.1.1 MapReduce Example: Word Count

Let us understand MapReduce by an example. For example: we have a text file as Sample.txt as Cat, Bear, Camel, Bird, Cat, Bird, Camel, Cat, Bear, Camel, Cat, Camel

1. First we divide the input into four parts so that individual nodes can handle the load.
2. We tokenize each word and assign weightage of value “1” to each word.
3. This way we will have a list of key-value pairs with key being the word and value as 1.
4. After this mapping phase, shuffling phase starts where all maps with same key are sent corresponding reducer.
5. Now each reducer will have a unique key and a list of values for each key which in this case is all 1s.
6. After that, each reducer will count the total number of 1s and assigns final count to each word.
7. The final output is then written to a file. (see [Figure 90](#))

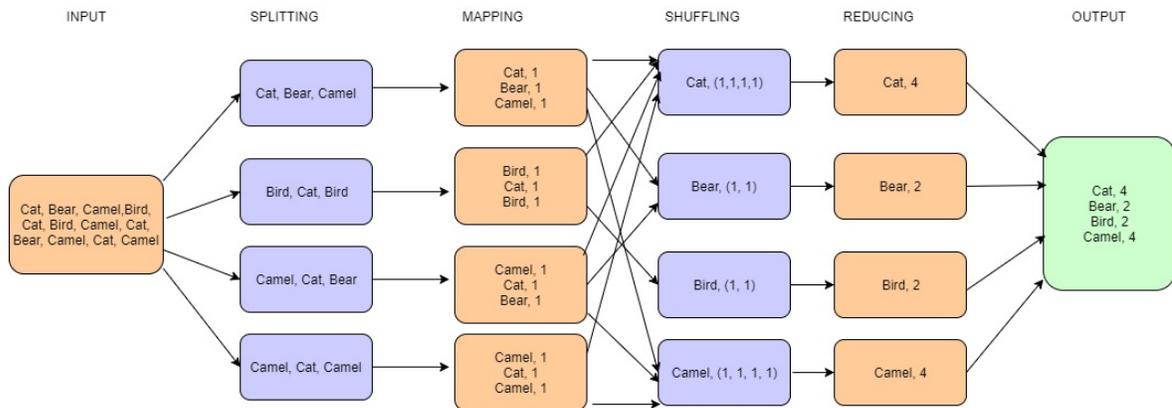


Figure 90: MapReduce WordCount [68]

Let us see an example of map() and reduce() methods in code for this word count example.

```
public static class Map extends Mapper<LongWritable,
    Text,
    Text,
    IntWritable> {

    public void map(LongWritable key,
        Text value,
        Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            value.set(tokenizer.nextToken());
            context.write(value, new IntWritable(1));
        }
    }
}
```

Here we have created a class Map which extends Mapper from MapReduce framework and we override map() method to declare the key/value pairs. Next, there will be a reduce method defined inside Reduce class as next and both input and output here is a key/value pairs:

```
public static class Reduce extends Reducer<Text,
    IntWritable,
    Text, IntWritable> {

    public void reduce(Text key,
        Iterable<IntWritable> values,
        Context context)
        throws IOException, InterruptedException {

        int sum=0;
        for(IntWritable x: values) {
            sum+=x.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

10.1.2 Hadoop MapReduce and Hadoop Spark

In earlier version of Hadoop, we could use MapReduce with HDFS directly but from 2.0 onwards, YARN(Cluster Resource Management) is introduced which acts as a layer between MapReduce and HDFS and using this YARN, many other BigData frameworks can connect to HDFS as well. (see [Figure 91](#))

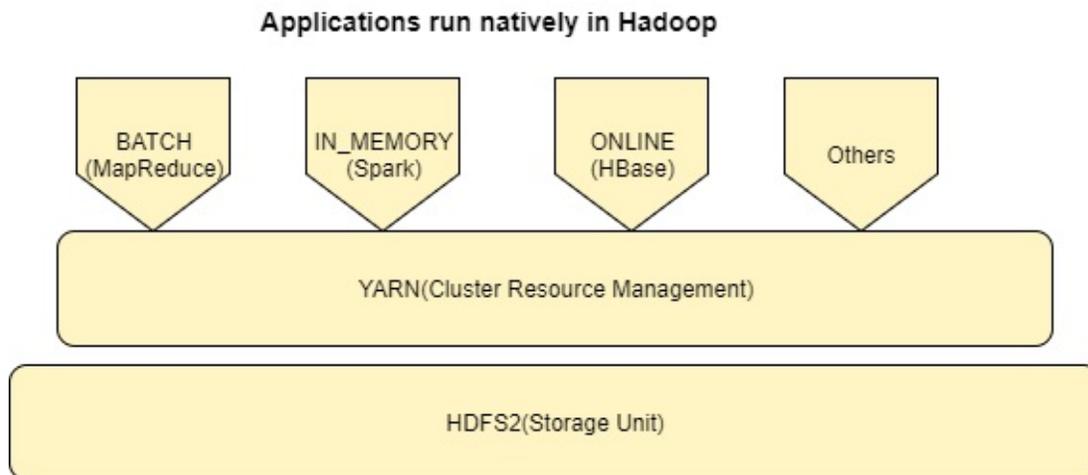


Figure 91: MapReduce Hadoop and Spark [69]

There are many big data frameworks available and there is always a question as to which one is the right one. Leading frameworks are Hadoop MapReduce and Apache Spark and choice depends on business needs. Let us start comparing both of these frameworks with respect to their processing capability.

10.1.2.1 Apache Spark

Apache Spark is lightning fast cluster computing framework. Spark is in-memory system. Spark is 100 time faster than Hadoop MapReduce.

10.1.2.2 Hadoop MapReduce

Hadoop MapReduce reads and writes on disk because of this it is a slow system and that affects the volume of data been processed. But Hadoop is a scalable and fault tolerant, it us good for linear processing.

10.1.2.3 Key Differences

The key differences between them are as follows:

1. **Speed:** Spark is lightning fast cluster computing framework and operates up to 100 times faster in-memory and 10 times faster than Hadoop on disk. In-memory processing reduces the disk read/write processes which are time consuming.
2. **Complexity:** Spark is easy to use since there are many APIs available but for Hadoop, developers need to code the functions which makes it harder.
3. **Application Management:** Spark can perform batch processing, interactive and Machine Learning and Streaming of data, all in the same cluster, which makes it a complete framework for data analysis whereas Hadoop is just a batch engine and it requires other frameworks for other tasks which makes it somewhat difficult to manage.
4. **Real-Time Data Analysis** Spark is capable of processing real time data with great efficiency. But Hadoop was designed primarily for batch processing so it cannot live data.
5. **Fault Tolerance:** Both the systems are fault tolerant so there is no need to restart the applications from scratch.
6. **Data Volume:** As the data for spark is held in memory larger data volumes are better managed in Hadoop.

10.1.3 References

- [70] <https://www.ibm.com/analytics/hadoop/mapreduce>
- [71] <https://en.wikipedia.org/wiki/MapReduce>
- [72] https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
- [68] https://www.edureka.co/blog/mapreduce-tutorial/?utm_source=youtube&utm_campaign=mapreduce-tutorial-161216-wr&utm_medium=description
- [73] <https://www.quora.com/What-is-the-difference-between-Hadoop-and-Spark>
- [74] <https://data-flair.training/blogs/apache-spark-vs-hadoop-mapreduce>
- [69] <https://www.youtube.com/watch?v=SqvAaB3vK8U&list=WL&index=25&t=2547s>

10.2 HADOOP

10.2.1 Hadoop

Hadoop is an open source framework for storage and processing of large datasets on commodity clusters. Hadoop internally uses its own file system called HDFS (Hadoop Distributed File System).

The motivation for Hadoop was introduced in Section Mapreduce

10.2.1.1 Hadoop and MapReduce

In this section we discuss about the usage Hadoop MapReduce architecture.



[Hadoop 13:19 Hadoop B](#)

10.2.1.2 Hadoop EcoSystem

In this section we discuss about the Hadoop EcoSystem and the architecture.



[Hadoop 12:57 Hadoop C](#)

10.2.1.3 Hadoop Components

In this section we discuss about Hadoop Components in detail.



[Hadoop 15:14 Hadoop D](#)

10.2.1.4 Hadoop and the Yarn Resource Manager

In this section we discuss about Yarn resource manager and novel components added to the Hadoop framework in case of improving the performance and minimizing fault tolerance.



[Hadoop 14:55 Hadoop E](#)

10.2.1.5 PageRank

In this section we discuss about a real world problem that can be solved using

the MapReduce technique. PageRank is a problem solved by the earliest stages of the Google.inc. In this section we discuss about the theoretical background about this problem and we discuss how this can be solved using the map reduce concepts.



[Hadoop 25:41 Hadoop F](#)

10.2.2 Installation of Hadoop

This section is using Hadoop version 3.1.1 in Ubuntu 18.04. We also describe the installation of the Yarn resource manager. We assume that you have ssh, and rsync installed and use emacs as editor.

If you use a newer version, and like to update this text please help

10.2.2.1 Releases

Hadoop changes on regular basis. Before following this section, we recommend that you visit

- <https://hadoop.apache.org/releases.html>

The list of downloadable files is also available at

and verify that you use an up to date version. If the version of this installation is outdated, we ask you as exercise to update it.

10.2.2.2 Prerequisites

```
sudo apt-get install ssh
sudo apt-get install rsync
sudo apt-get install emacs
```

10.2.2.3 User and User Group Creation

For security reasons we will install hadoop in a particular user and user group. We will use the following

```
sudo addgroup hadoop_group
sudo adduser --ingroup hadoop_group hduser
```

```
sudo adduser hduser sudo
```

These steps will provide sudo privileges to the created hduser user and add the user to the group `hadoop_group`.

10.2.2.4 Configuring SSH

Here we configure SSH key for the local user to install hadoop with a ssh-key. This is different from the ssh-key you used for Github, FutureSystems, etc. Follow this section to configure it for Hadoop installation.

The ssh content is included here because, we are making a ssh key for this specific user. Next, we have to configure ssh to be used by the hadoop user.

```
sudo su - hduser
ssh-keygen -t rsa
```

Follow the instructions as provided in the commandline. When you see the following console input, press ENTER. Here only we will create password less keys. IN general this is not a good idea, but for this case we make an exception.

```
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
```

Next you will be asked to enter a password for ssh configuration,

```
Enter passphrase (empty for no passphrase):
```

Here enter the same password

```
Enter same passphrase again:
```

Finally you will see something like this after these steps are finished.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0UBCPd6oYp7MEZCp0hMhNiJyQo6PaPCDu0T48xUDDc0 hduser@computer
The key's randomart image is:
+---[RSA 2048]-----+
| .+000                |
| .  oE.oo            |
|+ .. ...+.          |
|X+= . 0..           |
|XX.o  o.S           |
|BO+ + .o            |
|*o * +.             |
|*.. *               |
+---+-----+

```

```
| +.o.. |  
+----[SHA256]-----+
```

You have successfully configured ssh.

10.2.2.5 Installation of Java

If you are already logged into su, you can skip the next command:

```
su - hduser
```

Now execute the following commands to download and install java

```
mkdir -p ~/cloudmesh/bin  
cd ~/cloudmesh/bin  
wget -c --header "Cookie: \  
oraclelicense=accept-securebackup-cookie" \  
"http://download.oracle.com/otn-pub/java/jdk/8u191-b12/2787e4a523244c269598db4e85c51e0c/jdk-8u191-linux-x64.tar.gz"  
tar xvzf jdk-8u191-linux-x64.tar.gz
```

Please note that users must accept Oracle OTN license before downloading JDK.

10.2.2.6 Installation of Hadoop

First we will take a look on how to install Hadoop 3.1.1 on Ubuntu

16.04. We may need a prior folder structure to do the installation properly.

```
cd ~/cloudmesh/bin/  
wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-3.1.1/hadoop-3.1.1.tar.gz  
tar -xvzf hadoop-3.1.1.tar.gz
```

10.2.2.7 Hadoop Environment Variables

In Ubuntu the environmental variables are setup in a file called bashrc at it can be accessed the following way

```
emacs ~/.bashrc
```

Now add the following to your `~/.bashrc` file

```
export JAVA_HOME=~/cloudmesh/bin/jdk1.8.0_191  
export HADOOP_HOME=~/cloudmesh/bin/hadoop-3.1.1  
export YARN_HOME=$HADOOP_HOME  
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop  
export PATH=$HADOOP_HOME/bin:$JAVA_HOME/bin:$PATH
```

In Emacs to save the file `ctrl-x-s` and `ctrl-x-c` to exit. After editing you must update

the variables in the system.

```
source ~/.bashrc
java -version
```

If you have installed things properly there will be no errors. It will show the version as follows,

```
java version "1.8.0_191"
Java(TM) SE Runtime Environment (build 1.8.0_191-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
```

And verifying the hadoop installation,

```
hadoop
```

If you have successfully installed this, there must be a message shown as next.

```
Usage: hadoop [--config confdir] COMMAND
      where COMMAND is one of:
      fs                run a generic filesystem user client
      version           print the version
      jar <jar>         run a jar file
      checknative [-a|-h] check native hadoop and compression libraries availability
      distcp <srcurl> <desturl> copy file or directories recursively
      archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
      classpath         prints the class path needed to get the
      credential        interact with credential providers
                        Hadoop jar and the required libraries
      daemonlog         get/set the log level for each daemon
      trace             view and modify Hadoop tracing settings
      or
      CLASSNAME         run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
```

10.2.3 Hadoop Distributed File System (Hadoop HDFS)

10.2.3.1 Introduction

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware ([75](#)). It has many similarities with existing distributed file systems. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. The project URL is <https://hadoop.apache.org/hdfs/>.

10.2.3.2 Features

- Detect Hardware Failure Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.
- Support Streaming Data Access Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users.
- Support Large Data Sets Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

10.2.3.3 HDFS Components

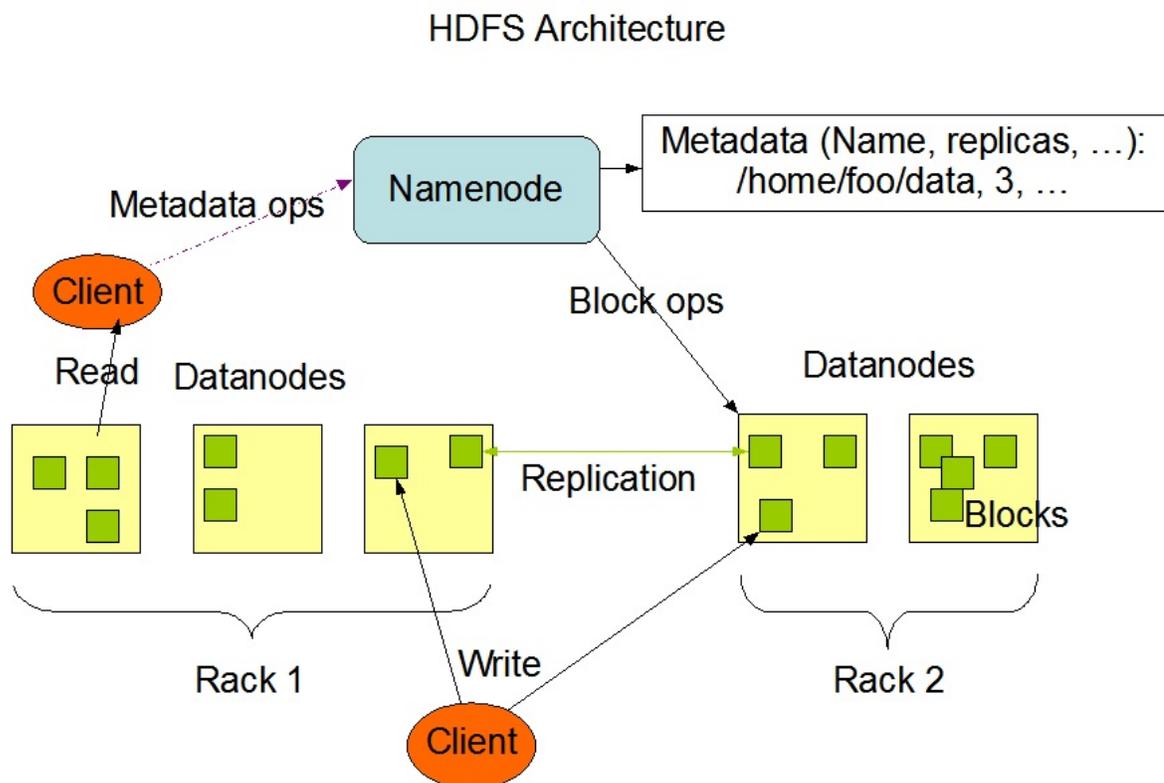


Figure 92: Hadoop HDFS [75]

10.2.3.3.1 NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients (see [Figure 92](#)). In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files.

Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata.

10.2.3.4 Usage

10.2.3.4.1 Java Client API

HDFS can be accessed from applications in many different ways. Natively, HDFS provides a Java API for applications to use. A C language wrapper for this Java API is also available. In addition, an HTTP browser can also be used to browse the files of an HDFS instance. Work is in progress to expose HDFS through the WebDAV protocol.

Here is an example of Java code to read and write files on HDFS:

```
// ===== Init HDFS File System Object
Configuration conf = new Configuration();
// Set FileSystem URI
conf.set("fs.defaultFS", hdfsuri);
// Because of Maven
conf.set("fs.hdfs.impl", org.apache.hadoop.hdfs.DistributedFileSystem.class.getName());
conf.set("fs.file.impl", org.apache.hadoop.fs.LocalFileSystem.class.getName());
// Set HADOOP user
System.setProperty("HADOOP_USER_NAME", "hdfs");
System.setProperty("hadoop.home.dir", "/");
//Get the filesystem - HDFS
FileSystem fs = FileSystem.get(URI.create(hdfsuri), conf);
```

```
//==== Read file
logger.info("Read file from hdfs");
//Create a path
Path hdfsreadpath = new Path(newFolderPath + "/" + fileName);
//Init input stream
FSDataInputStream inputStream = fs.open(hdfsreadpath);
//Classical input stream usage
String out= IOUtils.toString(inputStream, "UTF-8");
logger.info(out);
inputStream.close();
fs.close();
```

```
//==== Write file
logger.info("Begin Write file into hdfs");
//Create a path
Path hdfswritepath = new Path(newFolderPath + "/" + fileName);
//Init output stream
FSDataOutputStream outputStream=fs.create(hdfswritepath);
//Classical output stream usage
outputStream.writeBytes(fileContent);
outputStream.close();
logger.info("End Write file into hdfs");
```

10.2.3.4.2 FS Shell

HDFS allows user data to be organized in the form of files and directories. It provides a commandline interface called FS shell that lets a user interact with the data in HDFS. The syntax of this command set is similar to other shells (e.g. bash, csh) that users are already familiar with. Here are some sample action/command pairs:

```
bin/hadoop dfs -mkdir /foodir
bin/hadoop dfs -rmdir /foodir
bin/hadoop dfs -cat /foodir/myfile.txt
```

10.2.3.5 References

- HDFS Java API: <https://hadoop.apache.org/core/docs/current/api/>
- HDFS source code: https://hadoop.apache.org/hdfs/version_control.html

10.2.3.6 Exercises

- Hadoop Installation on your own Laptop/Desktop
- Hadoop MapReduce programming in Python
- Hadoop Installation on a cluster (at least two nodes in the cluster)
- Run MapReduce in a cluster

10.2.4 Apache HBase

10.2.4.1 Introduction

Apache HBase is the Hadoop database, a distributed, scalable, big data store.

Use Apache HBase when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables – billions of rows X millions of columns – atop clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

10.2.4.2 Features

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
- Easy to use Java API for client access.
- Block cache and Bloom Filters for real-time queries.
- Query predicate push down via server side Filters
- Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
- Extensible jruby-based (JIRB) shell
- Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX

10.2.4.3 Configuration

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///home/testuser/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/testuser/zookeeper</value>
  </property>
  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>>false</value>
    <description>
      Controls whether HBase will check for stream capabilities (hflush/hsync).

      Disable this if you intend to run on LocalFileSystem, denoted by a rootdir
      with the 'file:/' scheme, but be mindful of the NOTE below.

      WARNING: Setting this to false blinds you to potential data loss and
      inconsistent system state in the event of process and/or node failures. If
      HBase is complaining of an inability to use hsync or hflush it's most
      likely not a false positive.
    </description>
  </property>
</configuration>
```

10.2.4.4 Usage

10.2.4.4.1 Connect to HBase.

Connect to your running instance of HBase using the hbase shell command, located in the bin/ directory of your HBase install. In this example, some usage and version information that is printed when you start HBase Shell has been omitted. The HBase Shell prompt ends with a > character.

```
$ ./bin/hbase shell
hbase(main):001:0>
```

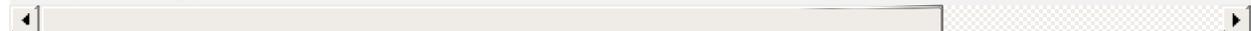
10.2.4.4.2 Create a table

```
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 0.4170 seconds

=> Hbase::Table - test
```

10.2.4.4.3 Describe a table

```
describe 'test'
Table test is ENABLED
test
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS =>
'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER =>
'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE',
=> '65536'}
1 row(s)
Took 0.9998 seconds
```



10.2.4.4 HBase MapReduce job

This example will simply copy data from one table to another.

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "ExampleReadWrite");
job.setJarByClass(MyReadWriteJob.class); // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500); // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false); // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
    sourceTable, // input table
    scan, // Scan instance to control CF and attribute selection
    MyMapper.class, // mapper class
    null, // mapper output key
    null, // mapper output value
    job);
TableMapReduceUtil.initTableReducerJob(
    targetTable, // output table
    null, // reducer class
    job);
job.setNumReduceTasks(0);

boolean b = job.waitForCompletion(true);
if (!b) {
    throw new IOException("error with job!");
}
```

10.2.4.5 References

- HBase API: <https://hbase.apache.org/book.html>

10.2.5 Hadoop Virtual Cluster Installation Using Cloudmesh



No



This version is dependent on an older version of cloudmesh.



:TODO: we need to add the installation instructions based on this version

10.2.5.1 Cloudmesh Cluster Installation

Before you start this lesson, you MUST finish cm_install.

This lesson is created and test under the newest version of Cloudmesh client.

Update yours if not.

To manage virtual cluster on cloud, the command is `cm cluster`. Try `cm cluster help` to see what other commands are and what options they supported.

10.2.5.1.1 Create Cluster

To create a virtual cluster on cloud, we must define an active cluster specification with `cm cluster define` command. For example, we define a cluster with 3 nodes:

```
$ cm cluster define --count 3
```

All options will use the default setting if not specified during cluster

define. Try `cm cluster help` command to see what options `cm cluster define` has and means, here is part of the usage information: :

```
$ cm cluster help
```

```
usage: cluster create [-n NAME] [-c COUNT] [-C CLOUD] [-u NAME] [-i IMAGE] [-f FLAVOR] [-k KEY] [-s NAME] [-AI]
```

Options:

```
-A --no-activate Do not activate this cluster
-I --no-floating-ip Do not assign floating IPs
-n NAME --name=NAME Name of the cluster
-c COUNT --count=COUNT Number of nodes in the cluster
-C NAME --cloud=NAME Name of the cloud
-u NAME --username=NAME Name of the image login user
-i NAME --image=NAME Name of the image
-f NAME --flavor=NAME Name of the flavor
-k NAME --key=NAME Name of the key
-s NAME --secgroup=NAME NAME of the security group
-o PATH --path=PATH Output to this path ...
```

Floating IP is a valuable and limited resource on cloud.

`cm cluster define` will assign floating IP to every node within the cluster by default. Cluster creation will fail if the floating IPs run out on cloud. When you run into error like this, use option `-I` or `--no-floating-ip` to avoid assigning floating IPs during cluster creation:

```
$ cm cluster define --count 3 --no-floating-ip
```

Then manually assign floating IP to one of the nodes. Use this node as a logging node or head node to log in to all the other nodes.

We can have multiple specifications defined at the same time. Every time a new cluster specification is defined, the counter of the default cluster name will increment. Hence, the default cluster name will be `cluster-001`, `cluster-002`, `cluster-003` and so on. Use `cm cluster avail` to check all the available cluster specifications:

```
$ cm cluster avail
cluster-001
  count           : 3
  image           : CC-Ubuntu14.04
  key             : xl41
  flavor          : m1.small
  secgroup        : default
  assignFloatingIP : True
  cloud           : chameleon
> cluster-002
  count           : 3
  image           : CC-Ubuntu14.04
  key             : xl41
  flavor          : m1.small
  secgroup        : default
  assignFloatingIP : False
  cloud           : chameleon
```

With `cm cluster use [NAME]`, we are able to switch between different specifications with given cluster name:

```
$ cm cluster use cluster-001
$ cm cluster avail
> cluster-001
  count           : 3
  image           : CC-Ubuntu14.04
  key             : xl41
  flavor          : m1.small
  secgroup        : default
  assignFloatingIP : True
  cloud           : chameleon
cluster-002
  count           : 3
  image           : CC-Ubuntu14.04
  key             : xl41
  flavor          : m1.small
  secgroup        : default
  assignFloatingIP : False
  cloud           : chameleon
```

This will activate specification `cluster-001` which assigns floating IP during creation rather than the latest one `cluster-002`.

With our cluster specification ready, we create the cluster with command `cm cluster allocate`. This will create a virtual cluster on the cloud with the activated specification:

```
$ cm cluster allocate
```

Each specification can have one active cluster, which means `cm cluster allocate` does nothing if there is a successfully active cluster.

10.2.5.1.2 Check Created Cluster

With command `cm cluster list`, we can see the cluster with the default name `cluster-001` we just created:

```
$ cm cluster list
cluster-001
```

Using `cm cluster nodes [NAME]`, we can also see the nodes of the cluster along with their assigned floating IPs of the cluster:

```
$ cm cluster nodes cluster-001
x141-001 129.114.33.147
x141-002 129.114.33.148
x141-003 129.114.33.149
```

If option `--no-floating-ip` is included during definition, you will see nodes without floating IP:

```
$ cm cluster nodes cluster-002
x141-004 None
x141-005 None
x141-006 None
```

To log in one of them, use command `cm vm assign IP [NAME]` to assign a floating IP to one of them:

```
$ cm vm ip assign x141-006
$ cm cluster nodes cluster-002
x141-004 None
x141-005 None
x141-006 129.114.33.150
```

Then you can log in this node as a head node of your cluster by `cm vm ssh [NAME]`:

```
$ cm vm ssh x141-006
cc@x141-006 $
```

10.2.5.1.3 Delete Cluster

Using `cm cluster delete [NAME]`, we are able to delete the cluster we created:

```
$ cm cluster delete cluster-001
```

Option `--all` can delete all the clusters created, so be careful:

:

```
$ cm cluster delete --all
```

Then we need to undefine our cluster specification with command `cm cluster undefine`

[NAME]:

```
$ cm cluster undefine cluster-001
```

Option `--all` can delete all the cluster specifications:

```
$ cm cluster undefine --all
```

10.2.5.2 Hadoop Cluster Installation

This section is built upon the previous one. Please finish the previous one before start this one.

10.2.5.2.1 Create Hadoop Cluster

To create a Hadoop cluster, we need to first define a cluster with `cm cluster define` command:

```
$ cm cluster define --count 3
```

To deploy a Hadoop cluster, we only support image `CC-Ubuntu14.04`

on Chameleon. DO NOT use `CC-Ubuntu16.04` or any other images. You will need to specify it if it's not the default image:

```
$ cm cluster define --count 3 --image CC-Ubuntu14.04
```

Then we define the Hadoop cluster upon the cluster we defined using `cm hadoop define` command:

```
$ cm hadoop define
```

Same as `cm cluster define`, you can define multiple specifications for the Hadoop cluster and check them with `cm hadoop avail`:

```
$ cm hadoop avail
> stack-001
  local_path      : /Users/tony/.cloudmesh/stacks/stack-001
  addons          : []
```

We can use `cm hadoop use [NAME]` to activate the specification with the given name:

```
$ cm hadoop use stack-001
```

May not be available for current version of Cloudmesh Client.

Before deploy, we need to use `cm hadoop sync` to checkout / synchronize the Big Data Stack from Github.com:

```
$ cm hadoop sync
```

To avoid errors, make sure you are able to connect to Github.com using SSH:

<https://help.github.com/articles/connecting-to-github-with-ssh/>.

Finally, we are ready to deploy our Hadoop cluster:

```
$ cm hadoop deploy
```

This process could take up to 10 minutes based on your network.

To check Hadoop is working or not. Use `cm vm ssh` to log into the `namenode` of the Hadoop cluster. It's usually the first node of the cluster:

```
$ cm vm ssh node-001  
cc@hostname$
```

Switch to user `hadoop` and check HDFS is set up or not:

```
cc@hostname$ sudo su - hadoop  
hadoop@hostname$ hdfs dfs -ls /  
Found 1 items  
drwxrwx--- - hadoop hadoop,hadoopadmin 0 2017-02-15 17:26 /tmp
```

Now the Hadoop cluster is properly installed and configured.

10.2.5.2.2 Delete Hadoop Cluster

To delete the Hadoop cluster we created, use command `cm cluster delete [NAME]` to delete the cluster with given name:

```
$ cm cluster delete cluster-001
```

Then undefine the Hadoop specification and the cluster specification:

```
$ cm hadoop undefine stack-001  
$ cm cluster undefine cluster-001
```

May not be available for current version of Cloudmesh Client.

10.2.5.3 Advanced Topics with Hadoop

10.2.5.3.1 Hadoop Virtual Cluster with Spark and/or Pig

To install Spark and/or Pig with Hadoop cluster, we first use command `cm hadoop define` but with `ADDON` to define the cluster specification.

For example, we create a 3-node Spark cluster with Pig. To do that, all we need is to specify `spark` as an `ADDON` during Hadoop definition:

```
$ cm cluster define --count 3
$ cm hadoop define spark pig
```

Using `cm hadoop addons`, we are able to check the current supported addon:

```
$ cm hadoop addons
```

With `cm hadoop avail`, we can see the detail of the specification for the Hadoop cluster:

```
$ cm hadoop avail
> stack-001
  local_path      : /Users/tony/.cloudmesh/stacks/stack-001
  addons          : [u'spark', u'pig']
```

Then we use `cm hadoop sync` and `cm hadoop deploy` to deploy our Spark cluster:

```
$ cm hadoop sync
$ cm hadoop deploy
```

This process will take 15 minutes or longer.

Before we proceed to the next step, there is one more thing we need to, which is to make sure we are able to ssh from every node to others without password. To achieve that, we need to execute `cm cluster cross_ssh`:

```
$ cm cluster cross_ssh
```

10.2.5.3.2 Word Count Example on Spark

Now with the cluster ready, let's run a simple Spark job, Word Count, on one of William Shakespeare's work. Use `cm vm ssh` to log into the `namenode` of the Spark cluster. It should be the first node of the cluster:

```
$ cm vm ssh node-001
cc@hostname$
```

Switch to user `hadoop` and check HDFS is set up or not:

```
cc@hostname$ sudo su - hadoop
hadoop@hostname$
```

Download the input file from the Internet:

```
wget --no-check-certificate -O inputfile.txt \
https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt
```

You can also use any other text file you preferred. Create a new directory `wordcount` within HDFS to store the input and output:

```
$ hdfs dfs -mkdir /wordcount
```

Store the input text file into the directory:

```
$ hdfs dfs -put inputfile.txt /wordcount/inputfile.txt
```

Save the following code as `wordcount.py` on the local file system on Namenode:

```
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    # tak two arguments, input and output
    if len(sys.argv) != 3:
        print("Usage: wordcount <input> <output>")
        exit(-1)

    # create Spark context with Spark configuration
    conf = SparkConf().setAppName("Spark Count")
    sc = SparkContext(conf=conf)

    # read in text file
    text_file = sc.textFile(sys.argv[1])

    # split each line into words
    # count the occurrence of each word
    # sort the output based on word
    counts = text_file.flatMap(lambda line: line.split(" ")) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a + b) \
        .sortByKey()

    # save the result in the output text file
    counts.saveAsTextFile(sys.argv[2])
```

Next submit the job to Yarn and run in distribute:

```
$ spark-submit --master yarn --deploy-mode client --executor-memory 1g \
--name wordcount --conf "spark.app.id=wordcount" wordcount.py \
hdfs://192.168.0.236:8020/wordcount/inputfile.txt \
hdfs://192.168.0.236:8020/wordcount/output
```

Finally, take a look at the result in the output directory:

```
$ hdfs dfs -ls /wordcount/outputfile/
Found 3 items
-rw-r--r-- 1 hadoop hadoop,hadoopadmin 0 2017-03-07 21:28 /wordcount/output/_SUCCESS
-rw-r--r-- 1 hadoop hadoop,hadoopadmin 483182 2017-03-07 21:28 /wordcount/output/part-00000
-rw-r--r-- 1 hadoop hadoop,hadoopadmin 639649 2017-03-07 21:28 /wordcount/output/part-00001
$ hdfs dfs -cat /wordcount/output/part-00000 | less
(u', 517065)
(u'', 241)
(u''\Tis', 1)
(u'A', 4)
(u'AS-IS', 1)
(u'Air,', 1)
(u'Alas,', 1)
(u'Amen'', 2)
(u'Amen"?', 1)
(u'Amen,', 1)
...
```

10.3 SPARK

10.3.1 Spark Lectures

This section covers an introduction to Spark that is split up into eight parts. We discuss Spark background, RDD operations, Shark, Spark ML, Spark vs Other Frameworks.

10.3.1.1 Motivation for Spark

In this section we discuss about the background of Spark and core components of Spark.



[Spark 15:57 Spark A](#)

10.3.1.2 Spark RDD Operations

In this section we discuss about the background of RDD operations along with other transformation functionality in Spark.



[Spark 12:17 Spark B](#)

10.3.1.3 Spark DAG

In this section we discuss about the background of DAG (direct acyclic graphs) operations along with other components like Shark in the earlier stages of Spark.



[Spark 10:37 Spark C](#)

10.3.1.4 Spark vs. other Frameworks

In this section we discuss about the real world applications that can be done using Spark. And also we discuss some comparison results obtained from experiments done in Spark along with Frameworks like Harp, Harp DAAL, etc. We discuss the benchmarks and performance obtained from such experiments.



[Spark 26:18 Spark D](#)

10.3.2 Installation of Spark

In this section we will discuss how to install Spark 2.3.2 in Ubuntu 18.04.

10.3.2.1 Prerequisites

We assume that you have ssh, and rsync installed and use emacs as editor.

```
sudo apt-get install ssh
sudo apt-get install rsync
sudo apt-get install emacs
```

10.3.2.2 Installation of Java

First download Java 8.

```
mkdir -p ~/cloudmesh/bin
cd ~/cloudmesh/bin
wget -c --header "Cookie: oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/8u161-b1
tar xvzf jdk-8u161-linux-x64.tar.gz
```

Then add the environmental variables to the bashrc file.

```
emacs ~/.bashrc

export JAVA_HOME=~/cloudmesh/bin/jdk1.8.0_161
export PATH=$JAVA_HOME/bin:$PATH
```

Source the bashrc file after adding the environmental variables.

```
source ~/.bashrc
```

10.3.2.3 Install Spark with Hadoop

Here we use Spark packaged with Hadoop. In this package Spark uses Hadoop 2.7.0 in the packaged version. Note that in Section [Hadoop Installation](#) we use for the vanilla Hadoop installation Hadoop 3.0.1.

Create the base directories and go to the directory.

```
mkdir -p ~/cloudmesh/bin  
cd ~/cloudmesh/bin
```

Then download Spark 2.3.2 as follows.

```
wget https://archive.apache.org/dist/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
```

Now extract the file,

```
tar xzf spark-2.3.2-bin-hadoop2.7.tgz  
mv spark-2.3.2-bin-hadoop2.7 spark-2.3.2
```

10.3.2.4 Spark Environment Variables

Open up bashrc file and add environmental variables as follows.

```
emacs ~/.bashrc
```

Go to the last line and add the following content.

```
export SPARK_HOME=~/cloudmesh/bin/spark-2.3.2  
export PATH=$SPARK_HOME/bin:$PATH
```

Source the bashrc file.

```
source ~/.bashrc
```

10.3.2.5 Test Spark Installation

Open up a new terminal and then run the following command.

```
spark-shell
```

If it has been configured properly, it will display the following content.

```
Spark context Web UI available at http://192.168.1.66:4041  
Spark context available as 'sc' (master = local[*], app id = local-1521674331361).  
Spark session available as 'spark'.
```

```
Welcome to
```



```
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.
```

Please check the console LOGS and find the port number on which the Spark Web UI is hosted. It will show something like:

Spark context Web UI available at: <some url>

Then take a look the following address in the browser.

```
http://localhost:4041
```

If you see the Spark Dashboard, then you can realize you have installed Spark successfully.

10.3.2.6 Install Spark With Custom Hadoop

Installing Spark with pre-existing Hadoop version is favorable, if you want to use the latest features from the latest Hadoop version or when you need a specific Hadoop version depending on the external dependencies to your project.

First we need to download the Spark packaged without Hadoop.

```
mkdir -p ~/cloudmesh/bin
cd ~/cloudmesh/bin
```

Then download Spark 2.3.2 as follows.

```
wget https://archive.apache.org/dist/spark/spark-2.3.2/spark-2.3.2-bin-without-hadoop.tgz
```

Now extract the file,

```
tar xzf spark-2.3.2-bin-without-hadoop.tgz
```

Then add the environmental variables,

If you have already installed Spark with Hadoop by following section [1.3](#) please update the current SPARK_HOME variable with the new path.

```
emacs ~/.bashrc
```


Type :help for more information.

Then take a look the following address in the browser.

```
http://localhost:4040
```

Please check the console LOGS and find the port number on which the Spark Web UI is hosted. It will show something like: Spark context Web UI available at the `_logs` folder

10.3.3 Spark Streaming

10.3.3.1 Streaming Concepts

Spark Streaming is one of the components extending from Core Spark. Spark streaming provides a scalable fault tolerant system with high throughput. For streaming data into spark, there are many libraries like Kafka, Flume, Kinesis, etc.

10.3.3.2 Simple Streaming Example

In this section, we are going to focus on making a simple streaming application using the network in your computer. Here we are going to expose a particular port and from that port we are going to continuously stream data by user entries and the word count is being calculated as the output.

First, create a Makefile

```
mkdir -p ~/cloudmesh/spark/examples/streaming
cd ~/cloudmesh/spark/examples/streaming
emacs Makefile
```

Then add the following content to Makefile.

Please add a tab when adding the corresponding command for a given instruction in Makefile. In pdf mode the tab is not clearly shown.

```
SPARKHOME = ${SPARK_HOME}
run-streaming:
    ${SPARKHOME}/bin/spark-submit streaming.py localhost 9999
```

Now we need to create file called `streaming.py`

Then add the following content.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "NetworkWordCount")

log4jLogger = sc._jvm.org.apache.log4j
LOGGER = log4jLogger.LogManager.getLogger(__name__)
LOGGER.info("Pyspark script logger initialized")

ssc = StreamingContext(sc, 1)

# Create a DStream that will connect to hostname:port, like localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.pprint()
ssc.start() # Start the computation
ssc.awaitTermination(100) # Wait for the computation to terminate
```

To run the code, we need to open up two terminals.

Terminal 1 :

First use netstat to open up a port to start communication.

```
nc -lk 9999
```

Terminal 2 :

Now run the Spark programme in the second terminal.

```
make run-streaming
```

In this terminal you can see an script running trying to read the stream coming from the port 9999. You can enter texts in the Terminal 1 and these texts will be tokenized and the word count is calculated and the result is shown in the Terminal 2.

10.3.3.3 Spark Streaming For Twitter Data

In this section we are going to learn how to use Twitter data as the streaming data source and use Spark Streaming capabilities to process the data. As the first step you must install the python packages using pip.

10.3.3.3.1 Step 1

```
sudo pip install tweepy
```

10.3.3.3.2 Step 2

Then you need to create an account in Twitter Apps. Go to and sign in to your twitter account or create a new twitter account. Then you need to create a new application, let's name this application as `Cloudmesh-Spark-Streaming`.

First you need to create an app with the app name we suggested in this section. The way to create the app is mentioned in [+Figure 93](#).

Create an application

The screenshot shows the 'Create an application' form in the Twitter Developer Portal. The form is titled 'Application Details' and contains the following fields and sections:

- Name ***: A text input field containing 'Cloudmesh-Spark-Streaming'. Below it is a note: 'Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.'
- Description ***: A text input field containing 'Testing Spark Streaming'. Below it is a note: 'Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.'
- Website ***: A text input field containing 'https://www.vibhatha.org'. Below it is a note: 'Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)'
- Callback URL**: An empty text input field. Below it is a note: 'Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.'

Below the form is a 'Developer Agreement' section with a checked checkbox and the text 'Yes, I have read and agree to the [Twitter Developer Agreement](#).' At the bottom of the form is a button labeled 'Create your Twitter application'.

Figure 93: Create Twitter App

Next we need to take a look at the dashboard created for the app. You can see how your dashboard looks like in [+Figure 94](#).

Your application has been created. Please take a moment to review and adjust your application's settings.

Cloudmesh-Spark-Streaming

Test OAuth

Details Settings **Keys and Access Tokens** Permissions

 Testing Spark Streaming
<https://www.vibhatha.org>

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Figure 94: Go To Twitter App Dashboard

Next the application tokens generated must be reviewed and it can be found in +[Figure 95](#), you need to go to the `Keys and Access Tokens` tab.

Cloudmesh-Spark-Streaming

Test OAuth

Details Settings Keys and Access Tokens **Permissions**

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level Read and write ([modify app permissions](#))

Owner

Owner ID

Application Actions

Regenerate Consumer Key and Secret

Change App Permissions

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

Create my access token

Figure 95: Create Your Twitter Settings

Now you need to generate the access tokens for the first time if you have not generated access tokens and this can be done by clicking the `Create my access token` button. See +[Figure 96](#)

Cloudmesh-Spark-Streaming

Test OAuth

Details Settings Keys and Access Tokens Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level Read and write (modify app permissions)

Owner

Owner ID

Application Actions

Regenerate Consumer Key and Secret

Change App Permissions

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

Create my access token

Figure 96: Create Your Twitter Access Tokens

The access tokens and keys are blurred in this section for privacy issues.

10.3.3.3 Step 3

Let us build a simple Twitter App to see if everything is okay.

```
mkdir -p ~/cloudmesh/spark/streaming
cd ~/cloudmesh/spark/streaming
emacs twitterstreaming.py
```

Add the following content to the file and make sure you update the corresponding token keys with your token values.

```
import tweepy

CONSUMER_KEY = 'your_consumer_key'
CONSUMER_SECRET = 'your_consumer_secret'
ACCESS_TOKEN = 'your_access_token'
ACCESS_TOKEN_SECRET = 'your_access_token_secret'
```

```

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth)

status = "Testing!"
api.update_status(status=status)

```

```
python twitterstreaming.py
```

10.3.3.3.4 Step 4

Let us start the twitter streaming exercise. We need to create a Tweet Listener in order to retrieve data from twitter regarding a topic of your choice. In this exercise, we have tried keywords like `trump, indiana, messi`.

```

mkdir -p ~/cloudmesh/spark/streaming
cd ~/cloudmesh/spark/streaming
emacs tweetlistener.py

```

Make your to replace strings related to secret keys and ip addresses by replacing these values depending on your machine configuration and twitter keys.

Now add the following content.

```

import tweepy
from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener
import socket
import json

CONSUMER_KEY = 'YOUR_CONSUMER_KEY'
CONSUMER_SECRET = 'YOUR_CONSUMER_SECRET'
ACCESS_TOKEN = 'YOUR_ACCESS_TOKEN'
ACCESS_SECRET = 'YOUR_SECRET_ACCESS'

class TweetListener(StreamListener):

    def __init__(self, csocket):
        self.client_socket = csocket

    def on_data(self, data):
        try:
            msg = json.loads( data )
            print( msg['text'].encode('utf-8') )
            self.client_socket.send( msg['text'].encode('utf-8') )
            return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
            return True

    def on_error(self, status):
        print(status)
        return True

def sendData(c_socket):
    auth = OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
    auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)

    twitter_stream = Stream(auth, TweetListener(c_socket))
    twitter_stream.filter(track=['messi']) # you can change this topic

if __name__ == "__main__":

```

```

s = socket.socket()
host = "YOUR_MACHINE_IP"
port = 5555
s.bind((host, port))

print("Listening on port: %s" % str(port))

s.listen(5)
c, addr = s.accept()

print( "Received request from: " + str( addr ) )

sendData( c )

```

10.3.3.3.5 step 5

Please replace the local file paths mentioned in this code with a file path of your preference depending on your workstation. And also IP address must be replaced with your ip address. The log folder path must be pre-created and make sure to replace the `registerTempTable` name with respect to the entity that you are referring. This will minimize the conflicts among different topics when you need to plot it in a simple manner.

Add the following content to the IPythonNote book as follows

Open up a terminal,

```

cd ~/cloudmesh/spark/streaming
jupyter notebook

```

Then in the browser the jupyter notebook is being loaded. There create a new IPython notebook called twittersparkstremer.

Then add the following content.

```

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SQLContext
from pyspark.sql.functions import desc

sc = SparkContext('local[2]', 'twittersparkstremer')

ssc = StreamingContext(sc, 10 )
sqlContext = SQLContext(sc)
ssc.checkpoint( "file:///home/<your-username>/cloudmesh/spark/streaming/logs/messi")

socket_stream = ssc.socketTextStream("YOUR_IP_ADDRESS", 5555)

lines = socket_stream.window( 20 )

from collections import namedtuple
fields = ("tag", "count" )
Tweet = namedtuple( 'Tweet', fields )

( lines.flatMap( lambda text: text.split( " " ) )
  .filter( lambda word: word.lower().startswith("#" ) )
  .map( lambda word: ( word.lower(), 1 ) )

```

```

        .reduceByKey( lambda a, b: a + b )
        .map( lambda rec: Tweet( rec[0], rec[1] ) )
        .foreachRDD( lambda rdd: rdd.toDF().sort( desc("count") )
            .limit(10).registerTempTable("tweetsmessi" ) )#change table name depending on your entity

sqlContext

<pyspark.sql.context.SQLContext at 0x7f51922ba350>

ssc.start()

import matplotlib.pyplot as plt
import seaborn as sn

import time
from IPython import display

count = 0
while count < 10:
    time.sleep( 20 )
    top_10_tweets = sqlContext.sql( 'Select tag, count from tweetsmessi' ) #change table name according to your entity
    top_10_df = top_10_tweets.toPandas()
    display.clear_output(wait=True)
    #sn.figure( figsize = ( 10, 8 ) )
    sn.barpplot( x="count", y="tag", data=top_10_df)
    plt.show()
    count = count + 1

ssc.stop()

```

10.3.3.3.6 step 6

Open `Terminal 1`, then do the following

```

cd ~/cloudmesh/spark/streaming
python tweetslistener.py

```

It will show that:

```

Listening on port: 5555

```

Open `Terminal 2`

Now we must start the Spark app by running the content in the IPython Notebook by pressing `SHIFT-ENTER` in each box to run each command. Make sure not to run twice the starting command of the SparkContext or initialization of SparkContext.

Now you will see streams in the `Terminal 1` and you can see plots after a while in the IPython Notebook.

Sample outputs can be seen in [+Figure 97](#), [+Figure 98](#), [+Figure 99](#), [+Figure 100](#).

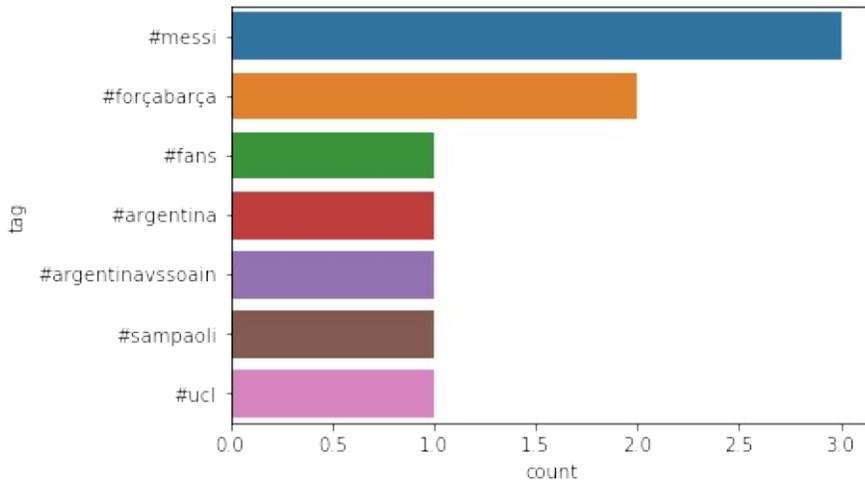


Figure 97: Twitter Topic Messi

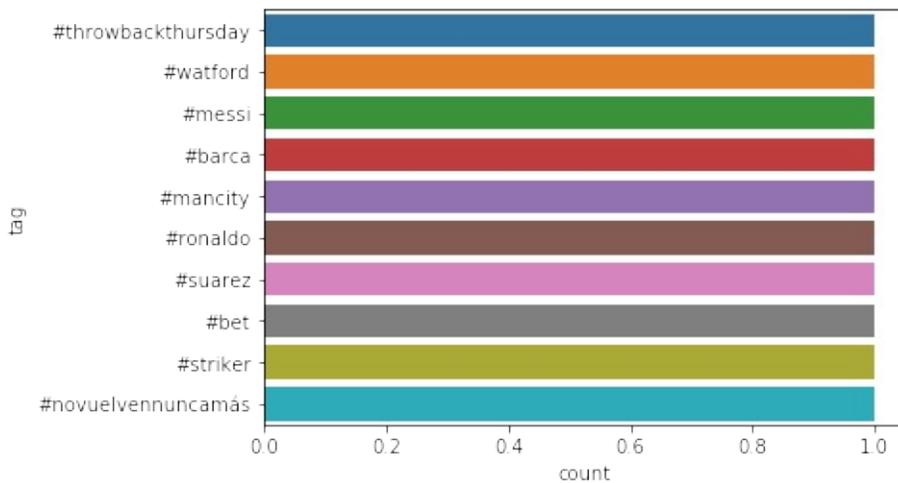


Figure 98: Twitter Topic Messi

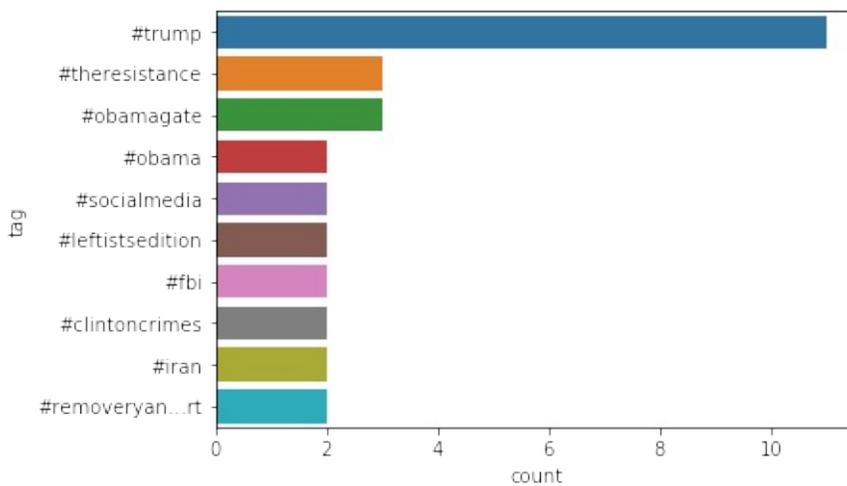


Figure 99: Twitter Topic Messi

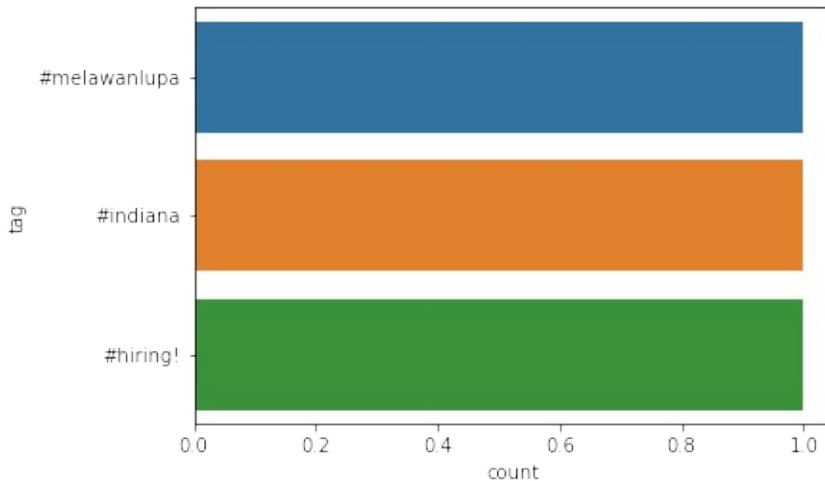


Figure 100: Twitter Topic Messi

10.3.4 User Defined Functions in Spark

Apache Spark is a fast and general cluster-computing framework which perform computational tasks up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk for high speed large-scale streaming, machine learning and SQL workloads tasks. Spark offers support for the applications development employing over 80 high-level operators using Java, Scala, Python, and R. Spark powers the combined or standalone use of a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. Spark can be utilized in standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos and it allows data access in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source.

User-defined functions (UDFs) are the functions created by developers when the built-in functionalities offered in a programming language, are not sufficient to do the required work. Similarly, Apache Spark UDFs also allow developers to enable new functions in higher level programming languages by extending built-in functionalities. It also allows developers to experiment with wide range of options for integrating UDFs with Spark SQL, MLlib and GraphX workflows.

This tutorial explains following:

How to install Spark in Linux, Windows and MacOS.

How to create and utilize user defined functions(UDF) in Spark using Python.

How to run the provided example using a provided docker file and make file.

10.3.4.1 Resources

- <https://spark.apache.org/>
- <http://www.scala-lang.org/>
- <https://docs.databricks.com/spark/latest/spark-sql/udf-in-python.html>

10.3.4.2 Instructions for Spark installation

10.3.4.2.1 Linux

First, JDK (Recommended version 8) should be installed to a path where there is no space.

- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Second, setup environment variables for JDK by adding bin folder path to to user path variable.

```
This $ export PATH = $PATH:/usr/local/java8/bin
```

Next, download and extract Scala pre-built version from

- <http://www.scala-lang.org/download/>

Then, setup environment variables for Scala by adding bin folder path to the user path variable.

```
$ export PATH = $PATH:/usr/local/scala/bin
```

Next, download and extract Apache Spark pre-built version.

- <https://spark.apache.org/downloads.html>

Then, setup environment variables for spark by adding bin folder path to the user path variable.

```
$ export PATH = $PATH:/usr/local/spark/bin
```

Finally, for testing the installation, please type the following command.

```
spark-shell
```

10.3.4.3 Windows

First, JDK should be installed to a path where there is no space in that path. Recommended JAVA version is 8.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Second, setup environment variables for jdk by adding bin folder path to to user path variable.

```
set JAVA_HOME=c:\java8
set PATH=%JAVA_HOME%\bin;%PATH%
```

Next, download and extract Apache Spark pre-built version.

<https://spark.apache.org/downloads.html>

Then, setup environment varibale for spark by adding bin folder path to the user path variable.

```
set SPARK_HOME=c:\spark
set PATH=%SPARK_HOME%\bin;%PATH%
```

Next, download the winutils.exe binary and Save winutils.exe binary to a directory (c:\hadoop\bin).

<https://github.com/steveloughran/winutils>

Then, change the winutils.exe permission using following command using CMD with administrator permission.

```
$ winutils.exe chmod -R 777 C:\tmp\hive
```

If your system doesnt have `hive` folder, make sure to create `c:\tmp\hive` directory.

Next, setup environment variables for hadoop by adding bin folder path to the user path variable.

```
set HADOOP_HOME=c:\hadoop\bin
set PATH=%HADOOP_HOME%\bin;%PATH%
```

Then, install Python 3.6 with anaconda (This is a bundled python installer for pyspark).

<https://anaconda.org/anaconda/python>

Finally, for testing the installation, please type the following command.

```
$ pyspark
```

10.3.4.4 MacOS

First, JDK should be installed to a path where there is no space in that path. Recommended JAVA version is 8.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Second, setup environment variables for jdk by adding bin folder path to to user path variable.

```
$ export JAVA_HOME=$(/usr/libexec/java_home)
```

Next, Install Apache Spark using Homebrew with following commands.

```
$ brew update
$ brew install scala
$ brew install apache-spark
```

Then, setup environment variable for spark with following commands.

```
$ export SPARK_HOME="/usr/local/Cellar/apache-spark/2.1.0/libexec/"
$ export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/build:$PYTHONPATH
$ export PYTHONPATH=$SPARK_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH
```

Next, install Python 3.6 with anaconda (This is a bundled python installer for pyspark)

<https://anaconda.org/anaconda/python>

Finally, for testing the installation, please type the following command.

```
$ pyspark
```

10.3.4.5 Instructions for creating Spark User Defined Functions

10.3.4.5.1 Example: Temperature conversion

In this example we convert temperature data from Celsius to Fahrenheit with filtering and sorting

10.3.4.5.1.1 Description about data set

The file **temperature_data.csv** contains temperature data of different weather stations and it has the following structure.

```
ITE00100554,18000101,TMAX,-75,,,E,  
ITE00100554,18000101,TMIN,-148,,,E,  
GM000010962,18000101,PRCP,0,,,E,  
EZE00100082,18000101,TMAX,-86,,,E,  
GM000010962,18000104,PRCP,0,,,E,  
EZE00100082,18000104,TMAX,-55,,,E,
```

We will only consider weather station ID (column 0), entrytype (column 2), temperature (column 3: it is in 10*Celsius)

10.3.4.5.1.2 How to write a python program with UDF

First, we need to import the relevant libraries to use Spark sql built in functionalities listed as follows.

```
from pyspark.sql import SparkSession  
from pyspark.sql import Row
```

Then, we need create a user defined function which will read the text input and process the data and return a spark sql Row object. It can be created as listed as follows.

```
def process_data(line):  
    fields = line.split(',')  
    stationID = fields[0]  
    entryType = fields[2]  
    temperature = float(fields[3]) * 0.1 * (9.0 / 5.0) + 32.0  
    return Row(ID=stationID, t_type=entryType, temp=temperature)
```

Then we need to create a Spark SQL session as listed as follows with an application name.

```
spark = SparkSession.builder.appName("Simple SparkSQL UDF example").getOrCreate()
```

Next, we read the raw data using spark build-in function `textFile()` as shown

next.

```
lines = spark.sparkContext.textFile("temperature_data.csv")
```

Then, we convert those read lines to a Resilient Distributed Dataset (RDD) of Row object using UDF (process_data) which we created as listed as follows.

```
parsedLines = lines.map(process_data)
```

Alternatively we could have written the UDF using a python lambda function to do the same thing as shown next.

```
parsedLines = lines.map(lambda line: Row(ID=line.split(',')[0],  
t_type=line.split(',')[2],  
temp=float(line.split(',')[3]) * 0.1 * (9.0  
/ 5.0) + 32.0))
```

Now, we can convert our RDD object to a Spark SQL Dataframe as listed as follows.

```
TempDataset = spark.createDataFrame(parsedLines)
```

Next, we can print and see the first 20 rows of data to validate our work as shown next.

```
TempDataset.show()
```

10.3.4.5.1.3 How to execute a python spark script

You can use **spark-submit** command to run a spark script as shown next.

```
spark-submit temperature_converter.py
```

If everything went well, you should see the following output.

```
+-----+-----+-----+  
|          ID|t_type|          temp|  
+-----+-----+-----+  
|EZE00100082| TMAX|90.14000000000001|  
|ITE00100554| TMAX|90.14000000000001|  
|ITE00100554| TMAX|          89.42|  
|EZE00100082| TMAX|          88.88|  
|ITE00100554| TMAX|          88.34|  
|ITE00100554| TMAX|87.80000000000001|  
|ITE00100554| TMAX|          87.62|  
|ITE00100554| TMAX|          87.62|  
|EZE00100082| TMAX|          87.26|  
|EZE00100082| TMAX|87.08000000000001|  
|EZE00100082| TMAX|87.08000000000001|  
|ITE00100554| TMAX|          86.72|  
|ITE00100554| TMAX|          86.72|  
|ITE00100554| TMAX|          86.72|  
|EZE00100082| TMAX|          86.72|  
|ITE00100554| TMAX|          86.0|  
|ITE00100554| TMAX|          86.0|
```

```

|ITE00100554| TMAX|          86.0|
|ITE00100554| TMAX|          85.64|
|ITE00100554| TMAX|          85.64|
+-----+-----+
only showing top 20 rows

```

10.3.4.5.1.4 Filtering and sorting

Now we are trying to find what is the maximum temperature reported for a particular weather station and print the data in ascending order. We can achieve this by using **where()** and **orderBy()** functions as shown next.

```

TempDatasetProcessed = TempDataset.where(TempDataset['t_type'] == 'TMAX')
                              .orderBy('temp', ascending=False).cache()

```

We achieved the filtering using temperature type and it filters out all the data which is not a TMAX.

Finally, we can print the data to see whether this worked or not using following statement.

```

TempDatasetProcessed.show()

```

Now, it is the time to run the python script again using following command.

```

spark-submit temperature_converter.py

```

If everything went well, you should see the following sorted and filtered output.

```

+-----+-----+-----+
|      ID|t_type|      temp|
+-----+-----+-----+
|EZE00100082| TMAX|90.14000000000001|
|ITE00100554| TMAX|90.14000000000001|
|ITE00100554| TMAX|      89.42|
|EZE00100082| TMAX|      88.88|
|ITE00100554| TMAX|      88.34|
|ITE00100554| TMAX|87.80000000000001|
|ITE00100554| TMAX|      87.62|
|ITE00100554| TMAX|      87.62|
|EZE00100082| TMAX|      87.26|
|EZE00100082| TMAX|87.08000000000001|
|EZE00100082| TMAX|87.08000000000001|
|ITE00100554| TMAX|      86.72|
|ITE00100554| TMAX|      86.72|
|ITE00100554| TMAX|      86.72|
|EZE00100082| TMAX|      86.72|
|ITE00100554| TMAX|      86.0|
|ITE00100554| TMAX|      86.0|
|ITE00100554| TMAX|      86.0|
|ITE00100554| TMAX|      85.64|
|ITE00100554| TMAX|      85.64|
+-----+-----+-----+
only showing top 20 rows

```

Complete python script is listed as follows as well as under this directory (temperature_converter.py).

https://github.com/cloudmesh-community/hid-sp18-409/blob/master/tutorial/spark_udfs/temperature_converter.py

```
from pyspark.sql import SparkSession
from pyspark.sql import Row

def process_data(line):
    fields = line.split(',')
    stationID = fields[0]
    entryType = fields[2]
    temperature = float(fields[3]) * 0.1 * (9.0 / 5.0) + 32.0
    return Row(ID=stationID, t_type=entryType, temp=temperature)

# Create a SparkSQL Session
spark = SparkSession.builder.appName('Simple SparkSQL UDF example')
    .getOrCreate()

# Get the raw data
lines = spark.sparkContext.textFile('temperature_data.csv')

# Convert it to a RDD of Row objects
parsedLines = lines.map(process_data)

# alternative lamda funtion

parsedLines = lines.map(lambda line: Row(ID=line.split(',')[0],
    t_type=line.split(',')[2],
    temp=float(line.split(',')[3]) * 0.1 * (9.0
    / 5.0) + 32.0))

# Convert that to a DataFrame
TempDataset = spark.createDataFrame(parsedLines)

# show first 20 rows temperature converted data
# TempDataset.show()

# Some SQL-style magic to sort all movies by popularity in one line!
TempDatasetProcessed = TempDataset.where(TempDataset['t_type'] == 'TMAX')
    .orderBy('temp', ascending=False).cache()

# show first 20 rows of filtered and sorted data
TempDatasetProcessed.show()
```

10.3.4.6 Instructions to install and run the example using docker

Following link is the home directory for the example explained in this tutorial.

https://github.com/cloudmesh-community/hid-sp18-409/tree/master/tutorial/spark_udfs

It contains following files

- Python script which contains the example: [temperature_converter.py](#)
- Temperature data file: [temperature_data.csv](#)
- Required python dependencies are put here: [requirements.txt](#)
- Docker file which automatically setup the codebase with dependency installation: [Dockerfile](#)

- Make file which will excute the example with a single command: [Makefile](#)

To install the example using docker plesse do the following steps.

First, you should install docker in to your computer.

Next, git clone the [project](#) . Alternatively you can also download the docker image from the docker hub. Then you don't need to do docker build.

```
$ docker pull kadupitiya/tutorial
```

Then, change the directory to **spark_udfs** folder.

Next, install the service using following make command

```
$ make docker-build
```

Finally, start the service using following make command

```
$ make docker-start
```

Now you should see the same output we saw at the end of the example explanation.

10.4 HADOOP ECOSYSTEM

10.4.1 ELASTIC MAP REDUCE

10.4.1.1 AWS Elastic Map Reduce (AWS EMR)



Learning Objectives

- Learn about EMR
 - Deploy an EMR cluster using:
 - Amazon's Command Line Interface (CLI)
 - Amazon's Web Interfaces
 - Run an example Spark application on an EMR cluster
-

10.4.1.1.1 Introduction

EMR is an Amazon product that allows for the creation of clusters of Elastic Compute Cloud (EC2) instances [76]. EMR allows user to take advantage of distributed computing capabilities. As the name suggests this product is designed to allow users to easily scale their cluster to meet their computing needs.

Amazon EMR facilitates you to analyze and process vast(huge) amounts of data by distributing the computational work across a cluster of virtual servers running in the AWS Cloud. The EMR cluster is managed using an open-source framework called Hadoop. Amazon EMR lets you focus on crunching or analyzing your data without having to worry about time-consuming setup, management, and tuning of Hadoop clusters or the compute capacity they rely on unlike other Hadoop distributors like Cloudera, Hortonworks etc.,

10.4.1.1.2 Why EMR?

The following are reasons given by Amazon for using EMR:

- Easy to Use: Launch cluster in a 5 to 10 minutes time as many cluster of nodes as you need
- Pay as you go: Pay an hourly rate (with AWS latest pricing model, customers can choose to pay in minutes)
- Flexible: Easily Add/ Remove capacity(Auto scale out and in anytime)
- Reliable: Spend less time for monitoring and can utilize in-built AWS tools which will reduce overhead
- Secure: Manage firewall (VPC both private and subnet)

EMR clusters can be created through relatively simple web interfaces or can be created through code using CLI. EMR Clusters can be configured for size and can be provisioned with open-source distributed frameworks such as SPARK and HBase, Presto, Flink and etc. Interact with data in AWS data stores such as Amazon S3, DynamoDB and etc.

Components Of EMR:

- Storage
- EC2 instance

- Clusters
- Security
- KMS

10.4.1.1.3 Understanding Clusters and Nodes

The component of Amazon EMR is the cluster. A cluster is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. Each instance in the cluster is called a node. Each node has a role within the cluster, referred to as the node type. Amazon EMR also installs different software components on each node type, giving each node a role in a distributed application like Apache Hadoop.

The node types in Amazon EMR are as follows:

- Master node: A node that manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing. The master node tracks the status of tasks and monitors the health of the cluster. Every cluster has a master node, and it is possible to create a single-node cluster with only the master node.
- Core node: A node with software components that run tasks and store data in the Hadoop Distributed File System (HDFS) on your cluster. Multi-node clusters have at least one core node.
- Task node: A node with software components that only runs tasks and does not store data in HDFS. Task nodes are optional.

The following diagram represents a cluster with one master node and four core nodes.

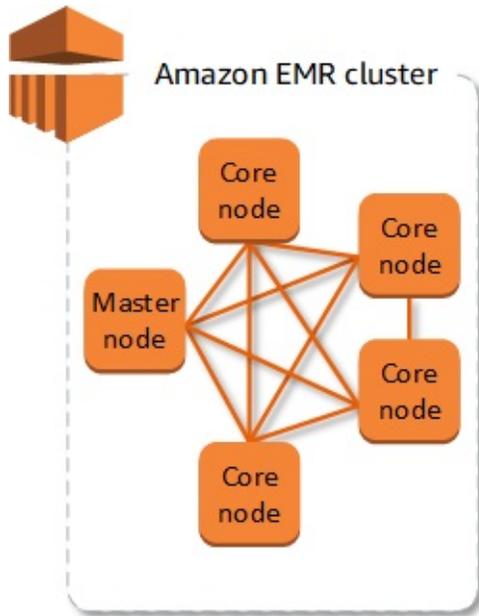


Figure 101: Cluster and Nodes [77]

10.4.1.1.4 Prerequisites

Official prerequisites are listed here:
<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-prerequisites.html>

- [AWS Account](#)
- [AWS Key Pair](#)
- [Install and Configure AWS CLI](#)
- [AWS Admin Access](#)
- [Linux Environment](#)

10.4.1.1.5 Creating EMR Cluster Using CLI

10.4.1.1.5.1 Create Security Roles

In this example we will use the default EMR security roles. These roles enable the nodes within the cluster to access each other and to access other AWS products.

```
aws emr create-default-roles
```

10.4.1.1.5.2 Setting up authentication

In this example we will be using Kerberos for authentication. The Kerberos configuration would allow you to add additional users to your EMR cluster.

Create a json file with the following content and save to a local file:

```
{
  "AuthenticationConfiguration": {
    "KerberosConfiguration": {
      "Provider": "ClusterDedicatedKdc",
      "ClusterDedicatedKdcConfiguration": {
        "TicketLifetimeInHours": 24
      }
    }
  }
}
```

Create the Kerberos configuration using the previously created json file:

```
aws emr create-security-configuration --name "KerberosSecurityConfiguration" --security-configuration file://MyKerberosSe
```

10.4.1.1.5.3 Determine the applicable subnet

The EMR cluster will run on a subnet so you need to determine the appropriate subnet for you availability zone. You will need to enter your default zone in the code next.

```
aws ec2 describe-subnets --filters "Name=availabilityZone,Values=us-east-2b"
```

The applicable information is returned as the `SubnetId` field.

10.4.1.1.5.4 Create the EMR cluster

In this example we will create a simple cluster with 3 nodes. One master node and two slave nodes. We will also specify the EC2 instance type (m4.large). These parameters are configurable and you can create larger clusters with more processing power. There are multiple EMR versions available, this example uses the latest version available at the time of creation.

There are a variety of applications that can be installed on the EMR cluster at creation, but in this case we will simply install Spark. The Kerberos password can be used to add users to your cluster once it is created. The KeyName is your EC2 key pair that is referenced in the Prerequisites section.

```
aws emr create-cluster --name "Test-Kerberized-Spark-Cluster" \
--release-label emr-5.17.0 \
--instance-type m4.large \
--instance-count 3 \
```

```
--use-default-roles \  
--ec2-attributes KeyName=your-key,SubnetId=your-subnet-id \  
--security-configuration KerberosSecurityConfiguration \  
--applications Name=Spark \  
--kerberos-attributes Realm=EC2.INTERNAL,KdcAdminPassword=your-password
```

10.4.1.1.5.5 Check the status of your cluster

The cluster may take several minutes to initialize. To check the status of your cluster use the cluster-id that was returned in the previous step.

```
aws emr describe-cluster --cluster-id your-cluster-id
```

10.4.1.1.5.6 Terminate your cluster

To terminate your cluster use the following command (hint: AWS charges apply while your cluster is up).

```
aws emr terminate-clusters --cluster-ids your-cluster-id
```

10.4.1.1.6 Creating EMR Cluster Using AWS Web Console

10.4.1.1.6.1 Set up authentication

Go to the AWS Console and ensure that the URL references your default region (see [Figure 102](#) and [Figure 103](#))

[AWS EMR](#)

Select the `Security configurations` and click `Create`. Give a meaningful name like: `KerberosSecurityConfiguration`. Then select `Kerberos` under `Authentication` and click `Create`.

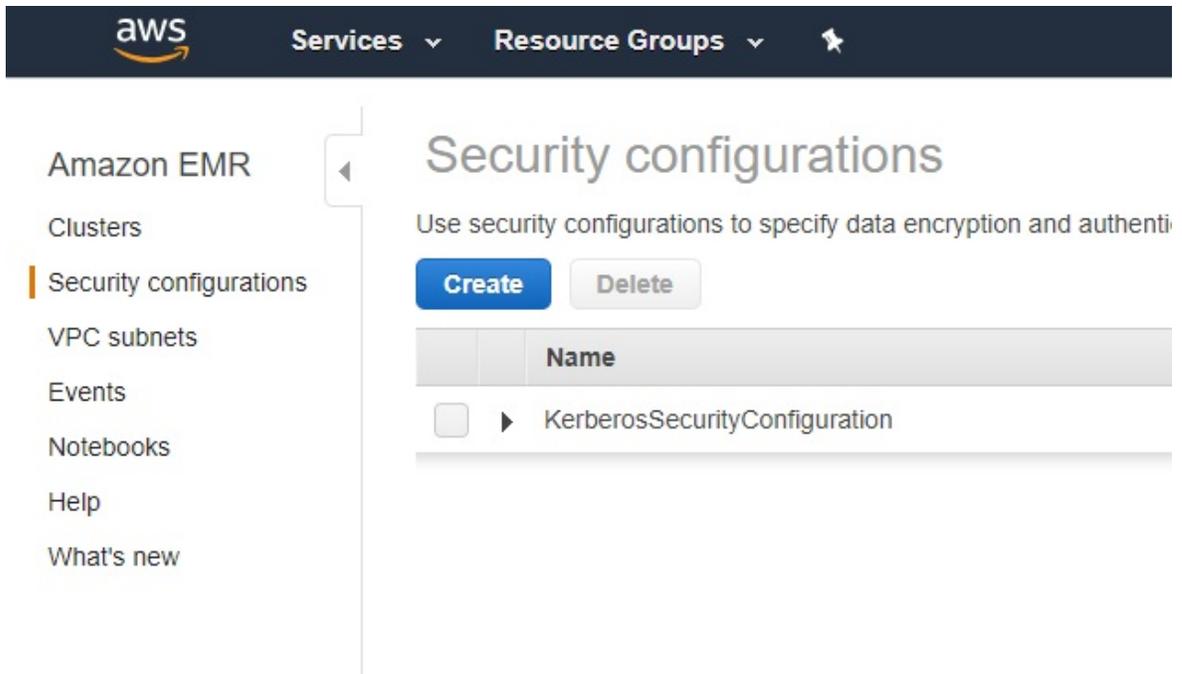


Figure 102: Set up Kerberos 1 [\[77\]](#)

Create security configuration

Name

Encryption

At-rest encryption

Enable and choose options for at-rest data encryption features in Amazon EMR, including Amazon S3 with EMRFS, local volumes attached to cluster instances, and block-transfer encryption for HDFS. [Learn more](#)

S3 encryption

Encryption mode

Local disk encryption

Key provider type

In-transit encryption

Enable and choose options for open-source encryption features that apply to in-transit data for specific applications. Available encryption options may vary by Amazon EMR release. [Learn more](#)

TLS certificate provider

Certificate provider type

Authentication

Kerberos

Enable Kerberos authentication for interactions between certain application components on your cluster using Kerberos principals. If enabled, a Kerberos Key Distribution Center (KDC) is installed on the master node of your cluster. Available authentication options and components may vary by Amazon EMR release. [Learn more](#)

Ticket lifetime hours ⓘ

Cross-realm trust

IAM roles for EMRFS

Use IAM roles for EMRFS requests to Amazon S3

When an Amazon S3 request is made through EMRFS, each Basis for access is evaluated in order. EMRFS assumes the corresponding IAM role for the first match. Specify the cluster Users or Groups, or S3 prefixes as the Basis for access. If no Basis for access matches the request, EMRFS uses the cluster's EMR role for EC2. [Learn more](#)

Cancel

Figure 103: Set up Kerberos 2 [77]

10.4.1.1.6.2 Create the EMR cluster

Go to the AWS Console (ensure that the URL references your default region)

[AWS EMR](#)

Click `Create cluster` (see Figure [Figure 104](#))

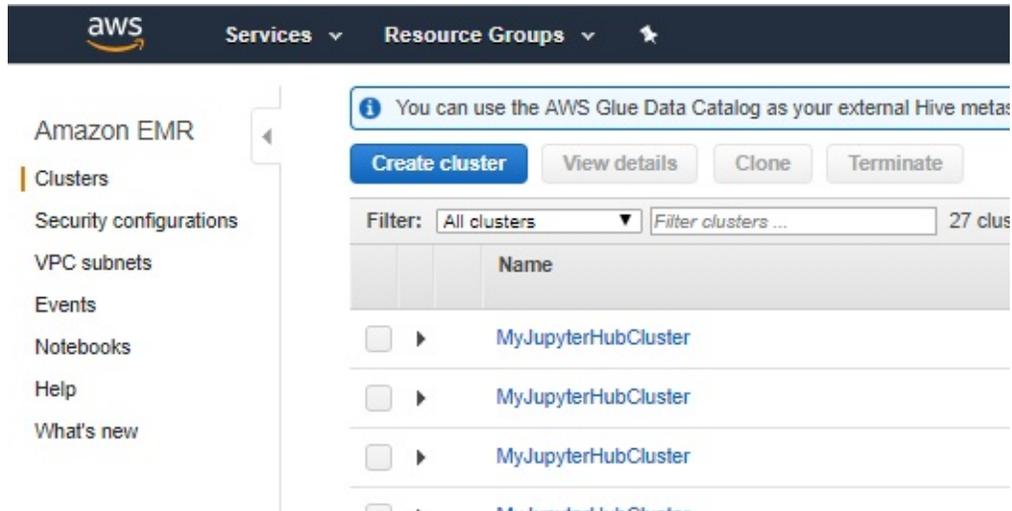


Figure 104: Set up EMR 1 [77]

- Select your desired EMR version
- Select Spark
- Select your desired instance types
- For this example deselect the `Logging` option
- Select your EC2 key Pair

Next, create a cluster (see Figure [Figure 105](#))

General Configuration

Cluster name

Logging ⓘ

Launch mode Cluster ⓘ Step execution ⓘ

Software configuration

Release ⓘ

Applications

- Core Hadoop: Hadoop 2.8.5 with Ganglia 3.7.2, Hive 2.3.3, Hue 4.2.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.8.4
- HBase: HBase 1.4.7 with Ganglia 3.7.2, Hadoop 2.8.5, Hive 2.3.3, Hue 4.2.0, Phoenix 4.14.0, and ZooKeeper 3.4.13
- Presto: Presto 0.212 with Hadoop 2.8.5 HDFS and Hive 2.3.3 Metastore
- Spark: Spark 2.3.2 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.0
- Use AWS Glue Data Catalog for table metadata ⓘ

Hardware configuration

Instance type ⓘ The selected instance type adds a default 32 GiB GP2 EBS volume per instance. [Learn more](#) ⓘ

Number of instances (1 master and 2 core nodes)

Security and access

EC2 key pair ⓘ [Learn how to create an EC2 key pair.](#)

Permissions Default Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR_DefaultRole](#) ⓘ

EC2 instance profile [EMR_EC2_DefaultRole](#) ⓘ

Cancel

Figure 105: Set up EMR 2 [77]

- Under `Advanced Options` select `Security` and then `YourKerberosSecurityConfiguration`
- Click `create cluster`

(See Figure [Figure 106](#))

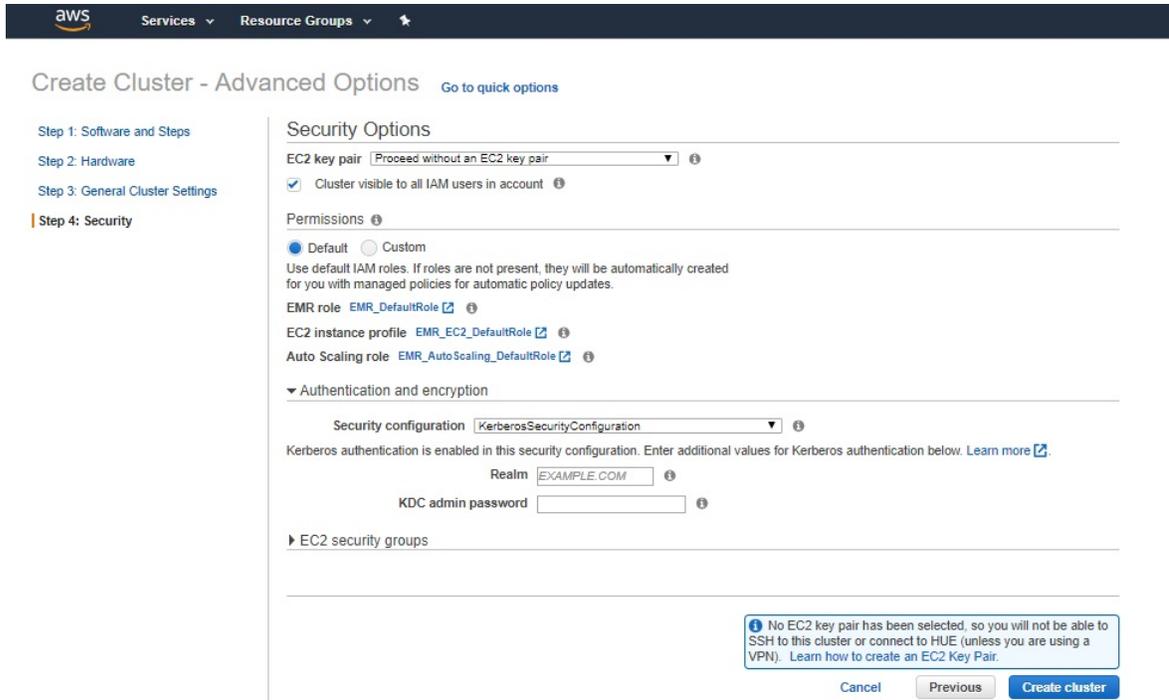


Figure 106: Set up EMR 3 [77]

10.4.1.1.6.3 View status and terminate EMR cluster

You can view the status of your cluster or terminate the cluster by navigating to `>Services>EMR>Clusters` within the AWS Console.

See Figure [Figure 107](#).

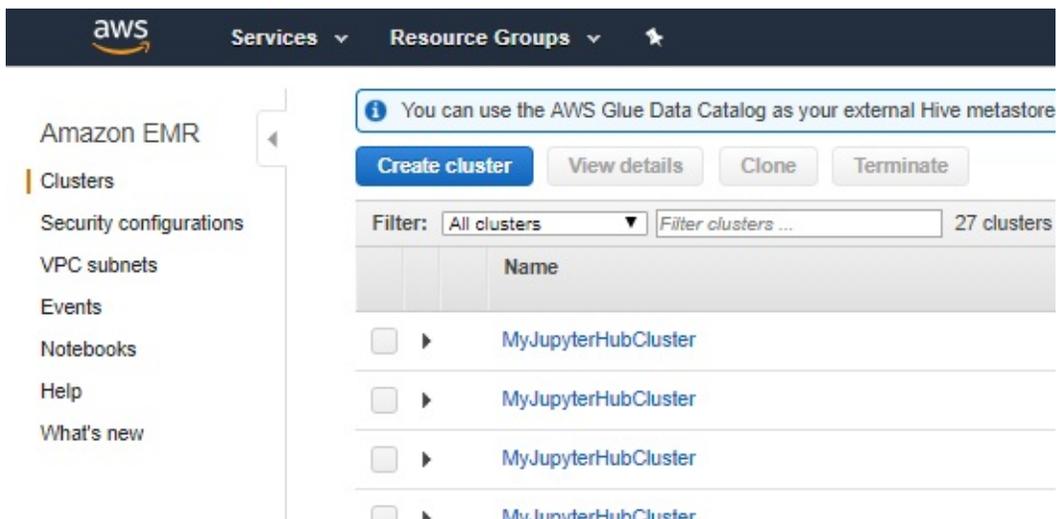


Figure 107: Set up EMR 4 [77]

10.4.1.1.6.4 Submit Work to a Cluster

When you run a cluster on Amazon EMR, you have several options as to how you specify the work that needs to be done.

Provide the entire definition of the work to be done in functions that you specify as steps when you create a cluster. This is typically done for clusters that process a set amount of data and then terminate when processing is complete.

Create a long-running cluster and use the Amazon EMR console, the Amazon EMR API, or the AWS CLI to submit steps, which may contain one or more jobs.

Create a cluster, connect to the master node and other nodes as required using SSH, and use the interfaces that the installed applications provide to perform tasks and submit queries, either scripted or interactively.

10.4.1.1.6.5 Processing Data

When you launch your cluster, you choose the frameworks and applications to install for your data processing needs. To process data in your Amazon EMR cluster, you can submit jobs or queries directly to installed applications, or you can run steps in the cluster.

- Submitting Jobs Directly to Applications:

You can submit jobs and interact directly with the software that is installed in your Amazon EMR cluster. To do this, you typically connect to the master node over a secure connection and access the interfaces and tools that are available for the software that runs directly on your cluster. For more information, see [Connect to the Cluster](#).

- Running Steps to Process Data

You can submit one or more ordered steps to an Amazon EMR cluster. Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.

The following is an example process using four steps:

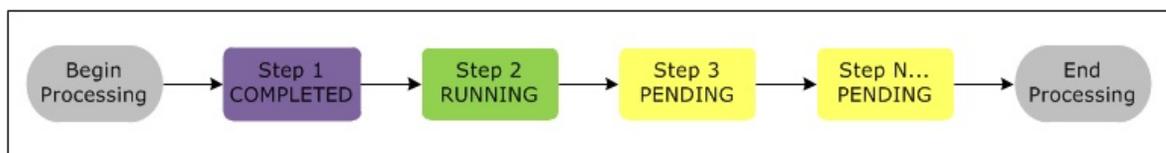
1. Submit an input dataset for processing.
2. Process the output of the first step by using a Pig program.
3. Process a second input dataset by using a Hive program.
4. Write an output dataset.

Generally, when you process data in Amazon EMR, the input is data stored as files in your chosen underlying file system, such as Amazon S3 or HDFS. This data passes from one step to the next in the processing sequence. The final step writes the output data to a specified location, such as an Amazon S3 bucket.

Steps are run in the following sequence:

1. A request is submitted to begin processing steps.
2. The state of all steps is set to PENDING.
3. When the first step in the sequence starts, its state changes to RUNNING. The other steps remain in the PENDING state.
4. After the first step completes, its state changes to COMPLETED.
5. The next step in the sequence starts, and its state changes to RUNNING. When it completes, its state changes to COMPLETED.
6. This pattern repeats for each step until they all complete and processing ends.

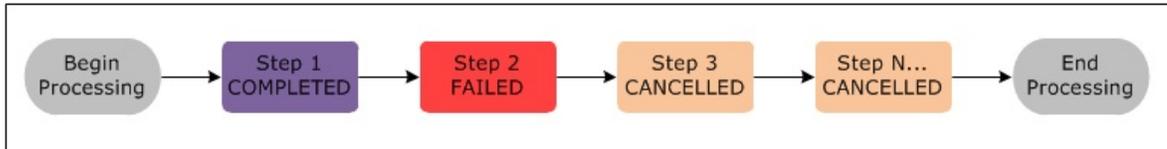
The following diagram represents the step sequence and change of state for the steps as they are processed.



Cluster and Nodes

If a step fails during processing, its state changes to `TERMINATED_WITH_ERRORS`. You can determine what happens next for each step. By default, any remaining steps in the sequence are set to `CANCELLED` and do not run. You can also choose to ignore the failure and allow remaining steps to proceed, or to terminate the cluster immediately.

The following diagram represents the step sequence and default change of state when a step fails during processing.



Cluster and Nodes

10.4.1.1.7 AWS Storage

S3 - Cloud based storage - Using EMRFS can directly connect s3 storage - Accessible from anywhere

Instance Store - Local storage - Data will be lost on start and stop EC2 instances

EBS - Network attached storage - Data preserved on start and stop - Accessible only through EC2 instances

10.4.1.1.8 Create EMR in AWS

10.4.1.1.8.1 Create the buckets

- Login to AWS console and create the buckets at <https://aws.amazon.com/console/>. To create the buckets, go to services (see [Figure 108](#), [Figure 109](#)), click on S3 under Storage, [Figure 110](#), [Figure 111](#), [Figure 112](#). Click on Create bucket button and then provide all the details to complete bucket creation.
- AWS Console

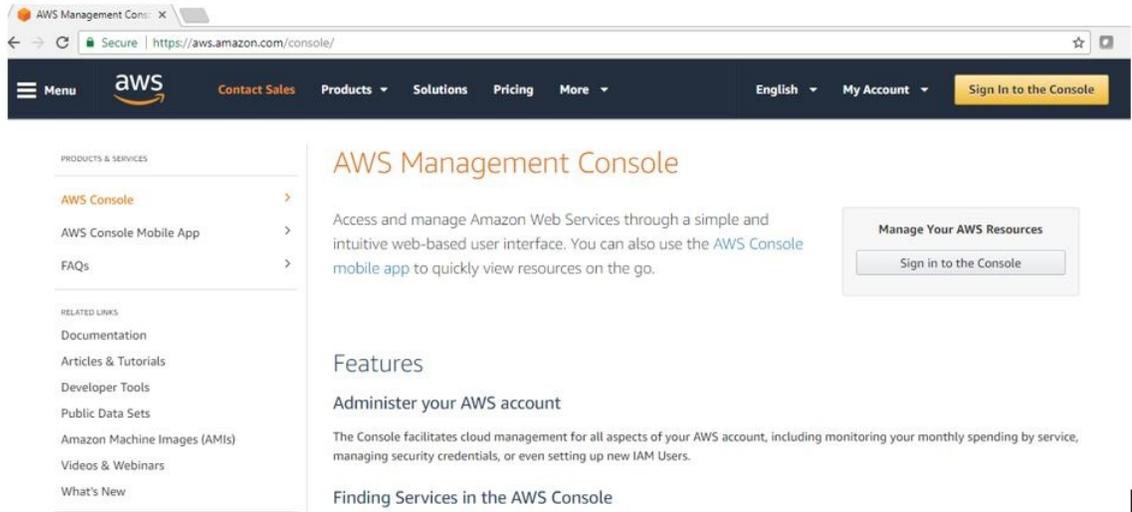


Figure 108: AWS Console

- AWS Login

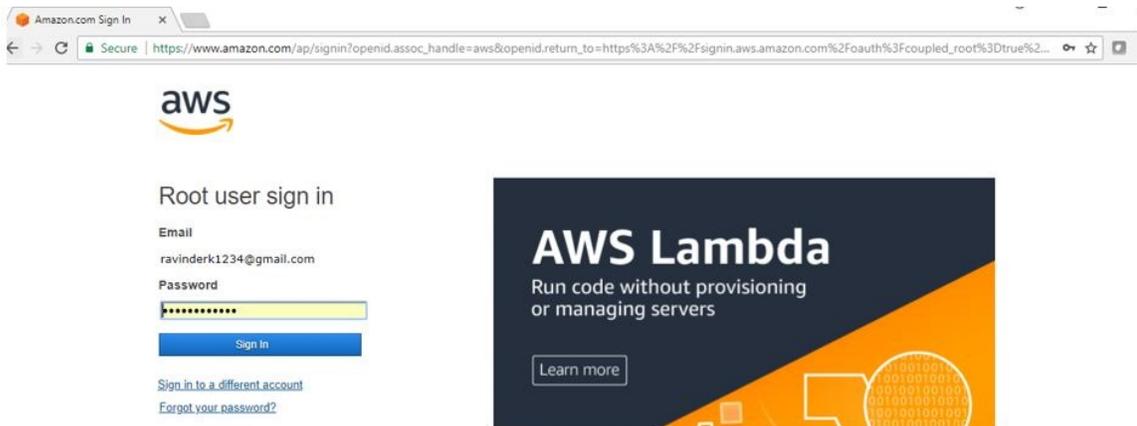


Figure 109: AWS Login

- S3 – Amazon Storage

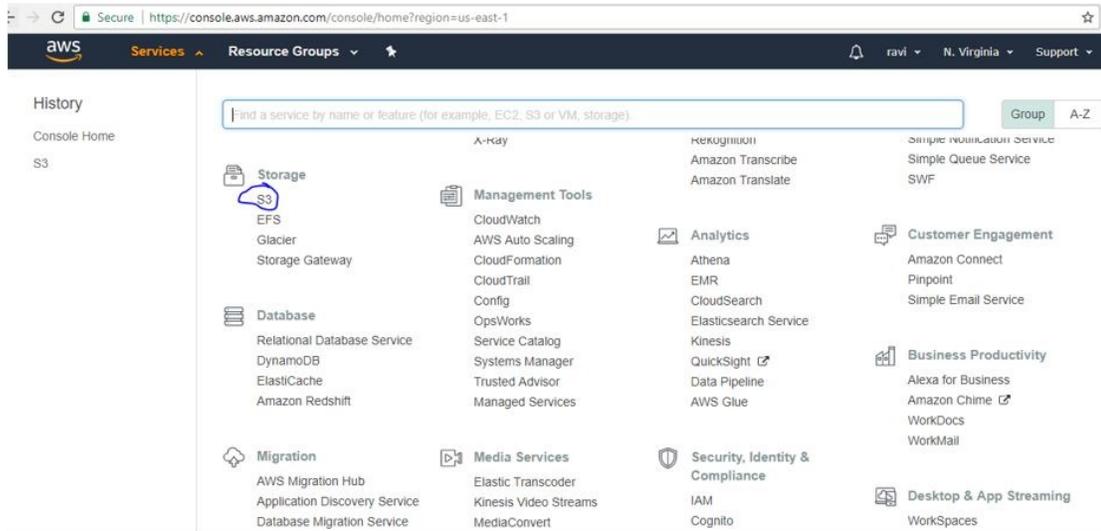


Figure 110: Amazon Storage

- S3 – Create buckets

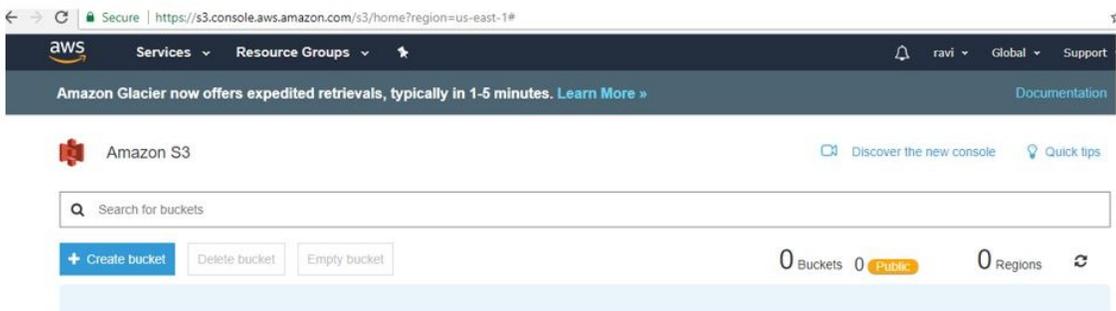


Figure 111: S3 buckets

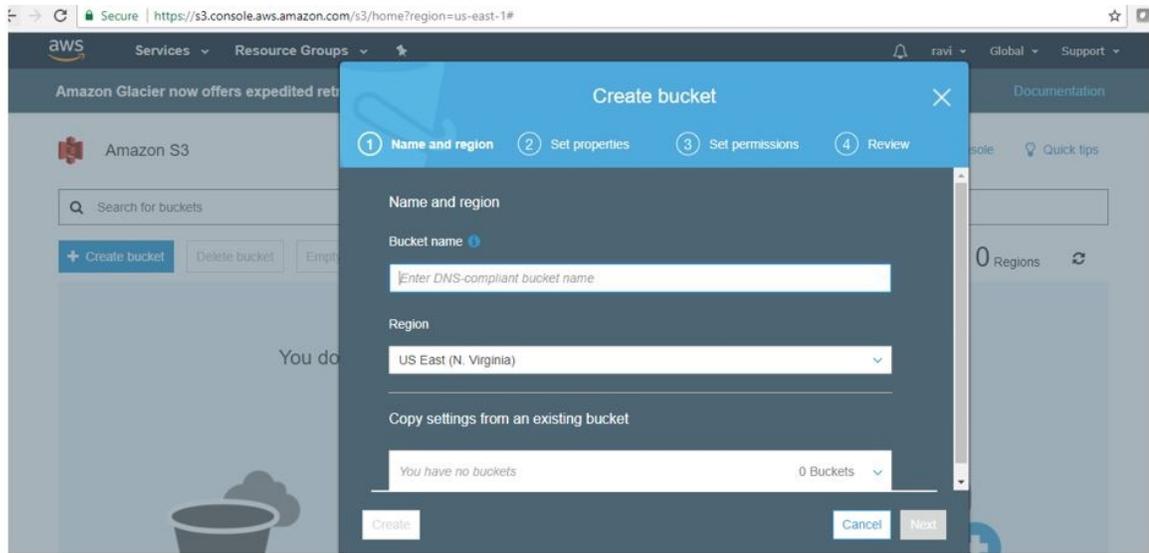


Figure 112: S3 buckets1

10.4.1.1.8.2 Create Key Pairs

- Login to AWS console, go to services, click on EC2 under compute. Select the Key pairs resource, click on Create Key Pair and provide Key Pair name to complete the Key pairs creation. See [Figure 113](#)
- Download the .pem file once Key value pair is created. This is needed to access AWS Hadoop environment from client machine. This need to be imported in Putty to access your AWS environemnt. See [Figure 114](#)

10.4.1.1.8.2.1 Create Key Value Pair Screen shots

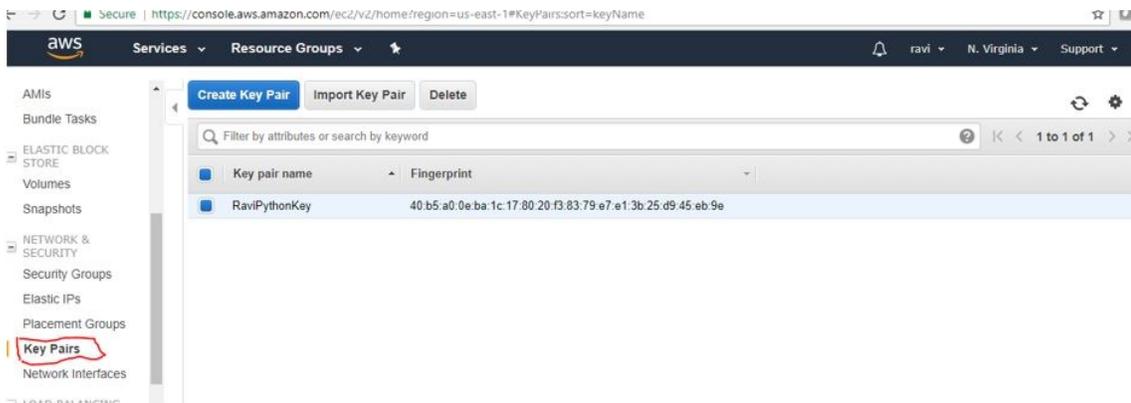


Figure 113: AMS Key Value Pair

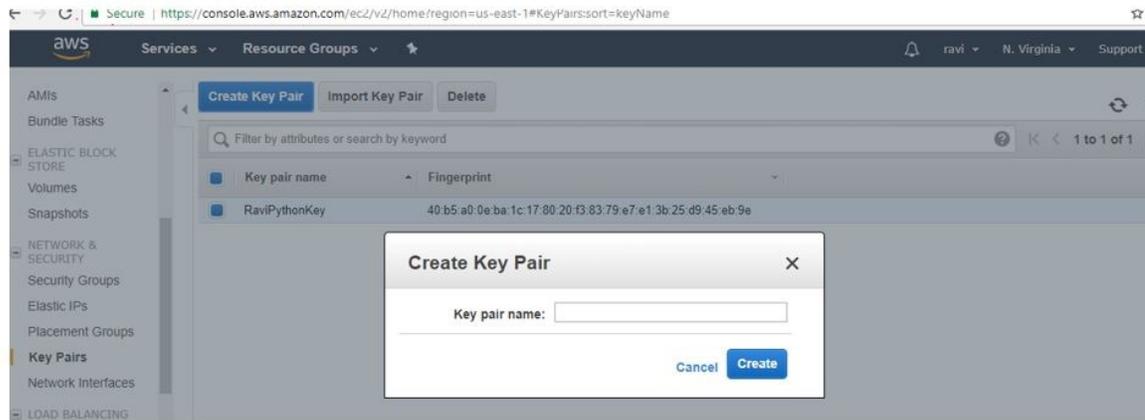


Figure 114: AMS Key Value Pair1

10.4.1.1.9 Create Step Execution – Hadoop Job

Login to AWS console, go to services and then select EMR. Click on Create Cluster. The cluster configuration provides details to complete to complete step execution creation. See: [Figure 115](#), [Figure 116](#), [Figure 117](#), [Figure 118](#), [Figure 119](#).

- Cluster name (Example: HadoopJobStepExecutionCluster)
- Select Logging check box and provide S3 folder location (Example: s3://bigdata-raviAndOrlyiuproject/logs/)
- Select launch mode as Step execution
- Select the step type and complete the step configuration
- Complete Software Configuration
- Complete Hardware Configuration
- Complete Security and access
- And then click on create cluster button
- Once job started, if there are no errors output file will be created in the output directory.

10.4.1.1.9.0.1 Screen shots

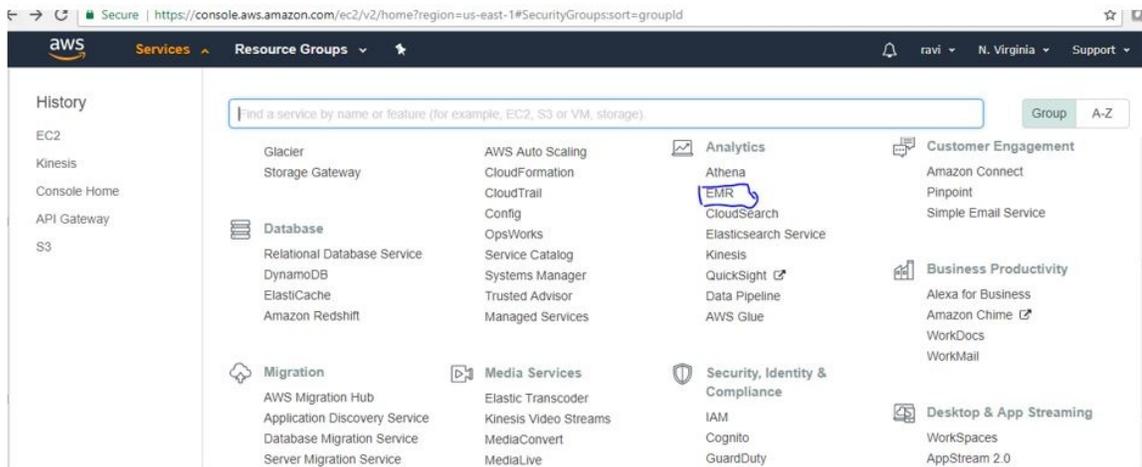


Figure 115: AWS EMR

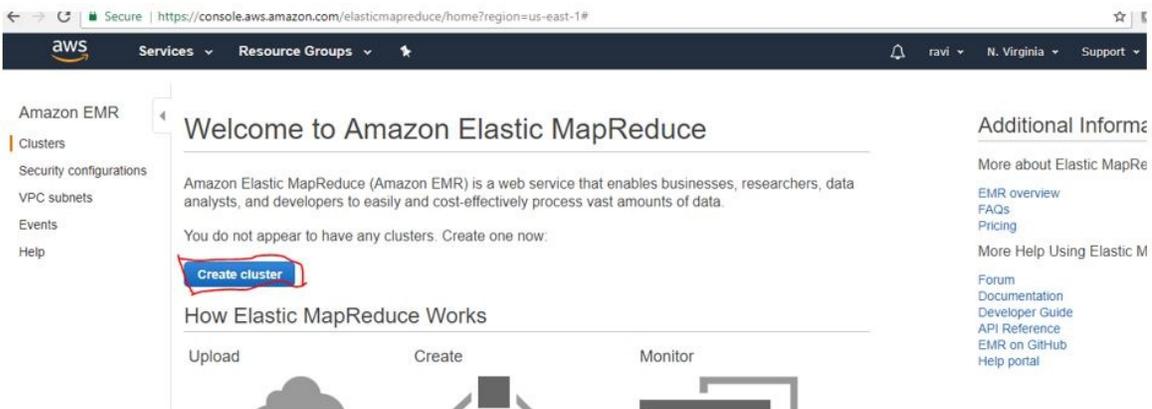


Figure 116: AWS Create EMR

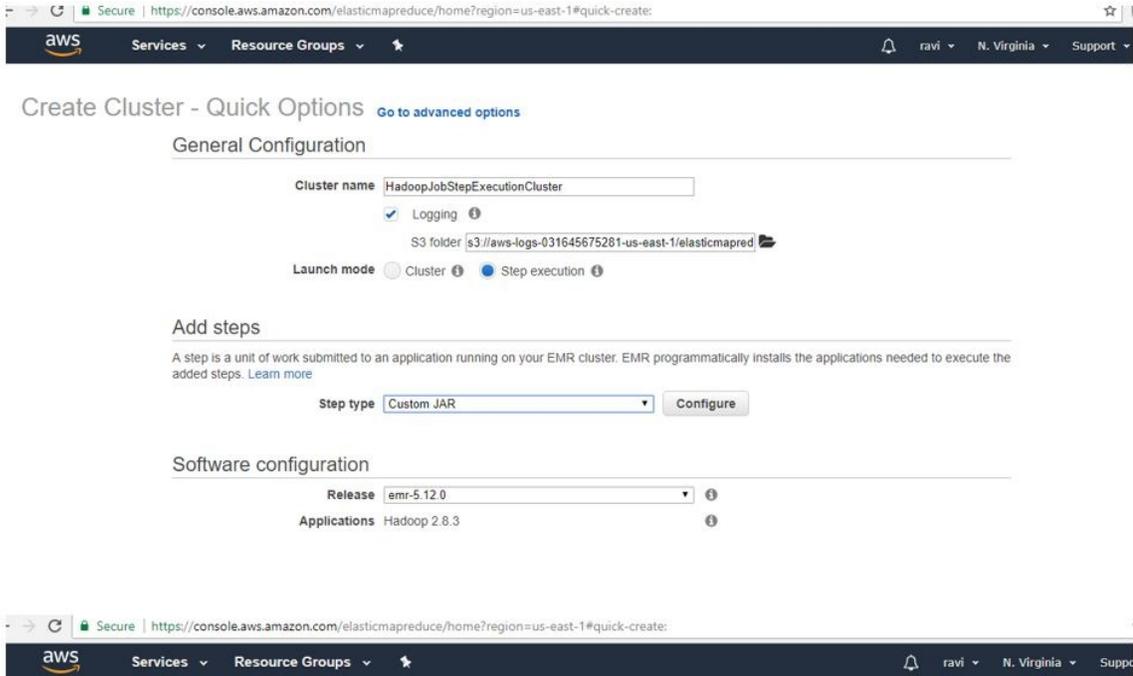


Figure 117: AWS Config EMR

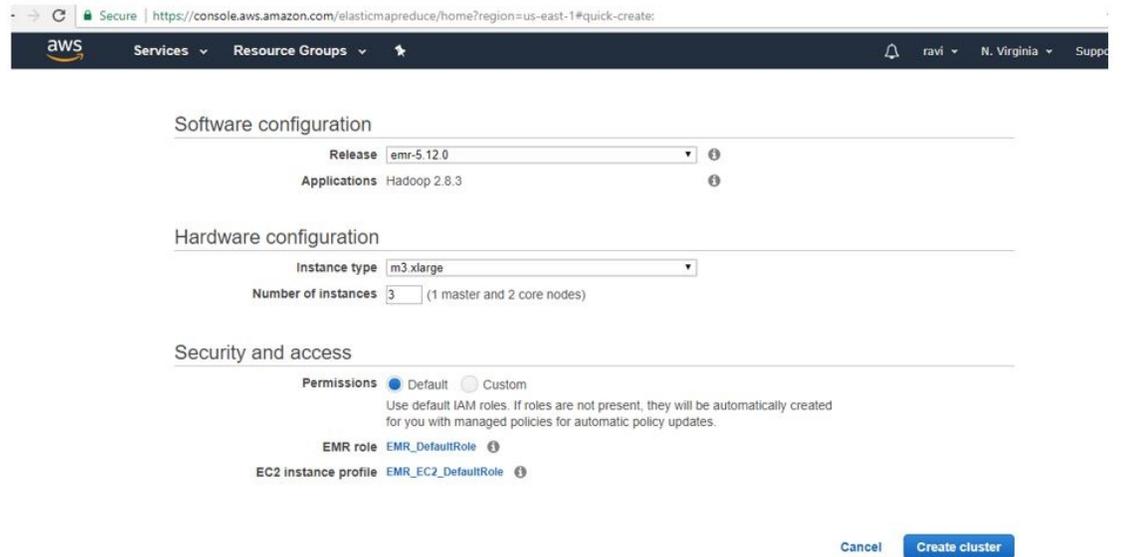


Figure 118: AWS Create Cluster

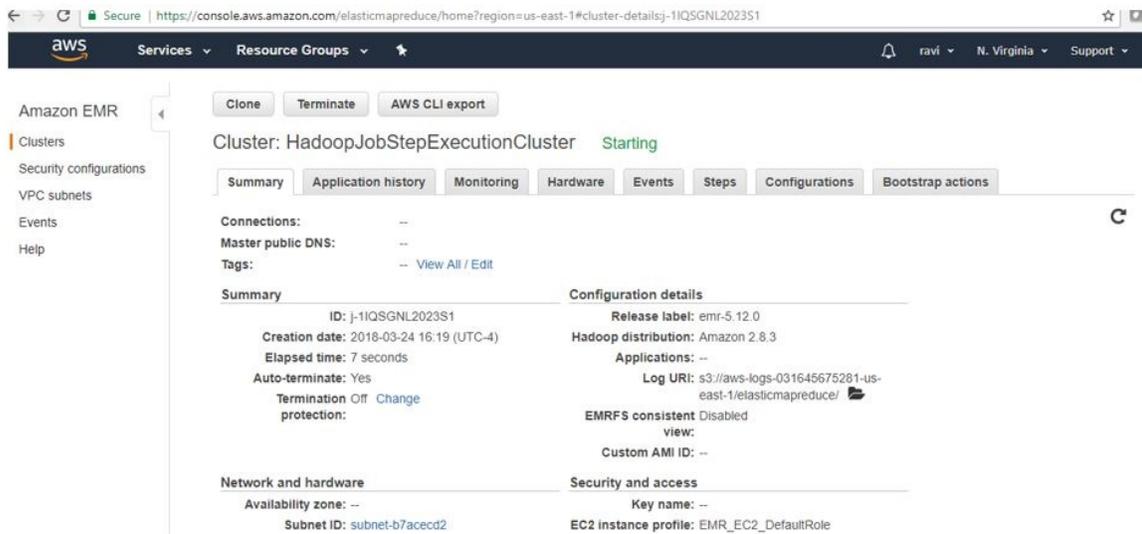


Figure 119: AWS Create Cluster1

10.4.1.1.10 Create a Hive Cluster

Login to AWS console, go to services and then select EMR. Click on Create Cluster. The cluster configuration provides details to complete. See, [Figure 120](#), [Figure 121](#), [Figure 122](#).

- Cluster name (Example: MyFirstCluster-Hive)
- Select Logging check box selected and provide S3 folder location
- Select launch mode as Cluster
- Complete software configuration (select hive application) and click on create cluster

10.4.1.1.10.1 Create a Hive Cluster - Screen shots

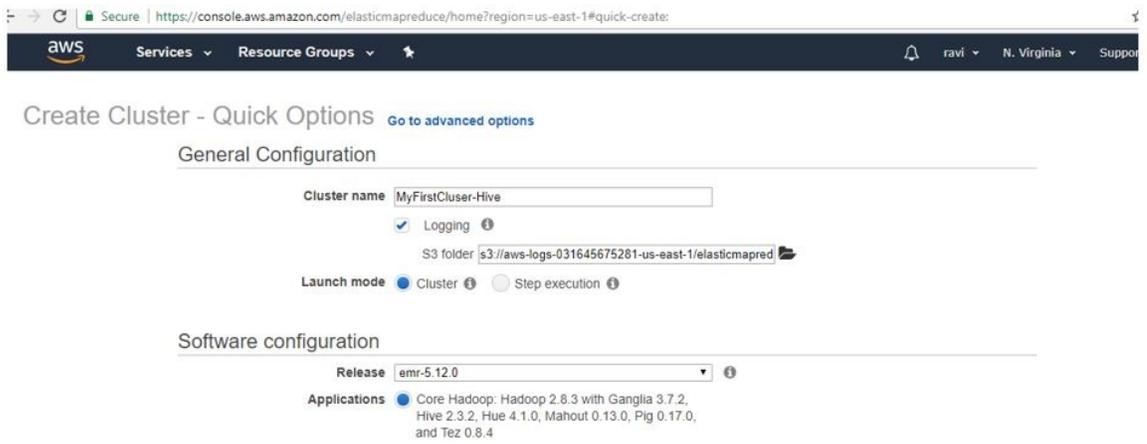


Figure 120: Hive Cluser

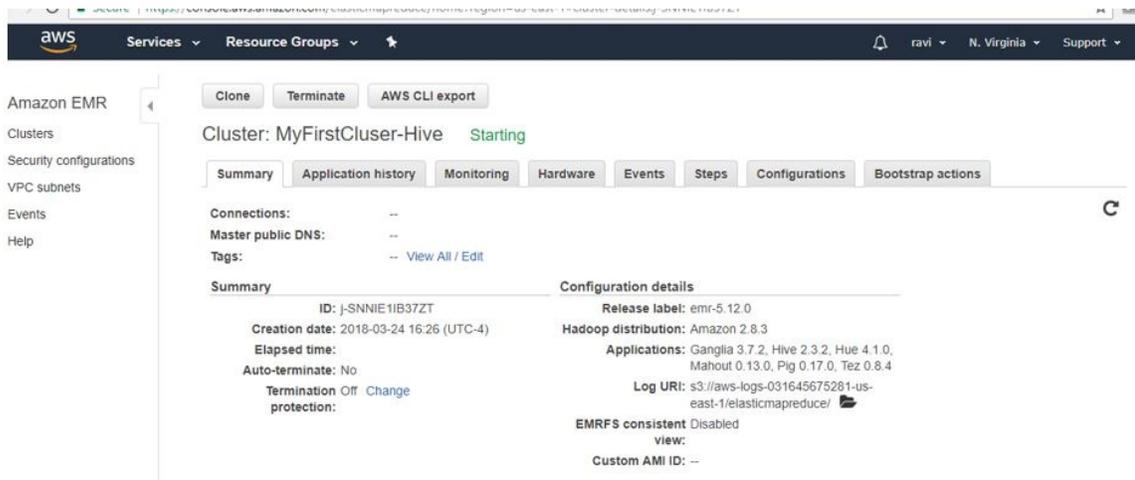


Figure 121: Hive Cluser1

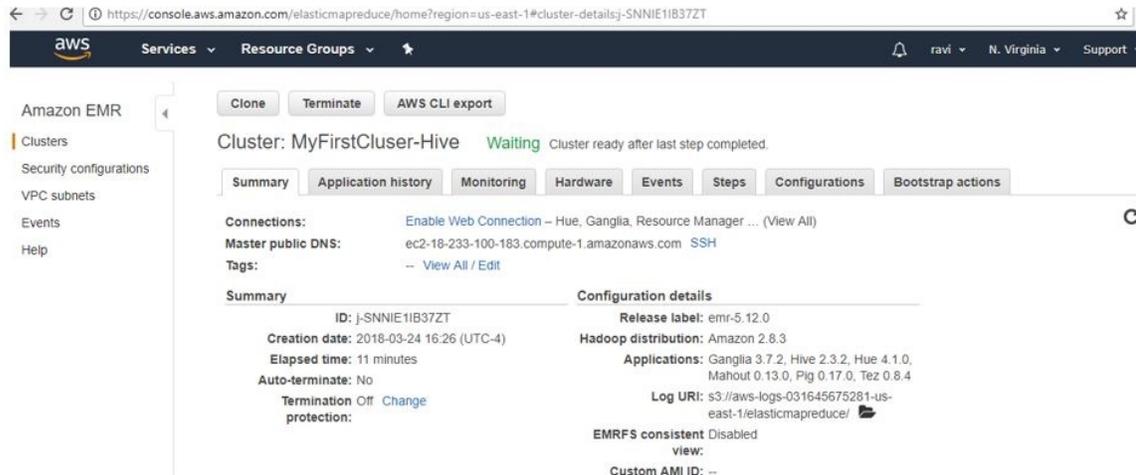


Figure 122: Hive Cluster2

10.4.1.1.11 Create a Spark Cluster

Login to AWS console, go to services and then select EMR. Click on Create Cluster. The cluster configuration provides details to complete. See, [Figure 123](#), [Figure 124](#), [Figure 125](#).

- Cluster name (Example: My Cluster - Spark)
- Select Logging check box selected and provide S3 folder location
- Select launch mode as Cluster
- Complete software configuration and click on create cluster
- Select application as Spark

10.4.1.1.11.1 Create a Spark Cluster - Screenshots

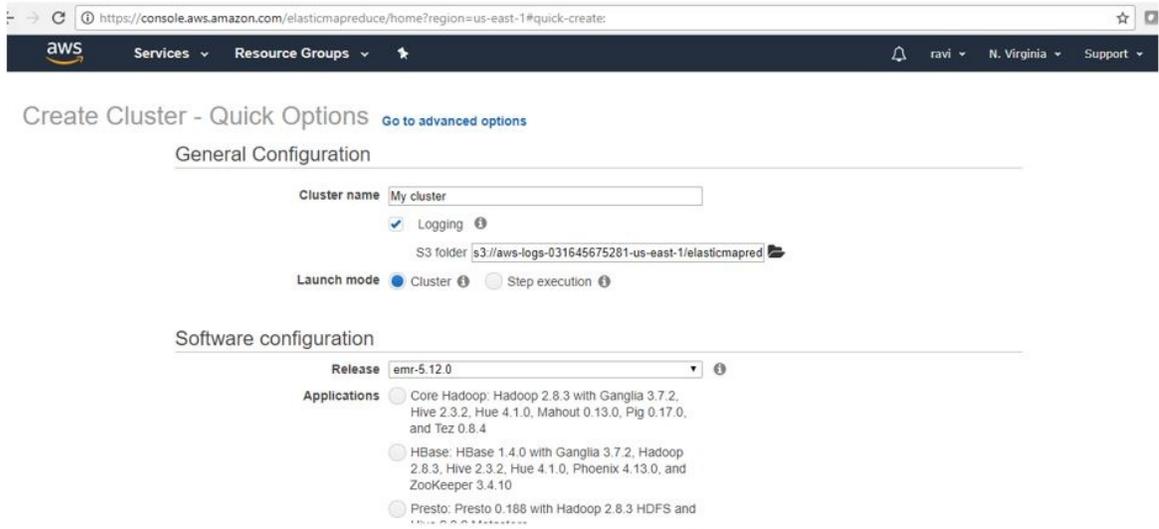


Figure 123: Spark Cluser



Figure 124: Spark Cluser

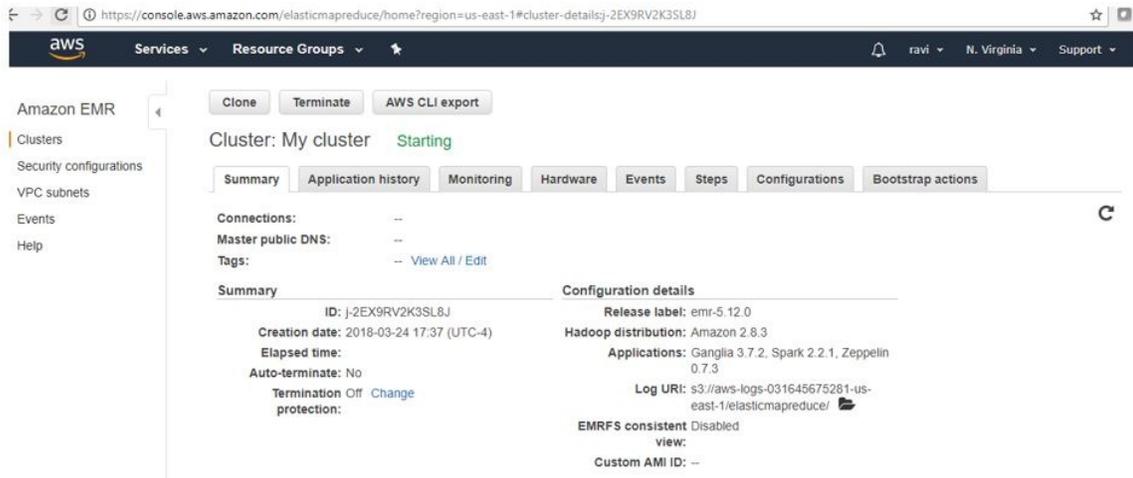


Figure 125: Spark Cluser

10.4.1.1.12 Run an example Spark job on an EMR cluster

10.4.1.1.12.1 Spark Job Description

You can submit Spark steps to a cluster as it is being created or to an already running cluster,

In this example we will execute a simple Python function on a text file using Spark on EMR. This is a standard word count application that will return the distinct words in the file along with the count of the number of times the words are present.

The Python file containing the application will be stored and referenced in a S3 bucket along with the text file being analyzed. The results of the Spark job will be returned to the same S3 bucket.

10.4.1.1.12.2 Creating the S3 bucket

```
aws s3 mb s3://test-analysis-bucket --region us-east-2
```

10.4.1.1.12.3 Copy files to S3

Create a `wordCount.py` file with the following code.

```
from __future__ import print_function
from pyspark import SparkContext
import sys
if __name__ == "__main__":
```

```

if len(sys.argv) != 3:
    print("Usage: testjob ", file=sys.stderr)
    exit(-1)
sc = SparkContext(appName="MyTestJob")
dataTextAll = sc.textFile(sys.argv[1])
dataRDD = dataTextAll.map(lambda x: x.split(",")).map(lambda y: (str(y[0]), float(y[1]))).reduceByKey(lambda a, b: a + b)
dataRDD.saveAsTextFile(sys.argv[2])
sc.stop()

```

You can then sync the folder you stored .py file in to your S3 bucket folder.

```
aws s3 sync your-local-folder-path s3://test-analysis-bucket/SparkTutorial
```

Store a text file locally and use the S3 sync function to make it available in your S3 bucket.

```
aws s3 sync your-local-folder-path s3://test-analysis-bucket/SparkTutorial/Input
```

10.4.1.1.12.4 Execute the Spark job on a running cluster

Using your cluster id and the paths within your S3 bucket run the following command (this assumes you have a cluster up and running).

```

aws emr add-steps --cluster-id your-cluster-id \
--steps Type=spark,Name=SparkWordCountApp,\
Args=[--deploy-mode,cluster,--master,yarn,\
--conf,spark.yarn.submit.waitAppCompletion=false,\
--num-executors,2,--executor-cores,2,--executor-memory,1g,\
s3://your-bucket/SparkTutorial/Python/WordCount.py,\
s3://your-bucket/SparkTutorial/Python/Input/input.txt,\
s3://your-bucket/SparkTutorial/Python/Output/]

```

10.4.1.1.12.5 Execute the Spark job while creating clusters

We can also run the same Spark step during the creation of a cluster using the following command (assumes you have already done pre-steps to creating an EMR cluster).

In this case the EMR cluster will spin up, run the Spark job, persist the results to your S3 bucket, and then auto terminate.

```

aws emr create-cluster \
--name "Test-Kerberized-Spark-Cluster" \
--release-label emr-5.17.0 \
--instance-type m4.large \
--instance-count 3 \
--use-default-roles \
--ec2-attributes KeyName=your-key,SubnetId=subnet-d0169eaa \py
--security-configuration KerberosSecurityConfiguration \
--applications Name=Spark \
--kerberos-attributes Realm=EC2.INTERNAL,KdcAdminPassword=your-pw \
--steps Type=spark,Name=SparkWordCountApp,\
Args=[--deploy-mode,cluster,--master,yarn,\
--conf,spark.yarn.submit.waitAppCompletion=false,\
--num-executors,2,--executor-cores,2,--executor-memory,1g,\
s3://your-bucket/SparkTutorial/Python/WordCount.py,\

```

```
s3://your-bucket/SparkTutorial/Python/Output/output.txt,\ns3://your-bucket/SparkTutorial/Python/Output/],\nActionOnFailure=CONTINUE \n--auto-terminate
```

10.4.1.1.12.6 View the results of the Spark job

You can use the AWS Console to view the results of the Spark Job.

Go to the AWS Console (ensure that the URL references your default region)

[AWS Console](#)

Navigate to the S3 bucket and folder you specified for the output (see [Figure 126](#))

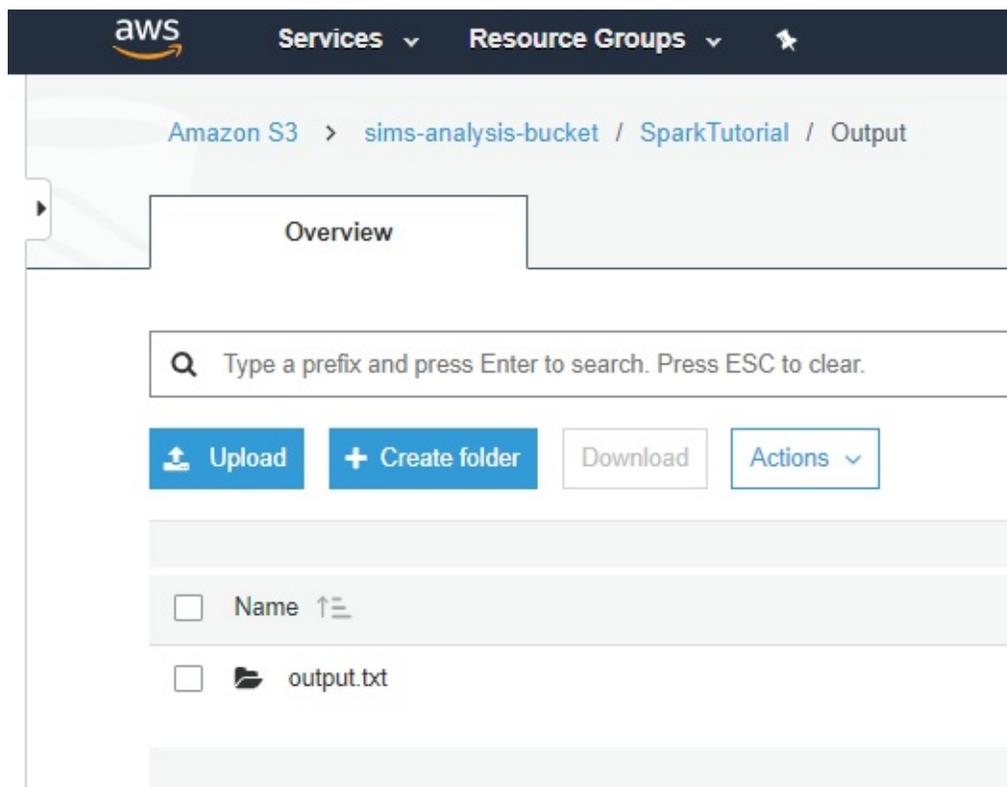


Figure 126: Set up EMR [77]

10.4.1.1.13 Conclusion

AWS EMR is a powerful tool for distributive processing. It is easy to use from wither the command line utilizing AWS CLI or through the AWS Console web interface.

10.4.2 TWISTER

10.4.2.1 Twister2

10.4.2.1.1 Introduction

Twister2[78] provides a data analytics hosting environment where it supports different data analytics including streaming, data pipelines and iterative computations. The functionality of Twister2 is similar to other Big data frameworks such as Apache Spark and Apache Flink. But there are a few key differences which differentiates Twister2 from other frameworks. Unlike many other big data systems that are designed around user APIs, Twister2 is built from bottom up to support different APIs and workloads. The aim of Twister2 is to develop a complete computing environment for data analytics.

One major goal of Twister2 is to provide independent components, that can be used by other big data systems and evolve separately. To this end Twister2 supports a composable architecture where developers can easily replace a small component in the system with a new implementation very easily. For example the resource scheduling layer has several implementations it supports, Kubernetes, Mesos, Slurm, Nomad and a standalone implementation, If a user wants to add support for another resources scheduler such as Yarn they can easily do so by implementing the well defined interfaces.

Twister2 supports both batch and streaming applications. Unlike other big data frameworks which either support batch or streaming in the core and develop the other on top of that, Twister2 natively supports both batch and streaming. Which allows Twister2 to make separate optimizations for each type.

Twister2 project is still less than 2 years old and still in it's early stages and going through rapid development to complete its functionality. It is an Open Source project which is licenced under the Apache 2.0[79]

10.4.2.1.2 Twister2 API's

Twister2 provides users with 3 levels on API's which can be used to write applications. The 3 API levels are shown in Figure [Figure 127](#).

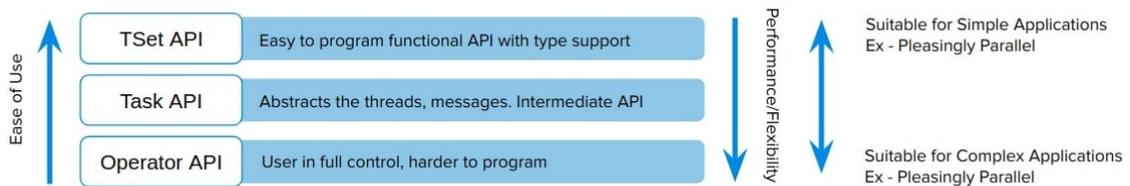


Figure 127: Twister2 API's

As shown in [Figure 127](#) each API level has different levels of abstraction and programming complexities. TSet API is the most high level in Twister2 which in some ways is similar to the RDD API in Apache Spark or DataSet API in Apache Flink. If the user wants more control over the application development they can opt to use a more lower level API's.

10.4.2.1.2.1 TSet API

TSet API is the most abstract API provided by Twister2. This allows user to develop their programs at the data layer, similar to the programming model of Apache Spark. Similar to RDD in Spark users can perform operations on top of TSet objects which will be automatically parallelized by the framework. To get a slight understanding of the Tset API take a look at the abstract example given on how TSet API can be used to implement KMeans algorithm.

```
public class KMeansJob extends TaskWorker {
    //.....
    @Override
    public void execute() {
        //.....
        TSet<double[][]> points = TSetBuilder.newBuilder(config).createSource(new Source<double[][]>() {
            //Code for source function to read data points
        }).cache();

        TSet<double[][]> centroids = TSetBuilder.newBuilder(config).createSource(new Source<double[][]>() {
            //Code for source function to read centers (or generate random centers)
        }).cache();

        for (
            int i = 0;
            i < iterations; i++) {
            TSet<double[][]> KmeansTSet = points.map(new MapFunction<double[][], double[][]>() {
                //Code for kmeans calculation, this will have access to the centroids which are passed in
            });
            KmeansTSet.addInput("centroids", centroids);

            Link<double[][]> allReduced = KmeansTSet.allReduce();
            TSet<double[][]> newCentroids = allReduced.map(new MapFunction<double[][], Object>() {
                /* Code that produces the new centers for the next iteration. The allReduce will result in
                 a sum or all the centers sent by each worker so this map function simply needs to compute the
                 average to get the new centers
                */
            });
            centroids.override(newCentroids);
        }
        //.....
    }
}
```

When programming at the TSet API level the developer does need to handle any information related to task and communications.

Note: The TSet API is currently under development and has not been released yet and therefore the API may change from what was discussed in this section, anyone who is interested can follow the development progress or contribute to the project through the GitHub repo[79].

10.4.2.1.2.2 Task API

The Task API allows developers to create their application at the Task level. The developer is responsible of managing task level details when developing at this API level, the upside of using the Task API is that it is more flexible than the TSet API so it allows developers to add custom optimizations to the application code. The TSet API is built on top of the Task API therefore the added layer of abstraction is bound to add slightly more overheads to the runtime, which you might be able to avoid by directly coding at the Task API level.

To get a better understanding of the Task API take a look at how the classic map reduce problem word count is implemented at using the Task API in the following code segment. This is only a portion of the example code, you can find the complete code for the example at[80].

```
public class WordCountJob extends TaskWorker {
//.....
@Override
public void execute() {
// source and aggregator
WordSource source = new WordSource();
WordAggregator counter = new WordAggregator();

// build the task graph
TaskGraphBuilder builder = TaskGraphBuilder.newBuilder(config);
builder.addSource("word-source", source, 4);
builder.addSink("word-aggregator", counter, 4).keyedReduce("word-source", EDGE,
new ReduceFn(Op.SUM, DataType.INTEGER), DataType.OBJECT, DataType.INTEGER);
builder.setMode(OperationMode.BATCH);

// execute the graph
DataFlowTaskGraph graph = builder.build();
ExecutionPlan plan = taskExecutor.plan(graph);
taskExecutor.execute(graph, plan);
}
//.....
}
```

More Task API examples can be found in Twister2 documentations[81].

10.4.2.1.3 Operator API

The lowest level API provided by Twister2 is the Operator API, this allows developers to develop applications at the communication level. However since this API only abstracts out communication operations, details such as task management need to be handled by the application developer. Again similar to the Task API this provides the developer with more flexibility to create more optimized applications, at the cost of being harder to program. Twister2 supports a variety of communication patterns, known as collective communications in the HPC world. These communications are highly optimized using various routing patterns to reduce the number of communication calls that go through the network to provide users with a extremely efficient Operator API. The following list show the communication operations that are supported by Twister2. You can find more information on each or these operations in the Twister2 documentation[82].

- Reduce
- Gather
- AllReduce
- AllGather
- Partition
- Broadcast
- Keyed Reduce
- Keyed Partition
- Keyed Gather

Initial Performance comparisons that are discussed in[83] show how Twister2 out performs popular frameworks such Apache Flink, Apache Spark and Apache Strom in many areas. For example the [Figure 128](#) shows a comparision between Twister2, MPI and Apache Spark versions of KMeans algorithm, please note that the graph is in logarithmic scale

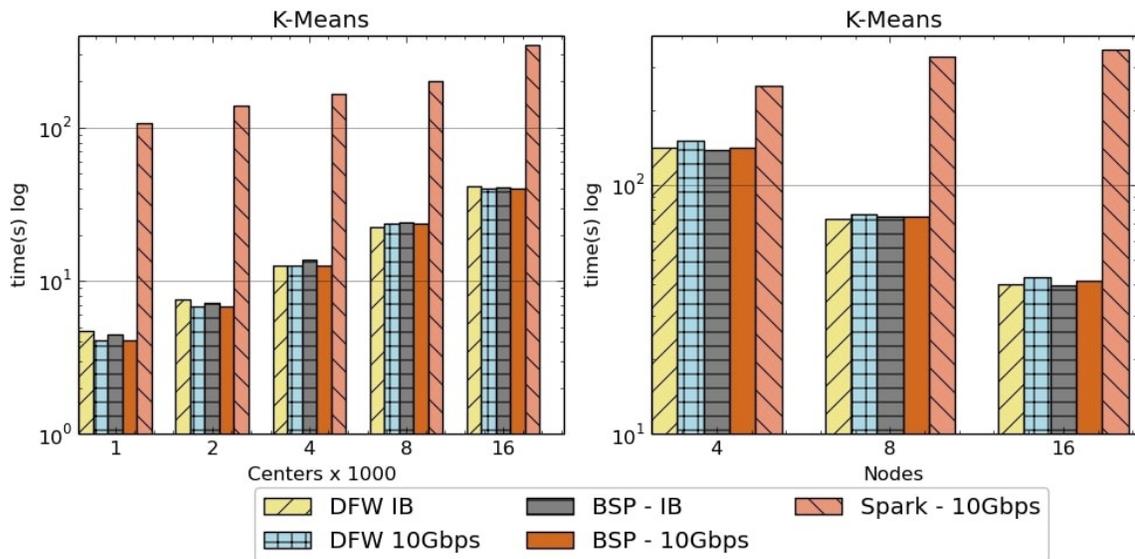


Figure 128: Kmeans Performance Comparison[84]

Notation : * **DFW** refers to Twister2 * **BSP** refers to MPI (OpenMPI)

This shows that Twister2 performs around ~10x faster than Apache Spark for KMeans. And that it is on par with implementations done using OpenMPI which is a widely used HPC framework.

10.4.2.1.3.1 Resources

- <http://www.iterativemapreduce.org/>
- <http://www.cs.allegeny.edu/sites/amohan/teaching/CMPSC441/paper10.pdf>
- <https://twister2.gitbook.io/twister2/>
- <http://dsc.soic.indiana.edu/publications/Twister2.pdf>
- <https://www.computer.org/csdl/proceedings/cloud/2018/7235/00/723501a38abs.html>

10.4.2.2 Twister2 Installation 🍄

10.4.2.2.1 Prerequisites

Because Twister2 is still in the early stages of development a binary release is not available as of yet, therefore to try out Twister2 users need to first build the binaries from the source code.

- Operating System :
 - Twister2 is tested and known to work on,
 - Red Hat Enterprise Linux Server release 7
 - Ubuntu 14.05, Ubuntu 16.10 and Ubuntu 18.10
- Java (Jdk 1.8) Covered in Section [\[s:hadoop-local-installation\]](#).
- G++ Compiler `sudo apt-get install g++`
- Maven Installation Explained in Section [Maven](#)
- OpenMPI Installation Explained in Section [OpenMPI](#)
- Bazel Build Installation Explained in Section [Bazel](#)
- Additional Libraries Explained in Section [Twister Extra](#)

10.4.2.2.1.1 Maven Installation

Execute the following commands to install Maven locally.

```
mkdir -p ~/cloudmesh/bin/maven
cd ~/cloudmesh/bin/maven
wget http://mirrors.ibiblio.org/apache/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz
tar xzf apache-maven-3.5.2-bin.tar.gz
```

Adding environmental variables

```
emacs ~/.bashrc
```

Add the following line at the end of the file.

```
MAVEN_HOME=~/.cloudmesh/bin/maven/apache-maven-3.5.2
PATH=$MAVEN_HOME/bin:$PATH
export MAVEN_HOME PATH

source ~/.bashrc
```

10.4.2.2.1.2 OpenMPI Installation

When you compile Twister2 it will automatically download and compile

OpenMPI 3.1.2 with it. If you don't like this version of OpenMPI and wants to use your own version, you can compile OpenMPI using following instructions.

- We recommend using OpenMPI 3.1.2
- Download OpenMPI 3.0.0 from <https://download.openmpi.org/release/open-mpi/v3.1/openmpi-3.1.2.tar.gz>
- Extract the archive to a folder named openmpi-3.1.2
- Also create a directory named build in some location. We will use this to install OpenMPI
- Set the following environment variables

```
BUILD=<path-to-build-directory>
OMPI_312=<path-to-openmpi-3.1.2-directory>
PATH=$BUILD/bin:$PATH
LD_LIBRARY_PATH=$BUILD/lib:$LD_LIBRARY_PATH
export BUILD OMPI_312 PATH LD_LIBRARY_PATH
```

- The instructions to build OpenMPI depend on the platform. Therefore, we highly recommend looking into the `$OMPI_1101/INSTALL` file. Platform specific build files are available in `$OMPI_1101/contrib/platform` directory.
- In general, please specify `--prefix=$BUILD` and `--enable-mpi-java` as arguments to configure script. If Infiniband is available (highly recommended) specify `--with-verbs=<path-to-verbs-installation>`. Usually, the path to verbs installation is `/usr`. In summary, the following commands will build OpenMPI for a Linux system.

```
cd $OMPI_312
./configure --prefix=$BUILD --enable-mpi-java
make -j 8;make install
```

- If everything goes well `mpirun --version` will show `mpirun (Open MPI) 3.1.2`. Execute the following command to install `$OMPI_312/ompi/mpi/java/java/mpi.jar` as a Maven artifact.

```
mvn install:install-file -DcreateChecksum=true -Dpackaging=jar -Dfile=$OMPI_312/ompi/mpi/java/java/mpi.jar -DgroupId=ompi
```

10.4.2.2.1.3 Install Extras

Install the other requirements as follows,

```
sudo apt-get install g++ git build-essential automake cmake libtool-bin zip
libunwind-setjmp0-dev zlib1g-dev unzip pkg-config python-setuptools -y sudo
apt-get install python-dev python-pip
```

Now you have successfully installed the required packages. Let us compile Twister2.

10.4.2.2.1.4 Compiling Twister2

Now lets get a clone of the source code.

```
git clone https://github.com/DSC-SPIDAL/twister2.git
```

You can compile the Twister2 distribution by using the bazel target as follows.

```
cd twister2
bazel build --config=ubuntu scripts/package:tarpkgs
```

This will build twister2 distribution in the file

```
bazel-bin/scripts/package/twister2-client-0.1.0.tar.gz
```

If you would like to compile the twister2 without building the distribution packages use the command

```
bazel build --config=ubuntu twister2/...
```

For compiling a specific target such as communications

```
bazel build --config=ubuntu twister2/comms/src/java:comms-java
```

10.4.2.2.1.5 Twister2 Distribution

After you've build the Twister2 distribution, you can extract it and use it to submit jobs.

```
cd bazel-bin/scripts/package/
tar -xvf twister2-0.1.0.tar.gz
```

10.4.2.3 Twister2 Examples

Twister documentation lists several examples[85] that users can leverage to better understand the Twister2 API's. Currently there are several Communication API examples and Task API examples available in the Twister2 documentation. In this section we will go through how an example can be executed with Twister2.

10.4.2.3.1 Submitting a Job

In order to run an example users need to submit the example to Twister2 using the `twister` command. This command is found inside the `bin` directory of the distribution.

Here is a description of the command

```
twister2 submit cluster job-type job-file-name job-class-name [job-args]
```

- `submit` is the command to execute
- `cluster` which resource manager to use, i.e. standalone, kubernetes, this should be the name of the configuration directory for that particular resource manager
- `job-type` at the moment we only support `jar`
- `job-file-name` the file path of the job file (the jar file)
- `job-class-name` name of the job class with a main method to execute

Here is an example command

```
./bin/twister2 submit standalone jar examples/libexamples-java.jar edu.iu.dsc.tws.examples.task.ExampleTaskMain -itr 80 -
```

In this command, `cluster` is `standalone` and has program arguments.

For this exercise we are using the `standalone` mode to submit a job. However Twister2 does support Kubernetes, Mesos, Slurm and Nomad resource schedulers if users want to submit jobs to larger cluster deployments.

10.4.2.3.2 Batch WordCount Example

In this section we will run a batch word count example from Twister2. This example only uses communication layer and resource scheduling layer. The threads are managed by the user program.

The example code can be found in

```
twister2/examples/src/java/edu/iu/dsc/tws/examples/basic/batch/wordcount/
```

When we install Twister2, it will compile the examples. Lets go to the installation directory and run the example.

```
cd bazel-bin/scripts/package/twister2-dist/  
./bin/twister2 submit standalone jar examples/libexamples-java.jar edu.iu.dsc.tws.examples.batch.wordcount.WordCountJob
```

This will run 4 executors with 8 tasks. So each executor will have two tasks. At the first phase, the 0-3 tasks running in each executor will generate words and after they are finished, 5-8 tasks will consume those words and create a count.

10.4.3 HADOOP RDMA

Acknowledgement: This section was copied and modified with permission from <https://www.chameleoncloud.org/appliances/17/docs/>

In Chameleon cloud it is possible to launch a virtual Hadoop cluster on bare-metal InfiniBand nodes with SR-IOV.

The CentOS 7 SR-IOV RDMA-Hadoop is based on a CentOS 7 Virtual Machine image, a VM startup script and a Hadoop cluster launch script, so that users can launch VMs with SR-IOV in order to run RDMA-Hadoop across these VMs on SR-IOV enabled InfiniBand clusters.

- Image name: CC-CentOS7-RDMA-Hadoop
- Default user account: cc
- Remote access: Key-Based SSH
- Root access: passwordless sudo from the cc account
- Chameleon admin access: enabled on the ccadmin account
- Cloud-init enabled on boot: yes
- Repositories (Yum): EPEL, RDO (OpenStack)
- Installed packages:
- Rebuilt kernel to enable IOMMU
- Mellanox SR-IOV drivers for InfiniBand
- KVM hypervisor
- Standard development tools such as make, gcc, gfortran, etc.
- Config management tools: Puppet, Ansible, Salt
- OpenStack command-line clients
- Included VM image name: chameleon-rdma-hadoop-appliance.qcow2
- Included VM startup script: start-vm.sh
- Included Hadoop cluster launch script: launch-hadoop-cluster.sh
- Default VM root password: nowlab

We refer to the chameleon cloud bare metal user guide for documentation on how to reserve and provision resources using the appliance of CC-CentOS7-RDMA-Hadoop.

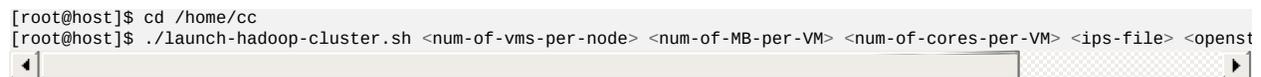
 link missing

10.4.3.1 Launching a Virtual Hadoop Cluster on Bare-metal InfiniBand Nodes with SR-IOV on Chameleon

We provide a CentOS 7 VM image (chameleon-rdma-hadoop-appliance.qcow2) and a Hadoop cluster launch script (launch-hadoop-cluster.sh) to facilitate users to setup Virtual Hadoop Clusters effortlessly.

First, launch bare-metal nodes using the RDMA-Hadoop Appliance and select one of the nodes as the bootstrap node. This node will serve as the host for the master node of the Hadoop cluster and will also be used to setup the entire cluster. Now, ssh to this node. Before you can launch the cluster, you have to download your OpenStack credentials file (see how to download your credentials file). Then, create a file (henceforth referred to as ips-file) with the ip addresses of the bare-metal nodes you want to launch your Hadoop cluster on (excluding the bootstrap node), each on a new line. Next, run these commands as root:

```
[root@host]$ cd /home/cc
[root@host]$ ./launch-hadoop-cluster.sh <num-of-vm-per-node> <num-of-MB-per-VM> <num-of-cores-per-VM> <ips-file> <openst
```



The launch cluster script will launch VMs for you, then install and configure Hadoop on these VMs. Note that when you launch the cluster for the first time, a lot of initialization is required. Depending on the size of your cluster, it may take some time to setup the cluster. After the cluster setup is complete, the script will print an output telling you that the cluster is setup and how you can connect to the Hadoop master node. Note that the minimum required memory for each VM is 8,192 MB. The Hadoop cluster will already be setup for use. For more details on how to use the RDMA-Hadoop package to run jobs, please refer to its user guide.

10.4.3.2 Launching Virtual Machines Manually

We provide a CentOS 7 VM image (chameleon-rdma-hadoop-appliance.qcow2) and a VM startup script (start-vm.sh) to facilitate users to launch VMs manually. Before you can launch a VM, you have to create a network port. To do this, source your OpenStack credentials file (see how to download your credentials file) and run this command:

```
[user@host]$ neutron port-create sharednet1
```

Note the MAC address and IP address are in the output of this command. You should use this MAC address while launching a VM and the IP address to ssh to the VM. You also need the PCI device ID of the virtual function that you want to assign to the VM. This can be obtained by running "lspci | grep Mellanox" and looking for the device ID (with format - XX:XX.X) of one of the virtual functions as shown next:

```
[cc@host]$ lspci | grep Mellanox
03:00.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3]
03:00.1 Network controller: Mellanox Technologies MT27500/MT27520 Family [ConnectX-3/ConnectX-3 Pro Virtual Function]
...
```

The PCI device ID of the Virtual Function is 03:00:1 in the previous example.

Now, you can launch a VM on your instance with SR-IOV using the provided VM startup script and corresponding arguments as follows with the root account.

```
[root@host]$ ./start-vm.sh <vm-mac> <vm-ifname> <virtual-function-device-id>
```

Please note that and are the ones you get from the outputs of previous commands. And is the name of VM virtual NIC interface. For example:

```
[root@host]$ ./start-vm.sh fa:16:3e:47:48:00 tap0 03:00:1
```

You can also edit corresponding fields in VM startup script to change the number of cores, memory size, etc.

You should now have a VM running on your bare metal instance. If you want to run more VMs on your instance, you will have to create more network ports. You will also have to change the name of VM virtual NIC interface to different ones (like tap1, tap2, etc.) and select different device IDs of virtual functions.

10.4.3.3 Extra Initialization when Launching Virtual Machines

In order to run RDMA-Hadoop across VMs with SR-IOV, and keep the size of VM image small, extra initialization will be executed when launching VM automatically, which includes:

- Detect Mellanox SR-IOV drivers, download and install it if nonexistent
- Detect Java package installed, download and install if non-existent
- Detect RDMA-Hadoop package installed, download and install if non-existent

After finishing the extra initialization procedure, you should be able to run Hadoop jobs with SR-IOV support across VMs. Note that this initialization will be done automatically. For more details about the RDMA-Hadoop package, please refer to its user guide.

10.4.3.4 Important Note for Tearing Down Virtual Machines and Deleting Network Ports

Once you are done with your experiments, you should kill all the launched VMs and delete the created network ports. If you used the launch-hadoop-cluster.sh script to launch VMs, you can do this by running the kill-vm.sh script as shown next. This script will kill all launched VMs and also delete all the created network ports.

```
[root@host]$ cd /home/cc
[root@host]$ ./kill-vm.sh <ips-file> <openstack-credentials-file>
\end{verbatim}
```

If you launched VMs using the start-vm.sh script, you should first manually kill all the VMs. Then, delete all the create

```
[user@host]$ neutron port-delete PORT
```

Please note that it is important to delete unused ports after experiments.

11 CONTAINER

11.1 INTRODUCTION TO CONTAINERS



Learning Objectives

- Knowing what a container is.
 - Differentiating Containers from Virtual Machines.
 - Understanding the historical aspects that lead to containers.
-

This section covers an introduction to containers that is split up into four parts. We discuss microservices, serverless computing, Docker, and kubernetes.

11.1.1 Motivation - Microservices

We discuss the motivation for containers and contrast them to virtual machines. Additionally we provide a motivation for containers as they can be used to microservices.



[Container 11:01 Container A](#)

11.1.2 Motivation - Serverless Computing

We enhance our motivation while contrasting containers and microservices while relating them to serverless computing. We anticipate that serverless computing will increase in importance over the next years



[Container 15:08 Container B](#)

11.1.3 Docker

In order for us to use containers, we go beyond the historical motivation that was

introduced in a previous section and focus on Docker a predominant technology for containers on Windows, Linux, and macOS

 [Container 40:09 Container C](#)

11.1.4 Docker and Kubernetes

We continue our discussion about docker and introduce kubernetes, allowing us to run multiple containers on multiple servers building a cluster of containers.

 [Container 50:14 Container D](#)

11.2 DOCKER

11.2.1 Introduction to Docker

Docker is the company driving the container movement and the only container platform provider to address every application across the hybrid cloud. Today's businesses are under pressure to digitally transform but are constrained by existing applications and infrastructure while rationalizing an increasingly diverse portfolio of clouds, datacenters and application architectures. Docker enables true independence between applications and infrastructure and developers and IT ops to unlock their potential and creates a model for better collaboration and innovation. An overview of docker is provided at

- <https://docs.docker.com/engine/docker-overview/>

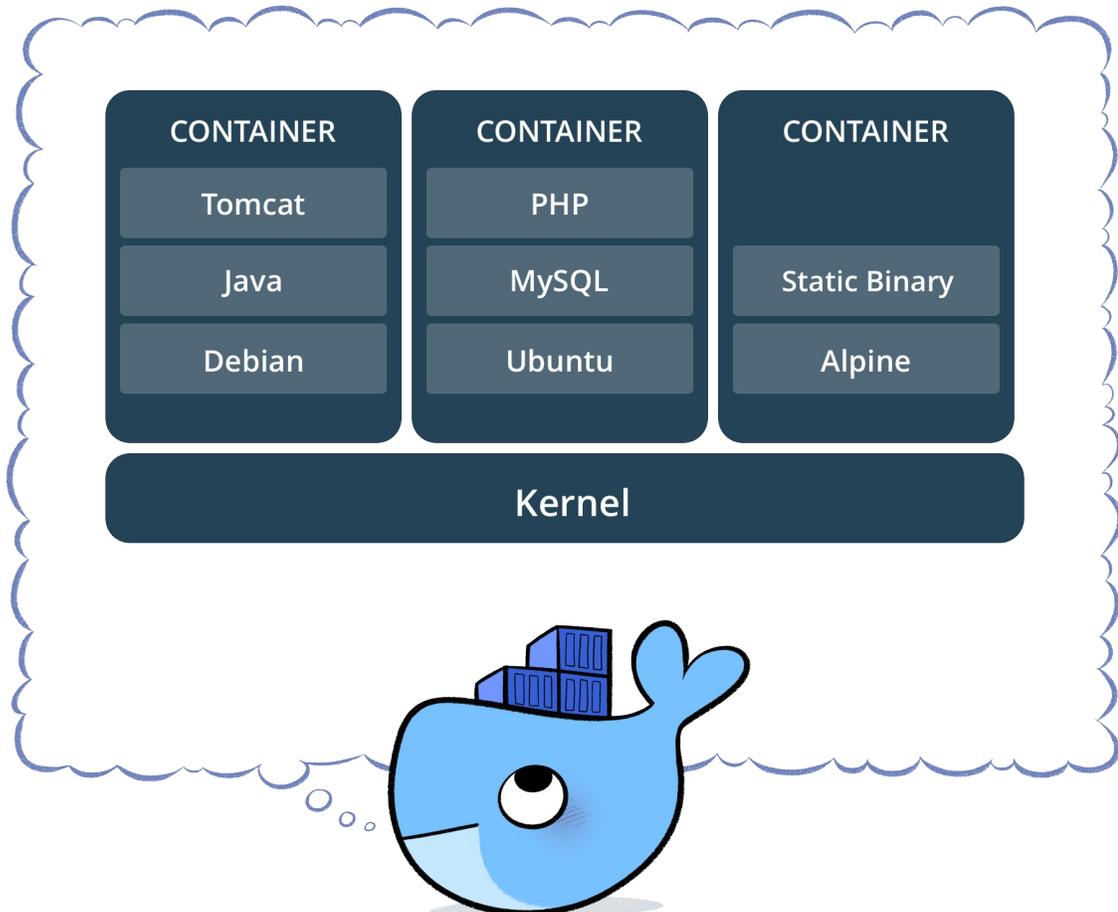


Figure 129: Docker Containers [\[Image Source\]](#) [86]

[Figure 129](#) shows how docker containers fit into the system ## Docker platform

Docker provides users and developers with the tools and technologies that are needed to manage their application development using containers. Developers can easily setup different environments for development, testing and production.

11.2.1.1 Docker Engine

The Docker engine can be thought of as the core of the docker runtime. The docker engine mainly provides 3 services. [Figure 130](#) shows how the docker engine is composed.

- A long running server which manages the containers
- A REST API
- A command line interface

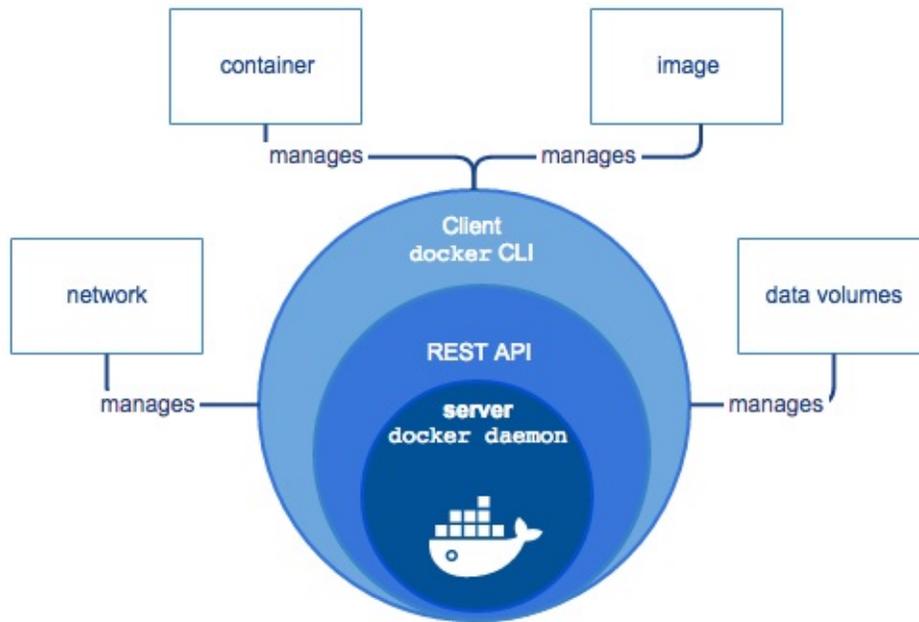


Figure 130: Docker Engine Component Flow [\[Image Source\]](#) [86]

11.2.1.2 Docker Architecture

The main concept of the docker architecture is based on the simple client-server model. Docker clients communicate with the Docker server also known as the Docker daemon to request various resources and services. The daemon manages all the background tasks that need to be performed to complete client requests. Managing and distributing containers, running the containers, building containers, etc. are responsibilities of the Docker daemon. [Figure 131](#) shows how the docker architecture is setup. The client module and server can run either in the same machine or in separate machines. In the latter case the communication between the client and server are done through the network.

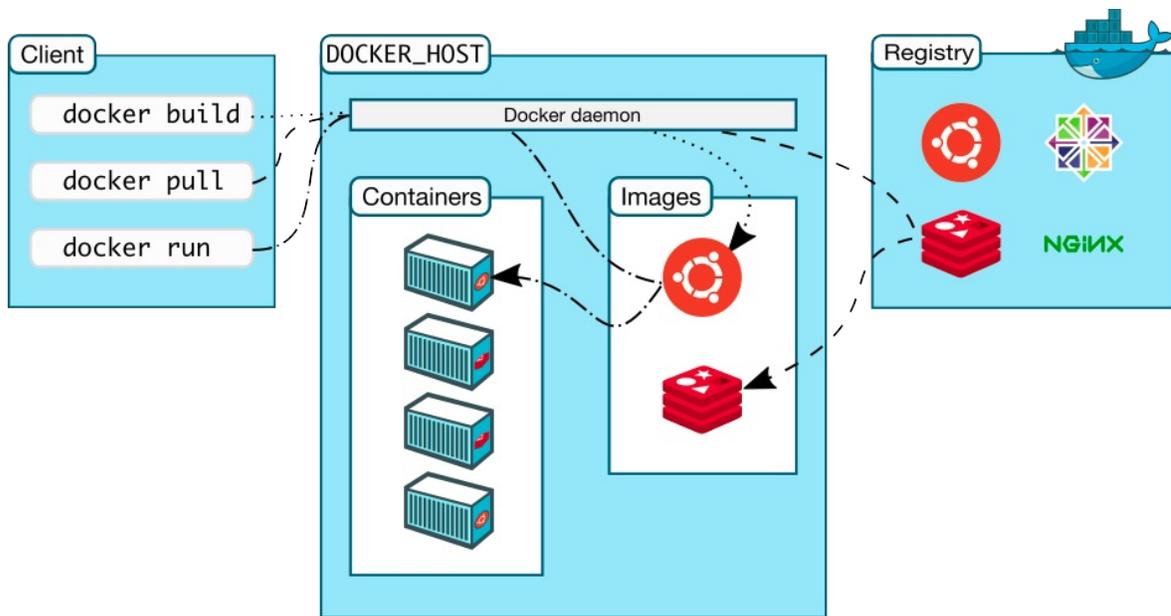


Figure 131: Docker Architecture [Image Source] [86]

11.2.1.3 Docker Survey

In 2016 Docker Inc. surveyed over 500 Docker developers and operations experts in various phases of deploying container-based technologies. The result is available in the *The Docker Survey 2016* as seen in [Figure 132](#).

- <https://www.docker.com/survey-2016>

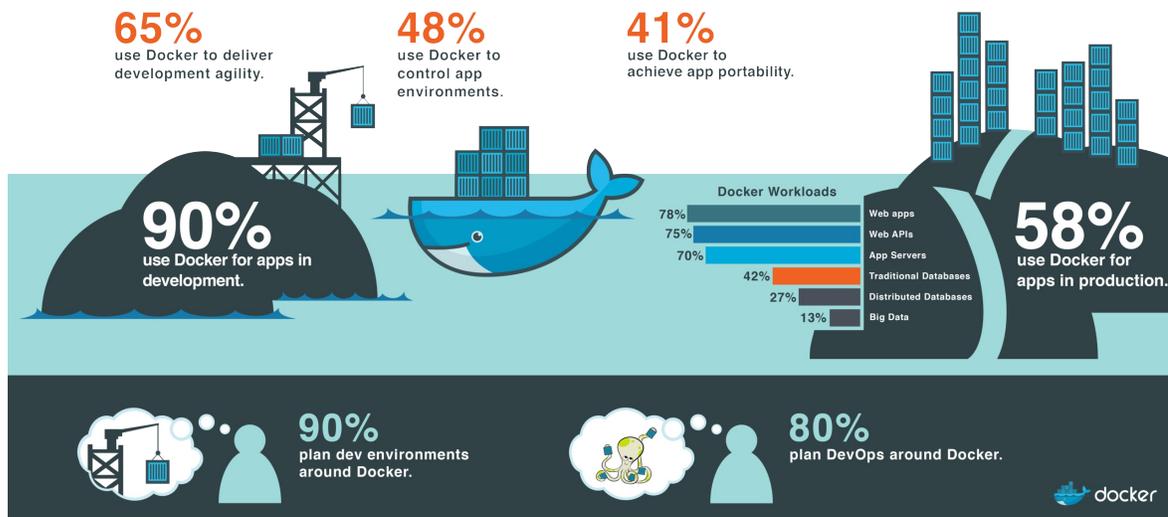


Figure 132: Docker Survey Results 2016 [Image Source] [86]

11.2.2 Running Docker Locally

⚠ Please verify if the instructions are still up to date. Rapid changes could mean they can be outdated quickly. Also we assume the ubuntu installations may have changed and may be different between 18.04 and 19.04.

The official installation documentation for docker can be found by visiting the following Web page:

- <https://www.docker.com/community-edition>

Here you will find a variety of packages, one of which will hopefully be suitable for your operating system. The supported operating systems currently include:

- OSX, Windows, CentOS, Debian, Fedora, Ubuntu, AWS, Azure

Please choose the one most suitable for you. For your convenience we provide you with installation instructions for OSX (Section [Docker on OSX](#)), Windows 10 (Section [Docker on Windows](#)) and Ubuntu (Section [Docker on ubuntu](#)).

11.2.2.1 Installation for OSX

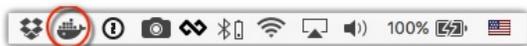
The docker community edition for OSX can be found at the following link

- <https://store.docker.com/editions/community/docker-ce-desktop-mac>

We recommend that at this time you get the version *Docker CE for MAC (stable)*

- <https://download.docker.com/mac/stable/Docker.dmg>

Clicking on the link will download a dmg file to your machine, that you then will need to install by double clicking and allowing access to the dmg file. Upon installation a `whale` in the top status bar shows that Docker is running, and you can access it via a terminal.



Docker integrated in the menu bar on OSX

11.2.2.2 Installation for Ubuntu

In order to install Docker community edition for Ubuntu, you first have to register the repository from where you can download it. This can be achieved as follows:

```
local$ sudo apt-get update
local$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  software-properties-common
local$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
local$ sudo apt-key fingerprint 0EBFCD88
local$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  local$(lsb_release -cs) \
  stable"
```

Now that you have configured the repository location, you can install it after you have updated the operating system. The update and install is done as follows:

```
local$ sudo apt-get update
local$ sudo apt-get install docker-ce
local$ sudo apt-get update
```

Once installed execute the following command to make sure the installation is done properly

```
local$ sudo systemctl status docker
```

This should give you an output similar to the next.

```
docker.service - Docker Application Container Engine
```

```
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
Active: active (running) since Wed 2018-10-03 13:02:04 EDT; 15min ago
   Docs: https://docs.docker.com
Main PID: 6663 (dockerd)
Tasks: 39
```

11.2.2.3 Installation for Windows 10

Docker needs Microsoft's Hyper-V to be enabled, but it will impact running the virtual machines

Steps to Install

- Download Docker for Windows(Community Edition) from the following link
<https://download.docker.com/win/stable/Docker%20for%20Windows%20In>
- Follow the Wizard steps in the installer
- Launch docker
- Docker usually launches automatically during windows startup.

11.2.2.4 Testing the Install

To test if it works execute the following commands in a terminal:

```
local$ docker version
```

You should see an output similar to

```
docker version

Client:
 Version:      17.03.1-ce
 API version:  1.27
 Go version:   go1.7.5
 Git commit:   c6d412e
 Built:        Tue Mar 28 00:40:02 2017
 OS/Arch:      darwin/amd64

Server:
 Version:      17.03.1-ce
 API version:  1.27 (minimum version 1.12)
 Go version:   go1.7.5
 Git commit:   c6d412e
 Built:        Fri Mar 24 00:00:50 2017
 OS/Arch:      linux/amd64
 Experimental: true
```

To see if you can run a container use

```
local$ docker run hello-world
```

Once executed you should see an output similar to

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a .....
Status: Downloaded newer image for
      hello-world:latest

Hello from Docker!
This message shows that your installation appears
to be working correctly.

To generate this message, Docker took the following
steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image
   from the Docker Hub.
3. The Docker daemon created a new container from that
   image which runs the executable that produces the
   output you are currently reading.
4. The Docker daemon streamed that output to the Docker
   client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu
container with:

local$ docker run -it ubuntu bash

Share images, automate workflows, and more with a
free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

11.2.3 Dockerfile

In order for us to build containers, we need to know what is in the container and how to create an image representing a container. To do this a convenient specification format called `Dockerfile` can be used. Once a `Dockerfile` is created, we can build images from it

We showcase here the use of a dockerfile on a simple example using a REST service.

This example is copied from the official docker documentation hosted at

- <https://docs.docker.com/get-started/part2/#publish-the-image>

11.2.3.1 Specification

It is best to start with an empty directory in which we create a Dockerfile.

```
local$ mkdir ~/cloudmesh/docker
local$ cd ~/cloudmesh/docker
```

Next, we create an empty file called `Dockerfile`

```
local$ touch Dockerfile
```

We copy the following contents into the Dockerfile and after that create a simple REST service

```
# Use an official Python runtime as a parent image
FROM python:3.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available
EXPOSE 80

# Run app.py when the container launches
CMD ["python", "app.py"]
```

We also create a `requirements.txt` file that we need for installing the necessary python packages

```
Flask
```

The example application we use here is a student info served via a RESTful service implemented using python flask. It is stored in the file `app.py`

```
from flask import Flask, jsonify
import os

app = Flask(__name__)

@app.route('/student/albert')
def alberts_information():
    data = {
        'firstname': 'Albert',
        'lastname': 'Zweistsein',
        'university': 'Indiana University',
        'email': 'albert@example.com'
    }
    return jsonify(**data)

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=80)
```

To build the container, we can use the following command:

```
local$ docker build -t students .
```

To run the service open a new window and cd into the directory where you code

is located. Now say

```
local$ docker run -d -p 4000:80 students
```

Your docker container will run and you can visit it by using the command

```
local$ curl http://localhost:4000/student/albert
```

To stop the container do a

```
local$ docker ps
```

and locate the id of the container, e.g., 2a19776ab812, and then run this

```
local$ docker stop 2a19776ab812
```

To delete the docker container image, you must first stop all instances using it and then remove the image. You can see the images with the command

```
local$ docker images
```

Then you can locate all containers using that image while looking in the IMAGE column or using a simple fgrep in case you have many images. Stop the containers using that image and that you can say

```
local$ docker rm 74b9b994c9bd
```

while the number is the container id

Once you killed all containers using that image, you can remove the image with the `rmi` command.

```
local$ docker rmi 8b3246425402
```

11.2.3.2 References

The reference documentation about docker files can be found at

- <https://docs.docker.com/engine/reference/builder/>

11.2.4 Docker Hub

Docker Hub is a cloud-based registry service which provides a “centralized

resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline” [86]. There are both private and public repositories. Private repository can only be used by people within their own organization.

Docker Hub is integrated into Docker as the default registry. This means that the docker pull command will initialize the download automatically from Docker Hub [87]. It allows users to download (pull), build, test and store their images for easy deployment on any host they may have [86].

11.2.4.1 Create Docker ID and Log In

A log-in is not necessary for pulling Docker images from the Hub but it is necessary for pushing images to dockerhub for sharing. Thus to store images on Docker hub you need to create an account by visiting [Docker Hub Web page](#). Dockerhub offers in general a free account, but it has restrictions. The free account allows you to share images that you distribute publicly, but it only allows one private Docker Hub Repository. In case you need more, you will need to upgrade to a paid plan.

For the rest of the tutorial we assume that you use the environment variable DOCKERHUB to indicate yourusername. It is easiset if you set it in your shell with

```
local$ export DOCKERHUB=<PUT YOUR DOCKER USERNAME HERE>
```

11.2.4.2 Searching for Docker Images

There are two ways to search for Docker images on Docker Hub:

One way is to use the Docker command line tool. We can open a terminal and run the *docker search* command. For example, the following command searches for centOS images:

```
local$ sudo docker search centos
```

you will see output similar to:

NAME	DESCRIPTION	STAR	OFFICIAL	AUTOMATED
------	-------------	------	----------	-----------

centos	Official CentOS	4130	[OK]
ansible/centos7	Ansible on Centos7	105	[OK]

...

If you do not want to use `sudo` with `docker` command each time you need to add the current user into the `docker` group. You can do that using the following command.

```
local$ sudo usermod -aG docker ${USER}
local$ su - ${USER}
```

This will prompt you to enter the password for the current user. Now you should be able to execute the previous command without using `sudo`.

Official repositories in `dockerhub` are public, certified repositories from vendors and contributors to Docker. They contain Docker images from vendors like Canonical, Oracle, and Red Hat that you can use as the basis to build your applications and services. There is one official repository in this list, the first one, *centos*.

The other way is to search via the *Web Search Box* at the top of the Docker web page by typing the keyword. The search results can be sorted by number of stars, number of pulls, and whether it is an official image. Then for each search result, you can verify the information of the image by clicking the *details* button to make sure this is the right image that fits your needs.

11.2.4.3 Pulling Images

A particular image (take `centos` as an example) can be pulled using the following command:

```
local$ docker pull centos
```

Tags can be used to specify the image to pull. By default the tag is `latest`, therefore the previous command is the same as the following:

```
local$ docker pull centos:latest
```

You can use a different tag:

```
local$ docker pull centos:6
```

To check the existing local docker images, run the following command:

```
local$ docker images
```

The results show:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	latest	26cb1244b171	2 weeks ago	195MB
centos	6	2d194b392dd1	2 weeks ago	195MB

11.2.4.4 Create Repositories

In order to push images to Docker Hub, you need to have a and account and create a repository.

When you first create a Docker Hub user, you see a *Get started with Docker Hub* screen, from which you can click directly into *Create Repository*. You can also use the *Create* menu to *Create Repository*. When creating a new repository, you can choose to put it in your Docker ID namespace, or that of any organization that you are in the owners team [88].

As an example, we created a repository `cloudtechnology` with the namespace `DOCKERHUB` (here `DOCKERHUB` is your docker hub username). Hence the full name is `DOCKERHUB/cloudtechnology`

11.2.4.5 Pushing Images

To push an image to the repository created, the following steps can be followed.

First, log into Docker Hub from the command line by specifying the username. If you encounter permission issues please use `sudo` in front of the command

```
$ docker login --username=DOCKERHUB
```

Enter the password when prompted. If everything worked you will get a message

similar to:

```
Login Succeeded
```

Second, check the image ID using:

```
$ docker images
```

the result looks similar to:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cloudmesh-nlp	latest	1f26a5f7a1b4	10 days ago	1.79GB
centos	latest	26cb1244b171	2 weeks ago	195MB
centos	latest	2d194b392dd1	2 weeks ago	195MB

Here, the the image with ID 1f26a5f7a1b4 is the one to push to Docker Hub. You can choose another image instead if you like.

Third, tag the image

```
$ docker tag 1f26a5f7a1b4 $DOCKERHUB/cloudmesh:v1.0
```

Here we have used a version number as a tag. However another good way of adding a tag is to use a keyword/tag that will help you understand what this container should be used in conjunction with, or what it represents.

Fourth, now the list of images will look something like

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cloudmesh-nlp	latest	1f26a5f7a1b4	10 d ago	1.79GB
\$DOCKERHUB/cloudmesh	v1.0	1f26a5f7a1b4	10 d ago	1.79GB
centos	latest	26cb1244b171	2 w ago	195MB
centos	latest	2d194b392dd1	2 w ago	195MB

Fifth, Now you can see an images under the name `$DOCKERHUB/cloudmesh`, we now need to push this image to the repository that we created on the docker hub website. For that execute the following command.

```
$ docker push $DOCKERHUB/cloudmesh
```

It shows something similar to, to make sure you can check on docker hub if the images that was pushed is listed in the repository that we created.

```
The push refers to repository [docker.io/$DOCKERHUB/cloudmesh]
18f9479cfc2c: Pushed
e9ddee98220b: Pushed
...
db584c622b50: Mounted from library/ubuntu
a94e0d5a7c40: Mounted from library/ubuntu
...
v1.0: digest: sha256:305b0f911077d9d6aab4b447b... size: 3463
```

Sixth, now the image is available on Docker Hub. Everyone can pull it since it is a public repository by using command:

```
$ docker pull USERNAME/cloudmesh
```

Please remember that the USERNAME is the username for the user that makes this image publically available. If you are the user you will see the value being the one from \$DOCKERHUB, If not you will see here the username of the user uploading the image

11.2.4.6 Resources

- The official [Overview of Docker Hub](#) [86]
- Information about using docker repositories can be found at [Repositories on Docker Hub](#) [88]
- [How to Use DockerHub](#) [87]
- [Docker Tutorial Series](#) [89]

11.2.5 Docker Compose

11.2.5.1 Introduction

Docker compose is a tool for defining and running multi-container using docker container to package them as an application. Docker compose uses a YAML file to specify the dependencies between the containers and their configuration. The nice feature is taht with a single command you create and start all the services from your configuration file and can maage the application including shutting it down.

Using docker compose includes a four-step process:

1. Define your application's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your application in a `docker-compose.yml` file so they can be specified in a single file and run with simple docker compose commands.
3. To start the application use the command `docker-compose up`
4. To shut down the application use the command `docker-compose down`

11.2.5.2 Installation

Docker compose can be installed on Windows 10 EDU/PRO, Linux, and macOS.

11.2.5.2.1 Install on MacOS

For macOS please go to this link to download a desktop version:

- <https://docs.docker.com/docker-for-mac/install/>

11.2.5.2.2 Install on Linux

On Linux you can run the command.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/
```

Please note that you use the newest version which can be found on the download Web page. After downloading, make sure that you apply executable permissions to binary:

```
sudo chmod +x /usr/local/bin/docker-compose
```

11.2.5.2.3 Install on Windows 10

11.2.5.2.3.1 System Requirements

In case you use Windows you need the following minimal requirements:

- Windows 10 64-bit
- Pro, Enterprise, or Education (Build 15063 or later).
- Hyper-V and Containers Windows features must be enabled.

The following hardware prerequisites are required to successfully run Client Hyper-V on Windows 10:

- 64 bit processor with Second Level Address Translation (SLAT)
- 4GB system RAM,
- BIOS-level hardware,
- Virtualization support must be enabled in the BIOS settings.

Go to this link to download a desktop version:

- <https://hub.docker.com/?overlay=onboarding>

11.2.5.2.4 Test the installation

It is important that you test your installation before you move forward. This can be done on the commandline with the command. More involved tests can be conducted while using the simple example depicted in this section.

```
$ docker-compose --version
docker-compose version 1.24.1, build 1110ad01
```

11.2.5.3 Docker Compose File Directives

To use docker compose, you will need a file that contains specifications of the containers and their dependencies. We will demonstrate this concept with a simple example.

We are starting a `redis` cache server, a `postgresql` database server, and containers `vote`, `result`, `worker`, `visualizer` to provide frontend and backend services that interact with the containers.

After you have reviewed the yml file, we will explain the different parts in more detail.

```
version: "3.7"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]

  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - "5000:80"
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure

  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - "5001:80"
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
```

```
volumes:
  - "/var/run/docker.sock:/var/run/docker.sock"
deploy:
  placement:
    constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
  db-data:
```

11.2.5.3.1 Configuration

11.2.5.3.1.1 build

The `build` attribute specifies either a string containing a path to the build context:

```
version: "3.7"
services:
  webapp:
    build: ./dir
```

11.2.5.3.1.2 context

The `context` attribute introduces either a path to a directory containing a Dockerfile, or a url to a git repository. This information is used during the build phase.

```
build:
  context: ./dir
```

11.2.5.3.1.3 ARGS

The `ARGS` attribute introduces environment variables accessible only during the build process.

```
ARG buildno
ARG gitcommithash
```

```
build:
  context: .
  args:
    buildno: 1
    gitcommithash: cdc3b19
```

11.2.5.3.1.4 command

The `command` attribute overrides the default command.

```
command: bundle exec thin -p 3000
```

11.2.5.3.1.5 depends_on

The `depends_on` attribute introduces dependencies between services. The container that depends on other containers, waits for them to become available. In the following example the web service depends on the db and redis services:

```
version: "3.7"
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

11.2.5.3.1.6 image

The `image` attribute specifies the image for the container. You can either use a repository/tag or a partial image ID to identify the image

```
image: redis
image: ubuntu:14.04
image: mongo
```

11.2.5.3.1.7 ports

The `ports` attribute exposes ports of the container. However, please note that the port mapping is incompatible with `network_mode: host`.

```
ports:
  - "3000"
  - "3000-3005"
  - "8000:8000"
  - "9090-9091:8080-8081"
  - "49100:22"
  - "127.0.0.1:8001:8001"
  - "127.0.0.1:5000-5010:5000-5010"
  - "6060:6060/udp"
```

11.2.5.3.1.8 volumes

The `volume` attribute mounts host paths or named volumes. A volume is specified as sub-options to a service.

You can mount a host path as part of a definition for a single service, and there is no need to define it in the top level `volumes` key.

11.2.5.4 Usages

11.2.5.4.1 Build A Service depending on MongoDB

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8080:8080"
    depends_on:
      - mongo
  mongo:
    image: mongo
    ports:
      - "27017:27017"
```

By default, `web` service can reach the `mongo` service by using the service's name as we configured the database URI to be

```
mongodb://mongo:27017.
```

To start the two docker containers you can use the command:

```
$ docker-compose up
```

We can close both docker containers with:

```
$ docker-compose down
```

11.3 DOCKER PAAS

11.3.1 Docker Clusters

In this section we present mechanisms for managing containers across multiple hosts. This includes docker swarm and kubernetes.

11.3.2 Docker Swarm

A swarm is a group of machines that are running Docker and are joined into a cluster. Docker commands are executed on a cluster by a swarm manager. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as *nodes*.

11.3.2.1 Terminology

In this section if a command is prefixed with `local$` it means the command is to be executed on your local machine. If it is prefixed with either `master` or `worker` that means the command is to be executed from within a virtual machine that was

created.

11.3.2.2 Creating a Docker Swarm Cluster

A swarm is made up of multiple nodes, which can be either physical or virtual machines. We use `master` as the name of the host that is run as master and `worker-1` as a host run as a worker, where the number indicates the i-th worker. The basic steps are:

1. run

```
master$ docker swarm init
```

to enable swarm mode and make your current machine a swarm manager,

2. then run

```
worker-1$ docker swarm join
```

on other machines to have them join the swarm as workers. Choose a tab described in next to see how this plays out in various contexts. We use VMs to quickly create a two-machine cluster and turn it into a swarm.

11.3.2.3 Create a Swarm Cluster with VirtualBox

In case you do not have access to multiple physical machines, you can create a virtual cluster on your machine with the help of virtual box. Instead of using `vagrant` we can use the built in `docker-machine` command to start several virtual machines.

If you do not have `virtualbox` installed on your machine install it on your machine. Additionally you would require `docker-machine` to be installed on your local machine. To install `docker-machine` on please follow instructions at the docker documentation at [Install Docker Machine](#)

To create the virtual machines you can use the command as follows:

```
local$ docker-machine create --driver virtualbox master
local$ docker-machine create --driver virtualbox worker-1
```

To list the VMs and get their ip addresses. Use this command to list the machines and get their IP addresses.

```
local$ docker-machine ls
```

11.3.2.4 Initialize the Swarm Manager Node and Add Worker Nodes

The first machine acts as the manager, which executes management commands and authenticates workers to join the swarm, and the second is a worker.

To instruct the first vm to become the master, first we need to login to the vm that was named `master`. To login you can use `ssh`, execute the following command on your local machine to login to the `master` vm.

```
local$ docker-machine ssh master
```

Now since we are inside the `master` vm we can configure this vm as the docker swarm manager. Execute the following command within the `master` vm in initialize swarm

```
master$ docker swarm init
```

If you get an error stating something similar to “could not choose an IP address to advertise since this system has multiple addresses on different interfaces”, use the following command instead. To find the IP address execute the command `ifconfig` and pick the ip address which is most similar to `192.x.x.x`.

```
master$ docker swarm init --advertise-addr 192.x.x.x
```

The output will look like this, where `IP-myvm1` is the ip address of the first vm

```
master$ Swarm initialized: current node (p6hmohoeuggtwj8xz91zbs5t) is now a manager.
```

To add a worker to this swarm, run the following command:

```
worker-1$ docker swarm join --token SWMTKN-1-5c3anju1pwx94054r3vx0v7j40byuggfu2cmesnx  
192.168.99.100:2377
```

To add a manager to this swarm, run `'docker swarm join-token manager'` and follow the instructions.

Now that we have the docker swarm manager up we can add worker machines to the swarm. The command that is printed in the output shown previously can be used to join workers to the manager. Please note that you need to use the output command that is generated when you run `docker swarm init` since the token values will

be different.

Now we need to use a separate shell to login to the worker vm that we created. Open up a new shell (or terminal) and use the following command to ssh into the `worker`

```
local$ docker-machine ssh worker-1
```

Once you are in the `worker` execute the following command to join `worker` to the swarm manager.

```
worker-1$ docker swarm join --token  
SWMTKN-1-5c3anju1pwx94054r3vx0v7j4obyuggfu2cmesnx 192.168.99.100:2377
```

The generic version of the command would be as follows, you need to fill in the correct values to values marked as '`<>`' to execute the command.

```
worker-1$ docker swarm join --token <token> <myvm ip>:<port>
```

You will see an output stating that this machine joined the docker swarm.

```
This node joined a swarm as a worker.
```

If you want to add another node as a manager to the current swarm you can execute the following command and follow the instructions. However this is not needed for this exercise.

```
newvm$ docker swarm join-token manager'
```

Run `docker-machine ls` to verify that `worker` is now the active machine, as indicated by the asterisk next to it.

```
local$ docker-machine ls
```

If the astrix is not present execute the following command

```
local$ sudo sh -c 'eval "$(docker-machine env worker-1)"; docker-machine ls'
```

The output will look similar to

NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
master	-	virtualbox	Running	tcp://192.168.99.100:2376		v18.06.1-ce	
worker-1	*	virtualbox	Running	tcp://192.168.99.102:2376		v18.06.1-ce	

11.3.2.5 Deploy the application on the swarm manager

Now we can try to deploy a test application. First we need to create a docker configuration file which we will name `docker-compose.yml`. Since we are in the vm we need to create the file using the terminal. follow the steps given next the create and save the file. First log into the `master`

```
local$ docker-machine ssh worker-1
```

Then,

```
master$ vi docker-compose.yml
```

This command will open an editor. Press the `Insert` button to enable editing and then copy paste the following into the document.

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - webnet
networks:
  webnet:
```

Then pres the `ECS` button and enter `:wq` to save and close the editor.

Once we have the file we can deploy the test application using the following command. which will be executed in the `master`

```
master$ docker stack deploy -c docker-compose.yml getstartedlab
```

To verify the services and associated containers have been distributed between both `master` and `worker`, execute the following command.

```
master$ docker stack ps getstartedlab
```

The output will look similar to

```
```bash ID NAME IMAGE NODE DESIRED STATE CURRENT STATE
ERROR PORTS wpqtkv69qbee getstartedlab_web.1 username/repo:tag worker-
1 Running Preparing 4 seconds ago whkiecyenuv0 getstartedlab_web.2
username/repo:tag master Running Preparing 4 seconds ago 13obecvxohh1
```

```
getstartedlab_web.3 username/repo:tag worker-1 Running Preparing 5 seconds ago
76srj0nflagi getstartedlab_web.4 username/repo:tag worker-1 Running Preparing 5 seconds ago
ymqoonad5c1f getstartedlab_web.5 username/repo:tag master Running Preparing 5 seconds ago
```

### 11.3.3 Docker and Docker Swarm on FutureSystems

This section is for IU students only that take classes with us.

This section introduces how to run Docker container on FutureSystems. Currently we have deployed Docker swarm on Echo.

#### 11.3.3.1 Getting Access

You will need an account on FutureSystems and be enrolled in an active project. To verify, try to see if you can log into `victor.futuresystems.org`. You need to be a member of a valid FutureSystems project, and had submitted an ssh public key via the FutureSystems portal.

For Fall 2018 classes at IU you need to be in the following project:

<https://portal.futuresystems.org/project/553>

If your access to the victor host has been verified, try to login to the docker swarm head node. To conveniently do this let us define some Linux environment variables to simplify the access and the material presented here. You can place them even in your `.bashrc` or `.bash_profile` so the information gets populated whenever you start a new terminal. If you directly edit the files make sure to execute the `source` command to refresh the environment variables for the current session using `source .bashrc` or `source .bash_profile`. Or you can close the current shell and reopen a new one.

```
local$ export ECHO=149.165.150.76
local$ export FS_USER=<put your futersystem account name here>
```

Now you can use the two variables that were set to login to the Echo server, using the following command

```
local$ ssh $FS_USER@$ECHO
```

**Note: If you have access to india but not the docker swarm system, your project may not have been authorized to access the docker swarm cluster. Send a ticket to FutureSystems ticket system to request this.**

Once logged in to the docker swarm head node, try to run:

```
echo$ docker run hello-world
```

to verify `docker run` works.

### 11.3.3.2 Creating a service and deploy to the swarm cluster

While `docker run` can start a container and you may even attach to its console, the recommended way to use a docker swarm cluster is to create a service and have it run on the swarm cluster. The service will be scheduled to one or many number of the nodes of the swarm cluster, based on the configuration. It is also easy to scale up the service when more swarm nodes are available. Docker swarm really makes it easier for service/application developers to focus on the functionality development but not worrying about how and where to bind the service to some resources/server. The deployment, access, and scaling up/down when necessary, are all managed transparently. Thus achieving the new paradigm of *serverless computing*.

As an example, the following command creates a service and deploy it to the swarm cluster, if the port is in use the port `9001` used in the command can be changed to an available port.

```
echo$ docker service create --name notebook_test -p 9001:8888 \
 jupyter/datascience-notebook start-notebook.sh
 --NotebookApp.password=NOTEBOOK_PASS_HASH
```

The `NOTEBOOK_PASS_HASH` can be generated in python:

```
>>> import IPython
>>> IPython.lib.passwd("YOUR_SELECTED_PASSWORD")
'sha1:52679c9:6762e266af44f86f3d170ca1.....'
```

So pass through the string starting with 'sha1:.....'.

The command pulls a published image from docker cloud, starts a container and runs a script to start the service inside the container with necessary parameters. The option “-p 9001:8888” maps the service port inside the container (8888) to

an external port of the cluster node (9001) so the service could be accessed from the Internet. In this example, you can then visit the URL:

```
local$ open http://$ECHO:9001
```

to access the Jupyter notebook. Using the specified password when you create the service to login.

Please note the service will be dynamically deployed to a container instance, which would be allocated to a swarm node based on the allocation policy. Docker makes this process transparent to the user and even created mesh routing so you can access the service using the IP address of the management head node of the swarm cluster, no matter which actual physical node the service was deployed to.

This also implies that the external port number used has to be free at the time when the service was created.

Some useful related commands:

```
echo$ docker service ls
```

lists the currently running services.

```
echo$ docker service ps notebook_test
```

lists the detailed info of the container where the service is running.

```
echo$ docker node ps NODE
```

lists all the running containers of a node.

```
echo$ docker node ls
```

lists all the nodes in the swarm cluster.

To stop the service and the container:

```
echo$ docker service rm notebook_test
```

### 11.3.3.3 Create your own service

You can create your own service and run it. To do so, start from a base image,

e.g., a ubuntu image from the docker cloud. Then you could:

- Run a container from the image and attach to its console to develop the service, and create a new image from the changed instance using command 'docker commit'.
- Create a dockerfile, which has the step by step building process of the service, and then build an image from it.

In reality, the first approach is probably useful when you are in the phase of develop and debug your application/service. Once you have the step by step instructions developed the latter approach is the recommended way.

Publish the image to the docker cloud by following this documentation:

- <https://docs.docker.com/docker-cloud/builds/push-images/>

Please make sure no sensitive information is included in the image to be published. Alternatively you could publish the image internally to the swarm cluster.

#### **11.3.3.4 Publish an image privately within the swarm cluster**

Once the image is published and available to the swarm cluster, you could start a new service from the image similar to the Jupyter Notebook example.

#### **11.3.3.5 Exercises**

E.Docker.FutureSystems.1:

*Obtain an account on future systems.*

E.Docker.FutureSystems.2:

*Create a REST service with swagger codegen and run it on the echo cloud (see example in [this section](#) )*

#### **11.3.4 Hadoop with Docker**

In this section we will explore the Map/Reduce framework using Hadoop provided through a Docker container.

We will showcase the functionality on a small example that calculates minimum, maximum, average and standard deviation values using several input files which contain float numbers.

This section is based on the hadoop release 3.1.1 which includes significant enhancements over the previous version of Hadoop 2.x. Changes include the use of the following software:

- CentOS 7
- systemctl
- Java SE Development Kit 8

A Dockerfile to create the hadoop deployment is available at

[\\*https://github.com/cloudmesh-community/book/blob/master/examples/docker/hadoop/3.1.1/Dockerfile](https://github.com/cloudmesh-community/book/blob/master/examples/docker/hadoop/3.1.1/Dockerfile)

### 11.3.4.1 Building Hadoop using Docker

You can build hadoop from the Dockerfile as follows:

```
$ mkdir cloudmesh-community
$ cd cloudmesh-community
$ git clone https://github.com/cloudmesh-community/book.git
$ cd book/examples/docker/hadoop/3.1.1
$ docker build -t cloudmesh/hadoop:3.1.1 .
```

The complete docker image for Hadoop consumes 1.5GB.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cloudmesh/hadoop	3.1.1	ba2c51f94348	1 hour ago	1.52GB

To use the image interactively you can start the container as follows:

```
$ docker run -it cloudmesh/hadoop:3.1.1 /etc/bootstrap.sh -bash
```

It may take a few minutes at first to download image.

### 11.3.4.2 Hadoop Configuration Files

The configuration files are included in the `conf` folder

### 11.3.4.3 Virtual Memory Limit

IN case you need more memory, you can increase it by changing the parameters in the file `mapred-site.xml`, for example:

- `mapreduce.map.memory.mba` to 4096
- `mapreduce.reduce.memory.mb` to 8192

### 11.3.4.4 hdfs Safemode leave command

A Safemode for HDFS is a read-only mode for the HDFS cluster, where it does not allow any modifications of files and blocks. Namenode disables safe mode automatically after starting up normally. If required, HDFS could be forced to leave the safe mode explicitly by this command:

```
$ hdfs dfsadmin -safemode leave
```

### 11.3.4.5 Examples

We included a statistics and a PageRank examples into the container. The examples are also available in github at

- <https://github.com/cloudmesh-community/book/tree/master/examples/docker/hadoop/3.1.1/examples>

We explain the examples next

#### 11.3.4.5.1 Statistical Example with Hadoop

After we launch the container and use the interactive shell, we can run the statistics Hadoop application which calculates the minimum, maximum, average, and standard deviation from values stored in a number of input files. Figure [Figure 133](#) shows the computing phases in a MapReduce job.

To achieve this, this Hadoop program reads multiple files from HDFS and provides calculated values. We walk through every step from compiling Java

source code to reading a output file from HDFS. The idea of this exercise is to get you started with Hadoop and the MapReduce concept. You may see the WordCount from Hadoop official website or documentation and this example has a same functions (Map/Reduce) except that you will be computing the basic statistics such as min, max, average, and standard deviation of a given data set.

The input to the program will be a text file(s) carrying exactly one floating point number per line. The result file includes *min*, *max*, *average*, and *standard deviation*.

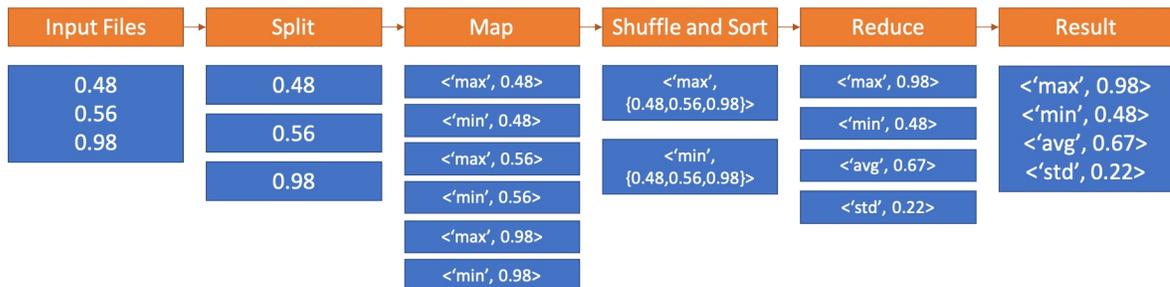


Figure 133: MapReduce example in Docker

#### 11.3.4.5.1.1 Base Location

The example is available within the container at:

```
container$ cd /cloudmesh/examples/statistics
```

#### 11.3.4.5.1.2 Input Files

A test input files are available under `/cloudmesh/examples/statistics/input_data` directory inside of the container. The statistics values for this input are *Min: 0.20 Max: 19.99 Avg: 9.51 StdDev: 5.55* for all input files.

10 files contain 55000 lines to process and each line is a random float point value ranging from 0.2 to 20.0.

#### 11.3.4.5.1.3 Compilation

The source code file name is `MinMaxAvgStd.java` which is available at

```
/cloudmesh/examples/statistics/src.
```

There are three functions in the code *Map*, *Reduce* and *Main* where *Map* reads each line of a file and updates values to calculate minimum, maximum values and *Reduce* collects mappers to produce average and standard deviation values at last.

```
$ export HADOOP_CLASSPATH=`$HADOOP_HOME/bin/hadoop classpath`
$ mkdir /cloudmesh/examples/statistics/dest
$ javac -classpath $HADOOP_CLASSPATH -d /cloudmesh/examples/statistics/dest /cloudmesh/examples/statistics/src/MinMaxAvgS
```

These commands simply prepare compiling the example code and the compiled class files are generated at the *dest* location.

#### 11.3.4.5.1.4 Archiving Class Files

Jar command tool helps archiving classes in a single file which will be used when Hadoop runs this example. This is useful because a jar file contains all necessary files to run a program.

```
$ cd /cloudmesh/examples/statistics
$ jar -cvf stats.jar -C ./dest/ .
```

#### 11.3.4.5.1.5 HDFS for Input/Output

The input files need to be uploaded to HDFS as Hadoop runs this example by reading input files from HDFS.

```
$ export PATH=$PATH:/HADOOP_HOME/bin
$ hadoop fs -mkdir stats_input
$ hadoop fs -put input_data/* stats_input
$ hadoop fs -ls stats_input/
```

If uploading is completed, you may see file listings like:

```
Found 10 items
-rw-r--r-- 1 root supergroup 13942 2018-02-28 23:16 stats_input/data_1000.txt
-rw-r--r-- 1 root supergroup 139225 2018-02-28 23:16 stats_input/data_10000.txt
-rw-r--r-- 1 root supergroup 27868 2018-02-28 23:16 stats_input/data_2000.txt
-rw-r--r-- 1 root supergroup 41793 2018-02-28 23:16 stats_input/data_3000.txt
-rw-r--r-- 1 root supergroup 55699 2018-02-28 23:16 stats_input/data_4000.txt
-rw-r--r-- 1 root supergroup 69663 2018-02-28 23:16 stats_input/data_5000.txt
-rw-r--r-- 1 root supergroup 83614 2018-02-28 23:16 stats_input/data_6000.txt
-rw-r--r-- 1 root supergroup 97490 2018-02-28 23:16 stats_input/data_7000.txt
-rw-r--r-- 1 root supergroup 111451 2018-02-28 23:16 stats_input/data_8000.txt
-rw-r--r-- 1 root supergroup 125337 2018-02-28 23:16 stats_input/data_9000.txt
```

#### 11.3.4.5.1.6 Run Program with a Single Input File

We are ready to run the program to calculate values from text files. First, we simply run the program with a single input file to see how it works. `data_1000.txt`

contains 1000 lines of floats, we use this file here.

```
$ hadoop jar stats.jar exercise.MinMaxAvgStd stats_input/data_1000.txt stats_output_1000
```

The command runs with input parameters which indicate a jar file (the program, stats.jar), exercise.MinMaxAvgStd (package name.class name), input file path (stats\_input/data\_1000.txt) and output file path (stats\_output\_1000).

The sample results that the program produces look like this:

```
18/02/28 23:48:50 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/02/28 23:48:50 INFO input.FileInputFormat: Total input paths to process: 1
18/02/28 23:48:50 INFO mapreduce.JobSubmitter: number of splits:1
18/02/28 23:48:50 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1519877569596_0002
18/02/28 23:48:51 INFO impl.YarnClientImpl: Submitted application application_1519877569596_0002
18/02/28 23:48:51 INFO mapreduce.Job: The url to track the job: http://f5e82d68ba4a:8088/proxy/application_1519877569596_
18/02/28 23:48:51 INFO mapreduce.Job: Running job: job_1519877569596_0002
18/02/28 23:48:56 INFO mapreduce.Job: Job job_1519877569596_0002 running in uber mode: false
18/02/28 23:48:56 INFO mapreduce.Job: map 0% reduce 0%
18/02/28 23:49:00 INFO mapreduce.Job: map 100% reduce 0%
18/02/28 23:49:05 INFO mapreduce.Job: map 100% reduce 100%
18/02/28 23:49:05 INFO mapreduce.Job: Job job_1519877569596_0002 completed successfully
18/02/28 23:49:05 INFO mapreduce.Job: Counters: 49
 File System Counters
 FILE: Number of bytes read=81789
 FILE: Number of bytes written=394101
 FILE: Number of read operations=0
 FILE: Number of large read operations=0
 FILE: Number of write operations=0
 HDFS: Number of bytes read=14067
 HDFS: Number of bytes written=86
 HDFS: Number of read operations=6
 HDFS: Number of large read operations=0
 HDFS: Number of write operations=2
 Job Counters
 Launched map tasks=1
 Launched reduce tasks=1
 Data-local map tasks=1
 Total time spent by all maps in occupied slots (ms)=2107
 Total time spent by all reduces in occupied slots (ms)=2316
 Total time spent by all map tasks (ms)=2107
 Total time spent by all reduce tasks (ms)=2316
 Total vcore-seconds taken by all map tasks=2107
 Total vcore-seconds taken by all reduce tasks=2316
 Total megabyte-seconds taken by all map tasks=2157568
 Total megabyte-seconds taken by all reduce tasks=2371584
 Map-Reduce Framework
 Map input records=1000
 Map output records=3000
 Map output bytes=75783
 Map output materialized bytes=81789
 Input split bytes=125
 Combine input records=0
 Combine output records=0
 Reduce input groups=3
 Reduce shuffle bytes=81789
 Reduce input records=3000
 Reduce output records=4
 Spilled Records=6000
 Shuffled Maps =1
 Failed Shuffles=0
 Merged Map outputs=1
 GC time elapsed (ms)=31
 CPU time spent (ms)=1440
 Physical memory (bytes) snapshot=434913280
 Virtual memory (bytes) snapshot=1497260032
 Total committed heap usage (bytes)=402653184
 Shuffle Errors
 BAD_ID=0
 CONNECTION=0
 IO_ERROR=0
 WRONG_LENGTH=0
```

```
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
 Bytes Read=13942
File Output Format Counters
 Bytes Written=86
```

The second line of the following logs indicates that the number of input files is 1.

#### 11.3.4.5.1.7 Result for Single Input File

We reads results from HDFS by:

```
$ hadoop fs -cat stats_output_1000/part-r-00000
```

The sample output looks like:

```
Max: 19.9678704297
Min: 0.218880718983
Avg: 10.225467263249385
Std: 5.679809322880863
```

#### 11.3.4.5.1.8 Run Program with Multiple Input Files

The first run was done pretty quickly (1440 milliseconds took according to the previous sample result) because the input file size was small (1,000 lines) and it was a single file. We provides more input files with a larger size (2,000 to 10,000 lines). Input files are already uploaded to HDFS. We simply run the program again with a slight change in the parameters.

```
$ hadoop jar stats.jar exercise.MinMaxAvgStd stats_input/ stats_output_all
```

The command is almost same except that an input path is a directory and a new output directory. Note that every time that you run this program, the output directory will be created which means that you have to provide a new directory name unless you delete it.

The sample output messages look like the following which is almost identical compared to the previous run except that this time the number of input files to process is 10, see the line two next:

```
18/02/28 23:17:18 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/02/28 23:17:18 INFO input.FileInputFormat: Total input paths to process: 10
18/02/28 23:17:18 INFO mapreduce.JobSubmitter: number of splits:10
18/02/28 23:17:18 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1519877569596_0001
18/02/28 23:17:19 INFO impl.YarnClientImpl: Submitted application application_1519877569596_0001
18/02/28 23:17:19 INFO mapreduce.Job: The url to track the job: http://f5e82d68ba4a:8088/proxy/application_1519877569596_
```

```

18/02/28 23:17:19 INFO mapreduce.Job: Running job: job_1519877569596_0001
18/02/28 23:17:24 INFO mapreduce.Job: Job job_1519877569596_0001 running in uber mode: false
18/02/28 23:17:24 INFO mapreduce.Job: map 0% reduce 0%
18/02/28 23:17:32 INFO mapreduce.Job: map 40% reduce 0%
18/02/28 23:17:33 INFO mapreduce.Job: map 60% reduce 0%
18/02/28 23:17:36 INFO mapreduce.Job: map 70% reduce 0%
18/02/28 23:17:37 INFO mapreduce.Job: map 100% reduce 0%
18/02/28 23:17:39 INFO mapreduce.Job: map 100% reduce 100%
18/02/28 23:17:39 INFO mapreduce.Job: Job job_1519877569596_0001 completed successfully
18/02/28 23:17:39 INFO mapreduce.Job: Counters: 49
 File System Counters
 FILE: Number of bytes read=4496318
 FILE: Number of bytes written=10260627
 FILE: Number of read operations=0
 FILE: Number of large read operations=0
 FILE: Number of write operations=0
 HDFS: Number of bytes read=767333
 HDFS: Number of bytes written=84
 HDFS: Number of read operations=33
 HDFS: Number of large read operations=0
 HDFS: Number of write operations=2
 Job Counters
 Launched map tasks=10
 Launched reduce tasks=1
 Data-local map tasks=10
 Total time spent by all maps in occupied slots (ms)=50866
 Total time spent by all reduces in occupied slots (ms)=4490
 Total time spent by all map tasks (ms)=50866
 Total time spent by all reduce tasks (ms)=4490
 Total vcore-seconds taken by all map tasks=50866
 Total vcore-seconds taken by all reduce tasks=4490
 Total megabyte-seconds taken by all map tasks=52086784
 Total megabyte-seconds taken by all reduce tasks=4597760
 Map-Reduce Framework
 Map input records=55000
 Map output records=165000
 Map output bytes=4166312
 Map output materialized bytes=4496372
 Input split bytes=1251
 Combine input records=0
 Combine output records=0
 Reduce input groups=3
 Reduce shuffle bytes=4496372
 Reduce input records=165000
 Reduce output records=4
 Spilled Records=330000
 Shuffled Maps =10
 Failed Shuffles=0
 Merged Map outputs=10
 GC time elapsed (ms)=555
 CPU time spent (ms)=16040
 Physical memory (bytes) snapshot=2837708800
 Virtual memory (bytes) snapshot=8200089600
 Total committed heap usage (bytes)=2213019648
 Shuffle Errors
 BAD_ID=0
 CONNECTION=0
 IO_ERROR=0
 WRONG_LENGTH=0
 WRONG_MAP=0
 WRONG_REDUCE=0
 File Input Format Counters
 Bytes Read=766082
 File Output Format Counters
 Bytes Written=84

```

### 11.3.4.5.1.9 Result for Multiple Files

```
$ hadoop fs -cat stats_output_all/part-r-00000
```

The expected result looks like:

```

Max: 19.999191254
Min: 0.200268613863

```

```
Avg: 9.514884854468903
Std: 5.553921579413547
```

### 11.3.4.5.2 Conclusion

The example program of calculating some values by reading multiple files shows how Map/Reduce is written by a Java programming language and how Hadoop runs its program using HDFS. We also observed the one of benefits using Docker container which is that the hassle of configuration and installation of Hadoop is not necessary anymore.

### 11.3.4.6 References

- The details of the new version is available from the official site at <http://hadoop.apache.org/docs/r3.1.1/index.html>

## 11.3.5 Docker Pagerank

PageRank is a popular example algorithm used to display the ability of big data applications to run parallel tasks. This example will show how the docker hadoop image can be used to execute the Pagerank example which is available in `/cloudmesh/examples/pagerank`

### 11.3.5.1 Use the automated script

We make the steps of compiling java source, archiving class files, load input files and run the program into one single script. To execute it with the input file: `PageRankDataGenerator/pagerank5000g50.input.0`, using 5000 urls and 1 iteration:

```
$ cd /cloudmesh/examples/pagerank
$./compileAndExecHadoopPageRank.sh PageRankDataGenerator/pagerank5000g50.input.0 5000 1
```

Result will look like

```
output.pagerank/part-r-00000
```

The head of the result will look like

```
head output.pagerank/part-r-00000
```

```
0 2.9999999999999997E-5
1 2.9999999999999997E-5
```

```
2 2.9999999999999997E-5
3 2.9999999999999997E-5
4 2.9999999999999997E-5
5 2.9999999999999997E-5
6 2.9999999999999997E-5
7 2.9999999999999997E-5
8 2.9999999999999997E-5
9 2.9999999999999997E-5
```

### 11.3.5.2 Compile and run by hand

If one wants to generate the java class files and archive them as the previous exercise, one could use the following code (which is actually inside compileAndExecHadoopPageRank.sh)

```
export HADOOP_CLASSPATH=`$HADOOP_PREFIX/bin/hadoop classpath`
mkdir /cloudmesh/examples/pagerank/dist
$ find /cloudmesh/examples/pagerank/src/indiana/cgl/hadoop/pagerank/ \
 -name "*.java"|xargs javac -classpath $HADOOP_CLASSPATH \
 -d /cloudmesh/examples/pagerank/dist
$ cd /cloudmesh/examples/pagerank/dist
$ jar -cvf HadoopPageRankMooc.jar -C . .
```

### Load input files to HDFS

```
$ export PATH=$PATH:$HADOOP_PREFIX/bin
$ cd /cloudmesh/examples/pagerank/
$ hadoop fs -mkdir input.pagerank
$ hadoop fs -put PageRankDataGenerator/pagerank5000g50.input.0 input.pagerank
```

- Run program with the [PageRank Inputs File Directory][PageRank Output Directory][Number of Urls][Number Of Iterations]

```
$ hadoop jar dist/HadoopPageRankMooc.jar indiana.cgl.hadoop.pagerank.HadoopPageRank input.pagerank output.pagerank 5000
```

### Result

```
$ hadoop fs -cat output.pagerank/part-r-00000
```

## 11.3.6 Apache Spark with Docker

### 11.3.6.1 Pull Image from Docker Repository

We use a Docker image from Docker Hub: (<https://hub.docker.com/r/sequenceiq/spark/>) This repository contains a Docker file to build a Docker image with Apache Spark and Hadoop Yarn.

```
$ docker pull sequenceiq/spark:1.6.0
```

## 11.3.6.2 Running the Image

In this step, we will launch a Spark container.

### 11.3.6.2.1 Running interactively

```
$ docker run -it -p 8088:8088 -p 8042:8042 -h sandbox sequenceiq/spark:1.6.0 bash
```

### 11.3.6.2.2 Running in the background

```
$ docker run -d -h sandbox sequenceiq/spark:1.6.0 -d
```

## 11.3.6.3 Run Spark

After a container is launched, we can run Spark in the following two modes: (1) yarn-client and (2) yarn-cluster. The differences between the two modes can be found here: <https://spark.apache.org/docs/latest/running-on-yarn.html>

### 11.3.6.3.1 Run Spark in Yarn-Client Mode

```
$ spark-shell --master yarn-client --driver-memory 1g --executor-memory 1g --executor-cores 1
```

### 11.3.6.3.2 Run Spark in Yarn-Cluster Mode

```
$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client --driver-memory 1g --executor-memory 1g --e
```

## 11.3.6.4 Observe Task Execution from Running Logs of SparkPi

Let us observe Spark task execution by adjusting the parameter of SparkPi and the Pi result from the following two commands.

```
$ spark-submit --class org.apache.spark.examples.SparkPi \
 --master yarn-client --driver-memory 1g \
 --executor-memory 1g \
 --executor-cores 1 $SPARK_HOME/lib/spark-examples-1.6.0-hadoop2.6.0.jar 10
$ spark-submit --class org.apache.spark.examples.SparkPi \
 --master yarn-client --driver-memory 1g \
 --executor-memory 1g \
 --executor-cores 1 $SPARK_HOME/lib/spark-examples-1.6.0-hadoop2.6.0.jar 10000
```

## 11.3.6.5 Write a Word-Count Application with Spark RDD

Let us write our own word-count with Spark RDD. After the shell has been

started, copy and paste the following code in console line by line.

#### 11.3.6.5.1 Launch Spark Interactive Shell

```
$ spark-shell --master yarn-client --driver-memory 1g --executor-memory 1g --executor-cores 1
```

#### 11.3.6.5.2 Program in Scala

```
val textFile = sc.textFile("file:///etc/hosts")
val words = textFile.flatMap(line => line.split("\\s+"))
val counts = words.map(word => (word, 1)).reduceByKey(_ + _)
counts.values.sum()
```

#### 11.3.6.5.3 Launch PySpark Interactive Shell

```
$ pyspark --master yarn-client --driver-memory 1g --executor-memory 1g --executor-cores 1
```

#### 11.3.6.5.4 Program in Python

```
textFile = sc.textFile("file:///etc/hosts")
words = textFile.flatMap(lambda line:line.split())
counts = words.map(lambda word:(word, 1)).reduceByKey(lambda x,y: x+y)
counts.map(lambda x:x[1]).sum()
```

### 11.3.6.6 Docker Spark Examples

#### 11.3.6.6.1 K-Means Example

First we need to pull the image from the Docker Hub :

```
$ docker pull sequenceiq/spark-native-yarn
```

It will take sometime to download the image. Now we have to run docker spark image interactively.

```
$ docker run -i -t -h sandbox sequenceiq/spark-native-yarn /etc/bootstrap.sh -bash
```

This will take you to the interactive mode.

Let us run a sample KMeans example. This is already built with Spark.

Here we specify the data data set from a local folder inside the image and we run the sample class KMeans in the sample package. The sample data set used is inside the sample-data folder. Spark has it's own format for machine learning datasets. Here the kmeans\_data.txt file contains the KMeans dataset.

```
$./bin/spark-submit --class sample.KMeans \
 --master execution-context:org.apache.spark.tez.TezJobExecutionContext \
 \
 --conf update-classpath=true \
 ./lib/spark-native-yarn-samples-1.0.jar /sample-data/kmeans_data.txt
```

If you run this successfully, you can get an output as shown here.

```
Finished iteration (delta = 0.0)
Final centers:
DenseVector(0.15000000000000002, 0.15000000000000002, 0.15000000000000002)
DenseVector(9.2, 9.2, 9.2)
DenseVector(0.0, 0.0, 0.0)
DenseVector(9.05, 9.05, 9.05)
```

### 11.3.6.2 Join Example

Run the following command to do a sample join operation on a given dataset. Here we use two datasets, namely join1.txt and join2.txt. Then we perform the join operation that we discussed in the theory section.

```
$./bin/spark-submit --class sample.Join --master execution-context:org.apache.spark.tez.TezJobExecutionContext --conf up
```

### 11.3.6.3 Word Count

In this example the wordcount.txt will be used to do the word count using multiple reducers. Number 1 at the end of the command determines the number of reducers. As spark can run multiple reducers, we can specify the number as a parameter to the programme.

```
$./bin/spark-submit --class sample.WordCount --master execution-context:org.apache.spark.tez.TezJobExecutionContext --cc
```

### 11.3.6.7 Interactive Examples

Here we need a new image to work on. Let us run the following command. This will pull the necessary repositories from docker hub, as we do not have most of the dependencies related to it. This can take a few minutes to download everything.

```
$ docker run -it-p 8888:8888 -v $PWD:/cloudmesh/spark --name spark jupyter/pyspark-notebook
```

Here you will get the following output in the terminal.

```
docker run -it -p 8888:8888 -v $PWD:/cloudmesh/spark --name spark jupyter/pyspark-notebook
Unable to find image 'jupyter/pyspark-notebook:latest' locally
latest: Pulling from jupyter/pyspark-notebook
a48c500ed24e: Pull complete
```

```
1e1de00ff7e1: Pull complete
0330ca45a200: Pull complete
471db38bcfbf: Pull complete
0b4aba487617: Pull complete
d44ea0cd796c: Pull complete
5ac827d588be: Pull complete
d8d7747a335e: Pull complete
08790511e3e9: Pull complete
e3c68aea9a5f: Pull complete
484c6d5fc38a: Pull complete
0448c1360cb9: Pull complete
61d7e6dc705d: Pull complete
92f1091ed72b: Pull complete
8045d3663a7e: Pull complete
1bde7ba25439: Pull complete
5618f8ed38b4: Pull complete
f08523cb6144: Pull complete
99eee56fda2f: Pull complete
b37b1ce39785: Pull complete
aee4b9eac4ea: Pull complete
f810ef87439d: Pull complete
038786dce388: Pull complete
ded31312ea33: Pull complete
30221ffdd1a6: Pull complete
da1d368f8592: Pull complete
523809a30a21: Pull complete
47ab1b230dd2: Pull complete
442f9435e1a9: Pull complete
Digest: sha256:f8b6309cd39481de1a169143189ed0879b12b56fe286d254d03fa34ccad90734
Status: Downloaded newer image for jupyter/pyspark-notebook:latest
Container must be run with group "root" to update passwd file
Executing the command: jupyter notebook
[I 15:47:52.900 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 15:47:53.167 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 15:47:53.167 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 15:47:53.176 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 15:47:53.177 NotebookApp] The Jupyter Notebook is running at:
[I 15:47:53.177 NotebookApp] http://(3a3d9f7e2565 or 127.0.0.1):8888/?token=f22492fe7ab8206ac2223359e0603a0dff54d98096ab7930
[I 15:47:53.177 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:47:53.177 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://(3a3d9f7e2565 or 127.0.0.1):8888/?token=f22492fe7ab8206ac2223359e0603a0dff54d98096ab7930
```

Please copy the url shown at the end of the terminal output and go to that url in the browser.

You will see the following output in the browser, (Use Google Chrome)



## Jupyter Notebook in Browser

First navigate to the work folder. Let us create a new python file here. Click python3 in the new menu.



## Create a new python file

Now add the following content in the new file. In Jupyter notebook, you can enter a python command or python code and press

```
SHIFT + ENTER
```

This will run the code interactively.

Now let's create the following content.

```
import os
os.getcwd()

import pyspark
sc = pyspark.SparkContext('local[*]')
rdd = sc.parallelize(range(1000))
rdd.takeSample(False, 5)
```

Now let us do the following.

In the following stage we configure spark context and import the necessary files.

```
os.makedirs("data")

from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.linalg import SparseVector
sc.version
```

Next stage we use sample data set by creating them in form of an array and we train the kmeans algorithm.

```
sparse_data = [
 SparseVector(3, {1:1.0}),
 SparseVector(3, {1:1.1}),
 SparseVector(3, {2:1.0}),
 SparseVector(3, {2:1.1})
]

model = KMeans.train(sc.parallelize(sparse_data), 2, initializationMode='k-means||',
 seed=50, initializationSteps=5, epsilon=1e-4)

model.predict(array([0.,1.,0.]))
model.predict(array([0.,0.,1.]))
```

```
model.predict(sparse_data[0])
model.predict(sparse_data[2])
```

In the final stage we put sample values and check the predictions on the cluster. In addition to that feed the data using SparseVector format and we add the kmeans initialization mode, the error margin and the palatalization. We put the step size as 5 for this example. In the previous one we did not specify any parameters.

The predict term predicts the cluster id which it belongs to.

```
data = array([0.0, 0.0, 1.0, 1.0, 9.0, 8.0, 8.0, 9.0]).reshape(4, 2)
model = KMeans.train(sc.parallelize(data), 2, initializationMode='random',
 seed=50, initializationSteps=5, epsilon=1e-4)
model.predict(array([0.0, 0.0])) == model.predict(array([1.0, 1.0]))
model.predict(array([8.0, 9.0]))
model.predict(array([8.0, 9.0])) == model.predict(array([9.0, 8.0]))
model.k
model.computeCost(sc.parallelize(data))
```

Then in the following way you can check whether two data points belong to one cluster or not.

```
isinstance(model.clusterCenters, list)
```

#### 11.3.6.7.1 Stop Docker Container

```
$ docker stop spark
```

#### 11.3.6.7.2 Start Docker Container Again

```
$ docker start spark
```

#### 11.3.6.7.3 Remove Docker Container

```
$ docker rm spark
```

## 11.4 KUBERNETES

---

### 11.4.1 Introduction to Kubernetes

---



Learning Objectives

- What is Kubernetes?
  - What are containers?
  - Cluster components in Kubernetes
  - Basic Units in Kubernetes
  - Run an example with Minikube
  - Interactive online tutorial
  - Have a solid understanding of Containers and Kubernetes
  - Understand the Cluster components of Kubernetes
  - Understand the terminology of Kubernetes
  - Gain practical experience with kubernetes
  - With minikube
  - With an interactive online tutorial
- 

Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers.

- <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

With Kubernetes, you can:

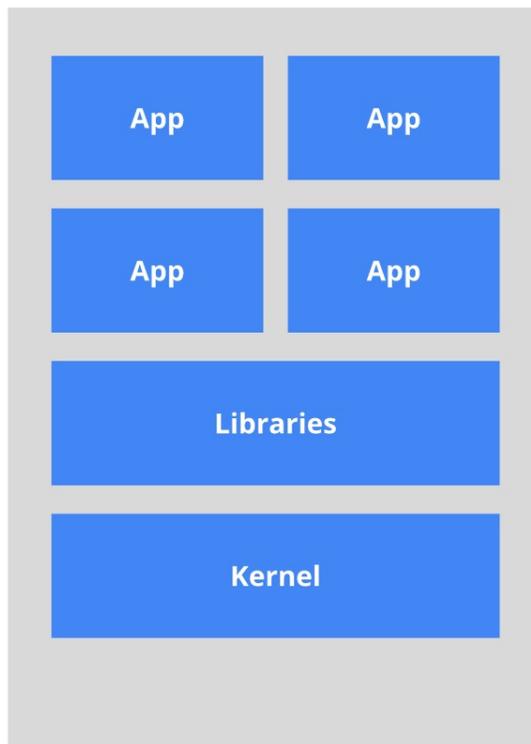
- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Roll out new features seamlessly.
- Limit hardware usage to required resources only.
- Run applications in public and private clouds.

Kubernetes is

- Portable: public, private, hybrid, multi-cloud
- Extensible: modular, pluggable, hookable, composable
- Self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

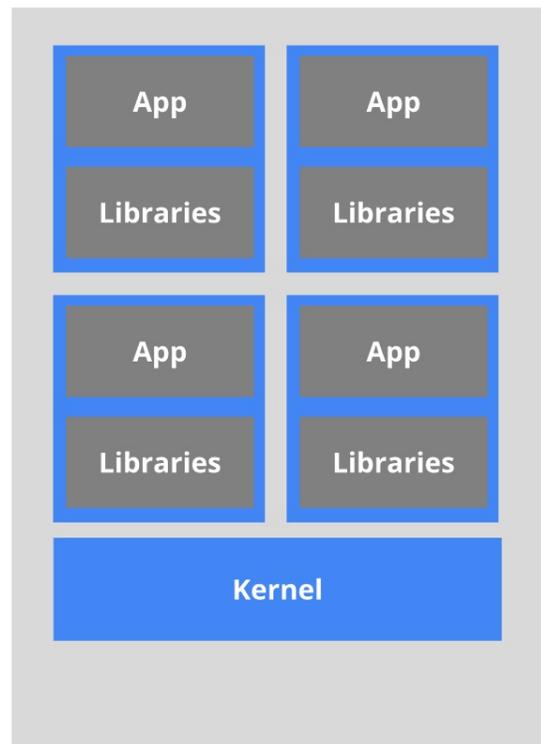
#### **11.4.1.1 What are containers?**

**The old way:** Applications on host



*Heavyweight, non-portable  
Relies on OS package manager*

**The new way:** Deploy containers



*Small and fast, portable  
Uses OS-level virtualization*

Figure 134: Kubernetes Containers [\[Image Source\]](#)

[Figure 134](#) shows a depiction of the container architecture.

### 11.4.1.2 Terminology

In kubernetes we are using the following terminology

Pods:

A pod (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers. A pod's contents are always co-located and co-scheduled, and run in a shared context. A pod models an application-specific *logical host*. It contains one or more application containers which are relatively tightly coupled. In a pre-container world, they would have executed on the same physical or virtual machine.

Services:

Service is an abstraction which defines a logical set of Pods and a policy by which to access them. Sometimes they are called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector.

Deployments:

A Deployment controller provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

### **11.4.1.3 Kubernetes Architecture**

The architecture of kubernetes is shown in [Figure 135](#).

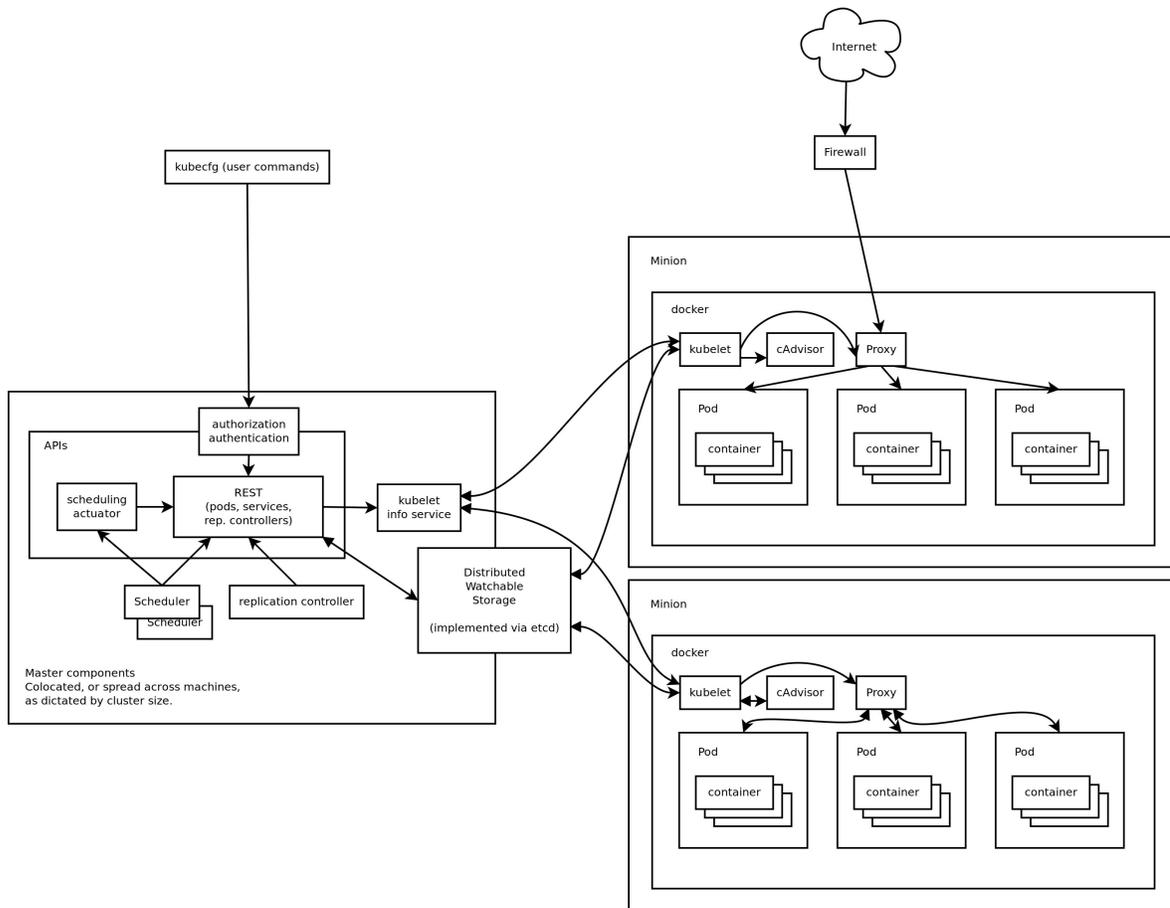


Figure 135: Kubernetes (Source: Google)

### 11.4.1.4 Minikube

To try out kubernetes on your own computer you can download and install minikube. It deploys and runs a single-node Kubernetes cluster inside a VM. Hence it provide a reasonable environment not only to try it out, but also for development [\[cite\]](#).

In this section we will first discuss how to install minikube and then showcase an example.

#### 11.4.1.4.1 Install minikube

##### 11.4.1.4.1.0.1 OSX

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.25.0/minikube-darwin-amd64 && chmod +x minikube &
```

#### 11.4.1.4.1.0.2 Windows 10

We assume that you have installed Oracle VirtualBox in your machine which must be a version 5.x.x.

Initially, we need to download two executables.

[Download Kubectl](#)

[Download Minikube](#)

After downloading these two executables place them in the cloudmesh directory we earlier created. Rename the `minikube-windows-amd64.exe` to `minikube.exe`. Make sure `minikube.exe` and `kubectl.exe` lie in the same directory.

#### 11.4.1.4.1.0.3 Linux

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.25.0/minikube-linux-amd64 && chmod +x minikube &&
```

Installing KVM2 is important for Ubuntu distributions

```
$ sudo apt install libvirt-bin qemu-kvm
$ sudo usermod -a -G libvirt $(whoami)
$ newgrp libvirt
```

We are going to run minikube using KVM2 libraries instead of virtualbox libraries for windows installation.

Then install the drivers for KVM2,

```
$ curl -LO https://storage.googleapis.com/minikube/releases/latest/docker-machine-driver-kvm2 && chmod +x docker-machine-
```

#### 11.4.1.4.2 Start a cluster using Minikube

##### 11.4.1.4.2.0.1 OSX Minikube Start

```
$ minikube start
```

##### 11.4.1.4.2.0.2 Ubuntu Minikube Start

```
$ minikube start --vm-driver=kvm2
```

#### 11.4.1.4.2.0.3 Windows 10 Minikube Start

In this case you must run Windows PowerShell as administrator. For this search for the application in search and right click and click Run as administrator. If you are an administrator it will run automatically but if you are not please make sure you provide the admin login information in the pop up.

```
$ cd C:\Users\\Documents\cloudmesh
$.\minikube.exe start --vm-driver="virtualbox"
```

#### 11.4.1.4.3 Create a deployment

```
$ kubectl run hello-minikube --image=k8s.gcr.io/echoserver:1.4 --port=8080
```

#### 11.4.1.4.4 Expose the servi

```
$ kubectl expose deployment hello-minikube --type=NodePort
```

#### 11.4.1.4.5 Check running status

This step is to make sure you have a pod up and running.

```
$ kubectl get pod
```

#### 11.4.1.4.6 Call service api

```
$ curl $(minikube service hello-minikube --url)
```

#### 11.4.1.4.7 Take a look from Dashboard

```
$ minikube dashboard
```

If you want to get an interactive dashboard,

```
$ minikube dashboard --url=true
http://192.168.99.101:30000
```

Browse to <http://192.168.99.101:30000> in your web browser and it will provide a GUI dashboard regarding minikube.

#### 11.4.1.4.8 Delete the service and deployment

```
$ kubectl delete service hello-minikube
$ kubectl delete deployment hello-minikube
```

#### 11.4.1.4.9 Stop the cluster

For all platforms we can use the following command.

```
$ minikube stop
```

#### 11.4.1.5 Interactive Tutorial Online

- Start cluster <https://kubernetes.io/docs/tutorials/kubernetes-basics/cluster-interactive/>
- Deploy app [https://kubernetes.io/docs/tutorials/kubernetes-basics/cluster-interactive](https://kubernetes.io/docs/tutorials/kubernetes-basics/cluster-interactive/)
- Explore <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore-intro/>
- Expose <https://kubernetes.io/docs/tutorials/kubernetes-basics/expose-intro/>
- Scale <https://kubernetes.io/docs/tutorials/kubernetes-basics/scale-intro/>
- Update <https://kubernetes.io/docs/tutorials/kubernetes-basics/update-interactive/>
- MiniKube <https://kubernetes.io/docs/tutorials/stateless-application/hello-minikube/>

### 11.4.2 Using Kubernetes on FutureSystems

This section introduces you on how to use the Kubernetes cluster on FutureSystems. Currently we have deployed kubernetes on our cluster called *echo*.

#### 11.4.2.1 Getting Access

You will need an account on FutureSystems and upload the ssh key to the FutureSystems portal from the computer from which you want to login to echo. To verify, if you have access try to see if you can log into victor.futuresystems.org. You need to be a member of a valid FutureSystems project.

For Fall 2018 classes at IU you need to be in the following project:

<https://portal.futuresystems.org/project/553>

If you have verified that you have access to the victor, you can now try to login to the kubernetes cluster head node with the same username and key. Run these first on your **local machine** to set the username and login host:

```
$ export ECHOK8S=149.165.150.85
$ export FS_USER=<put your futersystem account name here>
```

Then you can login to the kubernetes head node by running:

```
$ ssh $FS_USER@$ECHOK8S
```

**NOTE: If you have access to victor but not the kubernetes system, your project may not have been authorized to access the kubernetes cluster. Send a ticket to FutureSystems ticket system to request this.**

Once you are logged in to the kubernetes cluster head node you can run commands on the **remote echo kubernetes machine** (all commands shown in next except stated otherwise) to use the kubernetes installation there. First try to run:

```
$ kubectl get pods
```

This will let you know if you have access to kubernetes and verifies if the kubectl command works for you. Naturally it will also list the pods.

### 11.4.2.2 Example Use

The following command runs an image called Nginx with two replicas, Nginx is a popular web sever which is well known as a high performance load balancer.

```
$ kubectl run nginx --replicas=2 --image=nginx --port=80
```

As a result of this one deployment was created, and two PODs are created and started. If you encounter and error stating that the deployment already exists when executing the previous command that is because the command has already been executed. To see the deployment, please use the command, this command should work even if you noticed the error mentioned.

```
$ kubectl get deployment
```

This will result in the following output

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
------	---------	---------	------------	-----------	-----

```
nginx 2 2 2 2 7m
```

To see the pods please use the command

```
$ kubectl get pods
```

This will result in the following output

NAME	READY	STATUS	RESTARTS	AGE
nginx-7587c6fdb6-4jnh6	1/1	Running	0	7m
nginx-7587c6fdb6-pxpsz	1/1	Running	0	7m

If we want to see more detailed information we can use the command

```
$ kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-75...-4jnh6	1/1	Running	0	8m	192.168.56.2	e003
nginx-75...-pxpsz	1/1	Running	0	8m	192.168.255.66	e005

Please note the IP address field. Make sure you are using the IP address that is listed when you execute the command since the IP address may have changed. Now if we try to access the nginx homepage with wget (or curl)

```
$ wget 192.168.56.2
```

we see the following output:

```
--2018-02-20 14:05:59-- http://192.168.56.2/
Connecting to 192.168.56.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 612 [text/html]
Saving to: 'index.html'

index.html 100%[=====>] 612 --.-KB/s in 0s

2018-02-20 14:05:59 (38.9 MB/s) - 'index.html' saved [612/612]
```

It verifies that the specified image was running, and it is accessible from within the cluster.

Next we need to start thinking about how we access this web server from outside the cluster. We can explicitly expose the service with the following command. You can change the name that is set using `--name` to what you want. Given that it adheres to the naming standards. If the name you enter is already in the system your command will return an error saying the service already exists.

```
$ kubectl expose deployment nginx --type=NodePort --name=abc-nginx-ext
```

We will see the response

```
$ service "nginx-external" exposed
```

To find the exposed ip addresses, we simply issue the command

```
$ kubectl get svc
```

We see something like this

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	8h
abc-nginx-ext	NodePort	10.110.177.35	<none>	80:31386/TCP	3s

please note that we have given a unique name.

---

For IU students:

You could use your username or if you use one of our classes your hid. The number part will typically be sufficient. For class users that do not use the hid in the name we will terminate all instances without notification. In addition, we like you explicitly to add “-ext” to every container that is exposed to the internet. Naturally we want you to shut down such services if they are not in use. Failure to do so may result in termination of the service without notice, and in the worst case revocation of your privileges to use *echo*.

---

In our example you will find the port on which our service is exposed and remapped to. We find the port **31386** in the value **80:31386/TCP** in the ports column for the running container.

Now if we visit this URL, which is the public IP of the head node followed by the exposed port number, from a browser on **your local machine**

```
http://149.165.150.85:31386
```

you should see the ‘Welcome to nginx’ page.

Once you have done all the work needed using the service you can delete it using the following command.

```
$ kubectl delete service <service-name>
```

### 11.4.2.3 Exercises

E.Kubernetes.fs.1:

*Explore more complex service examples.*

E.Kubernetes.fs.2:

*Explore constructing a complex web app with multiple services.*

E.Kubernetes.fs.3:

*Define a deployment with a yaml file declaratively.*

## 11.5 RUNNING SINGULARITY CONTAINERS ON COMET

---

This section was copied from

- [https://www.sdsc.edu/support/user\\_guides/tutorials/singularity.html](https://www.sdsc.edu/support/user_guides/tutorials/singularity.html)

and modified. To use it you will need an account on comet which can be obtained via XSEDE. In case you use this material as part of a class please contact your teacher for more information.

### 11.5.1 Background

What is Singularity?

*“Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. This means that you don’t have to ask your cluster admin to install anything for you - you can put it in a Singularity container and run.”*

[from the Singularity web site at <http://singularity.lbl.gov/>]

There are numerous good tutorials on how to install and run Singularity on Linux, OS X, or Windows so we won’t go into much detail on that process here. In this tutorial you will learn how to run Singularity on Comet. First we will review how to access a compute node on Comet and provide a simple example

to help get you started. There are numerous tutorial on how to get started with Singularity, but there are some details specific to running Singularity on Comet which are not covered in those tutorials. This tutorial assumes you already have an account on Comet. You will also need access to a basic set of example files to get started. SDSC hosts a Github repository containing a 'Hello world!' example which you may clone with the following command:

```
git clone https://github.com/hpcdevops/singularity-hello-world.git
```

## 11.5.2 Tutorial Contents

- Why Singularity?
- Downloading & Installing Singularity
- Building Singularity Containers
- Running Singularity Containers on Comet
- Running Tensorflow on Comet Using Singularity

## 11.5.3 Why Singularity?

Listed next is a typical list of commands you would need to issue in order to implement a functional Python installation for scientific research:

```
COMMAND=apt-get -y install libx11-dev
COMMAND=apt-get install build-essential python-libdev
COMMAND=apt-get install build-essentyial openmpi-dev
COMMAND=apt-get install cmake
COMMAND=apt-get install g++
COMMAND=apt-get install git-lfs
COMMAND=apt-get install libXss.so.1
COMMAND=apt-get install libgdal1-dev libproj-dev
COMMAND=apt-get install libjsoncpp-dev libjsoncpp0
COMMAND=apt-get install libmpich-dev --user
COMMAND=apt-get install libpthread-stubs0 libpthread-stubs0-dev libx11-dev libx11-d
COMMAND=apt-get install libudev0:i386
COMMAND=apt-get install numpy
COMMAND=apt-get install python-matplotlib
COMMAND=apt-get install python3`
```

Singularity allows you to avoid this time-consuming series of steps by packaging these commands in a re-usable and editable script, allowing you to quickly, easily, and repeatedly implement a custom container designed specifically for your analytical needs.

[Figure 136](#) compares a VM vs. Docker vs. Singularity.

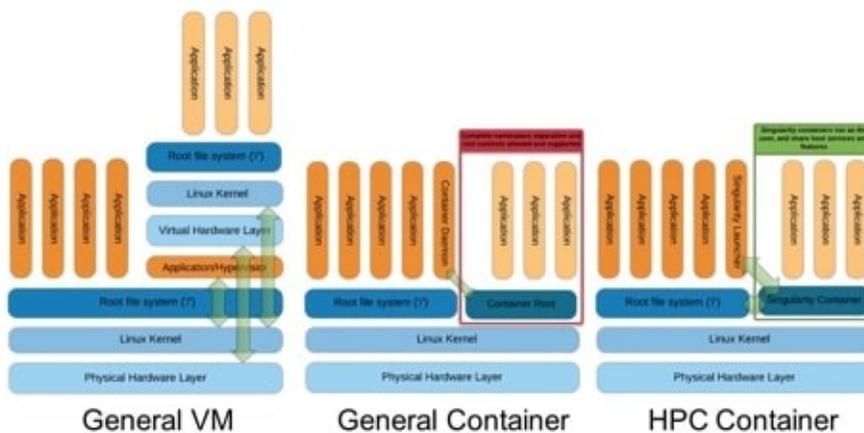


Figure 136: Singularity Container Architecture [90]

## 11.5.4 Hands-On Tutorials

The following tutorial includes links to asciinema video tutorials created by SDSC HPC Systems Manager, Trevor Cooper which allow you to see the console interactivity and output in detail. Look for the  icon like the one shown to the right corresponding to the task you are currently working on.

## 11.5.5 Downloading & Installing Singularity

- Download & Unpack Singularity
- Configure & Build Singularity
- Install & Test Singularity

### 11.5.5.1 Download & Unpack Singularity

First we download and unpack the source using the following commands (assuming your user name is `test_user` and you are working on your local computer with super user privileges):

```
[test_user@localhost ~]$ wget https://github.com/singularityware/singularity/releases/download/2.5.1/singularity-2.5.1.tar.gz tar -zxf singularity-2.5.1.tar.gz
```



[Singularity - download source and unpack in VirtualBox VM \(CentOS 7\)](#)

If the file is successfully extracted, you should be able to view the results:

```
[test_user@localhost ~]$ cd singularity-2.5.1/
[test_user@localhost singularity-2.5.1]$ ls
```

### 11.5.5.2 Configure & Build Singularity



#### [Singularity - configure and build in VirtualBox VM \(CentOS 7\)](#)

Next we configure and build the package. To configure, enter the following command (we will leave out the command prompts):

```
./configure
```

To build, issue the following command:

```
make
```

This may take several seconds depending on your computer.

### 11.5.5.3 Install & Test Singularity



#### [Singularity - install and test in VirtualBox VM \(CentOS 7\)](#)

To complete the installation enter:

```
sudo make install
```

You should be prompted to enter your admin password.

Once the installation is completed, you can check to see if it succeeded in a few different ways:

```
which singularity singularity -version
```

You can also run a selftest with the following command:

```
singularity selftest
```

The output should look something like:

```
+ sh -c test -f /usr/local/etc/singularity/singularity.conf (retval=0) OK + test -u
/usr/local/libexec/singularity/bin/action-suid (retval=0) OK + test -u /usr/local/libexec/singularity/bin/create-suid
(retval=0) OK + test -u /usr/local/libexec/singularity/bin/expand-suid (retval=0) OK + test -u
/usr/local/libexec/singularity/bin/export-suid (retval=0) OK + test -u /usr/local/libexec/singularity/bin/import-suid
(retval=0) OK + test -u /usr/local/libexec/singularity/bin/mount-suid (retval=0) OK
```

## 11.5.6 Building Singularity Containers

The process of building a Singularity container consists of a few distinct steps as follows.

- Upgrading Singularity (if needed)
- Create an Empty Container
- Import into Container
- Shell into Container
- Write into Container
- Bootstrap Container

We will go through each of these steps in detail.

### 11.5.6.1 Upgrading Singularity

We recommend building containers using the same version of Singularity, 2.5.1, as exists on Comet. This is a 2 step process.

#### Step 1: run the next script to remove your existing Singularity:

```
#!/bin/bash
#
A cleanup script to remove Singularity

sudo rm -rf /usr/local/libexec/singularity
sudo rm -rf /usr/local/etc/singularity
sudo rm -rf /usr/local/include/singularity
sudo rm -rf /usr/local/lib/singularity
sudo rm -rf /usr/local/var/lib/singularity/
sudo rm /usr/local/bin/singularity
sudo rm /usr/local/bin/run-singularity
sudo rm /usr/local/etc/bash_completion.d/singularity
sudo rm /usr/local/man/man1/singularity.1
```

#### Step 2: run the following script to install Singularity 2.5.1:

```
#!/bin/bash
#
A build script for Singularity (http://singularity.lbl.gov/)

declare -r SINGULARITY_NAME='singularity'
declare -r SINGULARITY_VERSION='2.5.1'
declare -r SINGULARITY_PREFIX='/usr/local'
declare -r SINGULARITY_CONFIG_DIR='/etc'

sudo apt update
sudo apt install python dh-autoreconf build-essential debootstrap

cd ../
tar -xzvf "${PWD}/tarballs/${SINGULARITY_NAME}-${SINGULARITY_VERSION}.tar.gz"
cd "${SINGULARITY_NAME}-${SINGULARITY_VERSION}"
./configure --prefix="${SINGULARITY_PREFIX}" --sysconfdir="${SINGULARITY_CONFIG_DIR}"
```

```
make
sudo make install
```

## 11.5.7 Create an Empty Container



[Singularity - create container](#)

To create an empty Singularity container, you simply issue the following command:

```
singularity create centos7.img
```

This will create a CentOS 7 container with a default size of ~805 Mb. Depending on what additional configurations you plan to make to the container, this size may or may not be big enough. To specify a particular size, such as ~4 Gb, include the `-s` parameter, as shown in the following command:

```
singularity create -s 4096 centos7.img
```

To view the resulting image in a directory listing, enter the following:

```
ls
```

## 11.5.8 Import Into a Singularity Container



[Singularity - import Docker image](#)

Next, we will import a Docker image into our empty Singularity container:

```
singularity import centos7.img docker://centos:7
```

## 11.5.9 Shell Into a Singularity Container



[Singularity - shell into container](#)

Once the container actually contains a CentOS 7 installation, you can ‘shell’ into it with the following:

```
singularity shell centos7.img
```

Once you enter the container you should see a different command prompt. At

this new prompt, try typing:

```
whoami
```

Your user id should be identical to your user id outside the container. However, the operating system will probably be different. Try issuing the following command from inside the container to see what the OS version is:

```
cat /etc/*-release
```

## 11.5.10 Write Into a Singularity Container



[Singularity - write into container](#)

Next, let's try writing into the container (as root):

```
sudo /usr/local/bin/singularity shell -w centos7.img
```

You should be prompted for your password, and then you should see something like the following:

```
Invoking an interactive shell within the container...
```

Next, let's create a script within the container so we can use it to test the ability of the container to execute shell scripts:

```
vi hello_world.sh
```

The previous command assumes you know the vi editor. Enter the following text into the script, save it, and quit the vi editor:

```
#!/bin/bash echo "Hello, World!"
```

You may need to change the permissions on the script so it can be executable:

```
chmod +x hello_world.sh
```

Try running the script manually:

```
./hello_world.sh
```

The output should be:

```
Hello, World!
```

## 11.5.11 Bootstrapping a Singularity Container



### [Singularity - bootstrapping a container](#)

Bootstrapping a Singularity container allows you to use what is called a ‘definitions file’ so you can reproduce the resulting container configurations on demand.

Let us say you want to create a container with Ubuntu, but you may want to create variations on the configurations without having to repeat a long list of commands manually. First, we need our definitions file. Given next is the contents of a definitions file which should suffice for our purposes.

```
Bootstrap: docker
From: ubuntu:latest
%runscript
exec echo "The runscript is the containers default runtime command!"

%files
/home/testuser/ubuntu.def /data/ubuntu.def

%environment
VARIABLE=HELLOWORLD
Export VARIABLE
%labels
AUTHOR testuser@sdsc.edu

%post
apt-get update && apt-get -y install python3 git wget
mkdir /data
echo "The post section is where you can install and configure your container."
```

To bootstrap your container, first we need to create an empty container.

```
singularity create -s 4096 ubuntu.img
```

Now, we simply need to issue the following command to configure our container with Ubuntu:

```
sudo /usr/local/bin/singularity bootstrap ./ubuntu.img ./ubuntu.def
```

This may take a while to complete. In principle, you can accomplish the same result by manually issuing each of the commands contained in the script file, but why do that when you can use bootstrapping to save time and avoid errors.

If all goes according to plan, you should then be able to shell into your new Ubuntu container.

## 11.5.12 Running Singularity Containers on Comet

Of course, the purpose of this tutorial is to enable you to use the San Diego Supercomputer Center's Comet supercomputer to run your jobs. This assumes you have an account on Comet already. If you do not have an account on Comet and you feel you can justify the need for such an account (i.e. your research is limited by the limited compute power you have in your government-funded research lab), you can request a 'Startup Allocation' through the XSEDE User Portal:

<https://portal.xsede.org/allocations-overview#types-trial>

You may create a free account on the XUP if you do not already have one and then proceed to submit an allocation request at the previously given link.

NOTE: SDSC provides a [Comet User Guide](#) to help get you started with Comet. Learn more about The San Diego Supercomputer Center at <http://www.sdsc.edu>.

This tutorial walks you through the following four steps towards running your first Singularity container on Comet:

- Transfer the Container to Comet
- Run the Container on Comet
- Allocate Resources to Run the Container
- Integrate the Container with Slurm
- Use existing Comet Containers

### 11.5.12.1 Transfer the Container to Comet



[Singularity - transfer container to Comet](#)

Once you have created your container on your local system, you will need to transfer it to Comet. There are multiple ways to do this and it can take a varying amount of time depending on its size and your network connection speeds.

To do this, we will use scp (secure copy). If you have a Globus account and your containers are more than 4 Gb you will probably want to use that file transfer

method instead of scp.

Browse to the directory containing the container. Copy the container to your scratch directory on Comet. By issuing the following command:

```
scp ./centos7.img comet.sdsc.edu:/oasis/scratch/comet/test_user/temp_project/
```

The container is ~805 Mb so it should not take too long, hopefully.

### 11.5.12.2 Run the Container on Comet



#### [Singularity - run container on Comet](#)

Once the file is transferred, login to Comet (assuming your Comet user is named `test_user`):

```
ssh test_user@comet.sdsc.edu
```

Navigate to your scratch directory on Comet, which should be something like:

```
[test_user@comet-ln3 ~]$ cd /oasis/scratch/comet/test_user/temp_project/
```

Next, you should submit a request for an interactive session on one of Comet's compute, debug, or shared nodes.

```
[test_user@comet-ln3 ~]$ srun --pty --nodes=1 --ntasks-per-node=24 -p compute -t 01:00:00 --wait 0 /bin/bash
```

Once your request is approved your command prompt should reflect the new node id.

Before you can run your container you will need to load the Singularity module (if you are unfamiliar with modules on Comet, you may want to review the Comet User Guide). The command to load Singularity on Comet is:

```
[test_user@comet-ln3 ~]$ module load singularity
```

You may issue the previous command from any directory on Comet. Recall that we added a `hello_world.sh` script to our `centos7.img` container. Let us try executing that script with the following command:

```
[test_user@comet-ln3 ~]$ singularity exec /oasis/scratch/comet/test_user/temp_project/singularity/centos7.img /hello_world.sh
```

If all goes well, you should see *Hello, World!* in the console output. You might also see some warnings pertaining to non-existent bind points. You can resolve this by adding some additional lines to your definitions file before you build your container. We did not do that for this tutorial, but you would use a command like the following in your definitions file:

```
create bind points for SDSC HPC environment mkdir /oasis /scratch/ /comet /temp_project
```

You will find additional examples located in the following locations on Comet:

```
/share/apps/examples/SI2017/Singularity
```

and

```
/share/apps/examples/SINGULARITY
```

### 11.5.12.3 Allocate Resources to Run the Container



#### [Singularity - allocate resources to run container](#)

It is best to avoid working on Comet's login nodes since they can become a performance bottleneck not only for you but for all other users. You should rather allocate resources specific for computationally-intensive jobs. To allocate a 'compute node' for your user on Comet, issue the following command:

```
[test_user@comet-ln3 ~]$ salloc -N 1 -t 00:10:00
```

This allocation requests a single node (-N 1) for a total time of 10 minutes (-t 00:10:00). Once your request has been approved, your computer node name should be displayed, e.g. comet-17-12.

Now you may login to this node:

```
[test_user@comet-ln3 ~]$ ssh comet-17-12
```

Notice that the command prompt has now changed to reflect the fact that you are on a compute node and not a login node.

```
[test_user@comet-06-04 ~]$
```

Next, load the Singularity module, shell into the container, and execute the `hello_world.sh` script:

```
[test_user@comet-06-04 ~]$ module load singularity [test_user@comet-06-04 ~]$ singularity shell centos7.img
[test_user@comet-06-04 ~]$./hello_world.sh
```

If all goes well, you should see *Hello, World!* in the console output.

### 11.5.12.4 Integrate the Container with Slurm



#### [Singularity - run container on Comet via Slurm](#)

Of course, most users simply want to submit their jobs to the Comet queue and let it run to completion and go on to other things while waiting. Slurm is the job manager for Comet.

Given next is a job script (which we will name `singularity_mvapich2_hellow.run`) which will submit your Singularity container to the Comet queue and run a program, `hellow.c` (written in C using MPI and provided as part of the examples with the `mvapich2` default installation).

```
#!/bin/bash ` ` #SBATCH --job-name="singularity_mvapich2_hellow" #SBATCH --output="singularity_mvapich2_hellow.%j.out"
#SBATCH --error="singularity_mvapich2_hellow.%j.err" #SBATCH --nodes=2 #SBATCH --ntasks-per-node=24 #SBATCH --
time=00:10:00 #SBATCH --export=all module load mvapich2_ib singularity

CONTAINER=/oasis/scratch/comet/$USER/temp_project/singularity/centos7-mvapich2.img

mpirun singularity exec ${CONTAINER} /usr/bin/hellow
```

The previous script requests 2 nodes and 24 tasks per node with a wall time of 10 minutes. Notice that two modules are loaded (see the line beginning with ‘module’), one for Singularity and one for MPI. An environment variable ‘CONTAINER’ is also defined to make it a little easier to manage long reusable text strings such as file paths.

You may need to add a line specifying with allocation to be used for this job. When you are ready to submit the job to the Comet queue, issue the following command:

```
[test_user@comet-06-04 ~]$ sbatch -p debug ./singularity_mvapich2_hellow.run
```

To view the status of your job in the Comet queue, issue the following:

```
[test_user@comet-06-04 ~]$ squeue -u test_user
```

When the job is complete, view the output which should be written to the output

file `singularity_mvapich2_hello.%j.out` where `%j` is the job ID (let's say the job ID is 1000001):

```
[test_user@comet-06-04 ~]$ more singularity_mvapich2_hello.1000001.out
```

The output should look something like the following:

```
Hello world from process 28 of 48
Hello world from process 29 of 48
Hello world from process 30 of 48
Hello world from process 31 of 48
Hello world from process 32 of 48
Hello world from process 33 of 48
Hello world from process 34 of 48
Hello world from process 35 of 48
Hello world from process 36 of 48
Hello world from process 37 of 48
Hello world from process 38 of 48
```

### 11.5.12.5 Use Existing Comet Containers

SDSC User Support staff, Marty Kandes, has built several custom Singularity containers designed specifically for the Comet environment.

[Learn more about these containers for Comet.](#)

An easy way to pull images from the singularity hub on comet is provided in the next video:



[Singularity - pull from singularity-hub on Comet](#)

Comet supports the capability to pull a container directly from any properly configured remote singularity hub. For example, the following command can pull a container from the hpcdevops singularity hub straight to an empty container located on Comet:

```
comet$ singularity pull shub://hpcdevops/singularity-hello-world:master
```

The resulting container should be named something like `singularity-hello-world.img`.

Learn more about Singularity Hubs and container collections at:

<https://singularity-hub.org/collections>

That's it! Congratulations! You should now be able to run Singularity containers on Comet either interactively or through the job queue. We hope you found this tutorial useful. Please contact [support@xsede.org](mailto:support@xsede.org) with any questions you might have. Your Comet-related questions will be routed to the amazing SDSC Support Team.

## 11.5.13 Using Tensorflow With Singularity

One of the more common advantages of using Singularity is the ability to use pre-built containers for specific applications which may be difficult to install and maintain by yourself, such as Tensorflow. The most common example of a Tensorflow application is character recognition using the MNIST dataset. You can learn more about this dataset at <http://yann.lecun.com/exdb/mnist/>.

XSEDE's Comet supercomputer supports Singularity and provides several pre-built container which run Tensorflow. Given next is an example batch script which runs a Tensorflow job within a Singularity container on Comet. Copy this script and paste it into a shell script named `mnist_tensorflow_example.sb`.

```
#!/bin/bash
#SBATCH --job-name="TensorFlow"
#SBATCH --output="TensorFlow.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:k80:1
#SBATCH -t 01:00:00
```

## 11.5.14 Run the job

```
module load singularity
singularity exec
/share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img lsb_release
-a
singularity exec
/share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img python -m
tensorflow.models.image.mnist.convolutional
```

To submit the script to Comet, first you'll need to request a compute node with the following command (replace account with your XSEDE account number):

```
[test_user@comet-ln3 ~]$ srun --account=your_account_code --partition=gpu-shared --gres=gpu:1 --pty --nodes=1 --ntasks-pe
```

To submit a job to the Comet queue, issue the following command:

```
[test_user@comet-06-04 ~]$ sbatch mnist_tensorflow_example.sb
```

When the job is done you should see an output file in your output directory containing something resembling the following:

```
Distributor ID: Ubuntu
Description: Ubuntu 16.04 LTS
Release: 16.04
Codename: xenial
^[[33mWARNING: Non existent bind point (directory) in container: '/scratch'
^[[0mI tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 0000:85:00:0
Total memory: 11.17GiB
Free memory: 11.11GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0: Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80)
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Initialized!
Step 0 (epoch 0.00), 40.0 ms
Minibatch loss: 12.054, learning rate: 0.010000
Minibatch error: 90.6%
Validation error: 84.6%
Step 100 (epoch 0.12), 12.6 ms
Minibatch loss: 3.293, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 7.0%

Step 8400 (epoch 9.77), 11.5 ms
Minibatch loss: 1.596, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Step 8500 (epoch 9.89), 11.5 ms
Minibatch loss: 1.593, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.8%
Test error: 0.9%
```

Congratulations! You have successfully trained a neural network to recognize ascii numeric characters.

## 11.5.15 Resources

- Docker documentation <https://docs.docker.com/>
- Kubernetes documentation <https://kubernetes.io/docs/home/>
- Container Orchestration Tools: Compare Kubernetes vs Docker Swarm <https://platform9.com/blog/compare-kubernetes-vs-docker-swarm/>
- Gentle introduction to Containers <https://www.slideshare.net/jpetazzo/introduction-docker-linux-containers-lxc>

### 11.5.15.1 Tutorialspoint

Several tutorials on docker that can help you understand the concepts in more detail

- <https://www.tutorialspoint.com/docker/index.htm>
- [https://www.tutorialspoint.com/docker/docker\\_tutorial.pdf](https://www.tutorialspoint.com/docker/docker_tutorial.pdf)
- [https://www.tutorialspoint.com/docker/docker\\_pdf\\_version.htm](https://www.tutorialspoint.com/docker/docker_pdf_version.htm)

## 11.6 EXERCISES

---

### E.Docker.1: MongoDB Container

*Develop a docker file that uses the mongo distribution from Dockerhub and starts a MongoDB database on the regular port while communicating to your container.*

*What are the parameters on the command line that you need to define?*

### E.Docker.2: MongoDB Container with authentication

*Develop a MongoDB container that includes an outhenticated user. You must use the cloudmesh.yaml file for specifying the information for the admin user and password.*

1. *How do you add the user?*
2. *How do you start the container?*
3. *Showcase the use of the authentication with a simple script or pytest.*

*You are allowed tou sue docker compose, but make sure you read the password ond username from the yaml file. YoU must not configure it by hand in the compose yaml file. You can use cloudmesh commands to read the username and password.*

```
cms config value cloudmesh.data.mongo.MONGO_USERNAME
cms config value cloudmesh.data.mongo.MONGO_PASSWORD
```

## E.Docker.3: Cloudmesh Container

*In this assignment we will explore the use of two containers. We will be leveraging the assignment E.Docker.2.*

*First, you will start the authenticated docker MongoDB container*

*You will be writing an additional dockerfile, that creates cloudmesh in a docker container. Upon start the parameter passed to the container will be executed in the container. You will use the .ssh and .cloudmesh directory from your native file system.*

*For hints, please look at*

- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/docker/ubuntu-19.04/Dockerfile>
- <https://github.com/cloudmesh/cloudmesh-cloud/blob/master/docker/ubuntu-19.04/Makefile>

*To jump start you try*

```
make image
make shell
```

*Explore! Understand what is done in the Makefile*

*Questions:*

1. *How would you need to modify the Dockerfile to complete it?*
2. *Why did we outcomment the MongoDB related tasks in the Dockerfile?*
3. *How do we need to establish communication to the MongoDB container*
4. *Could docker compose help, or would it be too complicated, e.g. what if the mongo container already runs?*
5. *Why would it be dangerous to store the cloudmesh.yaml file inside the container? Hint: DockerHub.*
6. *Why should you at this time not upload images to DockerHub?*

## E.Docker.Swarm.1: Documentation

*Develop a section in the handbook that deploys a Docker Swarm cluster on a number of ubuntu machines. Note that this may actually be easier as docker and docker swarm are distributed with recent versions of ubuntu. Just in case we are providing a link to an effort we found to install docker swarm. However we have not checked it or identified if it is useful.*

- <https://rominirani.com/docker-swarm-tutorial-b67470cf8872>

#### E.Docker.Swarm.2: Google Compute Engine

*Develop a section that deploys a Docker Swarm cluster on Google Compute Engine. Note that this may actually be easier as docker and docker swarm are distributed with recent versions of ubuntu. Just in case we are providing a link to an effort we found to install docker swarm. However we have not checked it or identified if it is useful.*

- <https://rominirani.com/docker-swarm-on-google-compute-engine-364765b400ed>

#### E.SingleNodeHadoop:

*Setup a single node hadoop environment.*

*This includes:*

1. *Create a Dockerfile that deploys hadoop in a container*
2. *Develop sample applications and tests to test your cluster. You can use wordcount or similar.*

*you will find a comprehensive installation instruction that sets up a hadoop cluster on a single node at*

- <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html>

#### E.MultiNodeHadoop:

*Setup a hadoop cluster in a distributed environment.*

*This includes:*

- 1. Create docker compose and Dockerfiles that deploys hadoop in kubernetes*
- 2. Develop sample applications and tests to test your cluster. You can use wordcount or similar.*

*you will find a comprehensive installation instruction that sets up a hadoop cluster in a distributed environment at*

- <https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-common/ClusterSetup.html>*

*You can use this set of instructions or identify other resources on the internet that allow the creation of a hadoop cluster on kubernetes. Alternatively you can use docker compose for this exercise.*

## **E. SparkCluster: Documentation**

*Develop a high quality section that installs a spark cluster in kubernetes. Test your deployment on minikube and also on Futuresystems echo.*

*You may want to get inspired from the talk Scalable Spark Deployment using Kubernetes:*

- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-1/>*
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-2/>*
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-3/>*
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-4/>*
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-5/>*
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-6/>*

- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-7/>
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-8/>
- <http://blog.madhukaraphatak.com/scaling-spark-with-kubernetes-part-9/>

*Make sure you do not plagiarize.*

## 12 SERVERLESS

### 12.1 FAAS

---

#### 12.1.1 Introduction

FaaS or Function as a service is a new paradigm in cloud computing that is gaining popularity recently. FaaS is also known as Serverless Computing to some. While the name Serverless implies that no servers are involved this is not true. So FaaS would be a better term to describe this technology. FaaS is built around functions and events. Functions are generally stateless and are executed within isolated containers. The execution of the functions can be thought of as an event driven model. A program or application in FaaS consist of a set of functions and a set of events or triggers that invoke or activate those functions. A function activation can result in another function activation as a result.

Generally FaaS specifies a set of constraints on what a function can be. The constraints include storage constraints such as a maximum size for the function, maximum memory allowed, execution time, etc. The exact constraints differ from provider to provider. AWS Lambda is considered as one of first FaaS offerings. Now most cloud providers offer their own version of FaaS. Several popular FaaS providers are listed next.

#### 12.1.2 Serverless Computing

In Serverless Computing, servers are still there, its just that we dont need to manage them.

Another advantage of going serverless is that you no longer need to keep a server running all the time. The *server* suddenly appears when you need it, then disappears when you're done with it. Now you can think in terms of functions instead of servers, and all your business logic can now live within these functions.

#### 12.1.3 Faas provider

- AWS Lambda <https://aws.amazon.com/lambda>
- Azure Functions <https://azure.microsoft.com/en-us/services/functions>
- IBM Cloud Functions <https://console.bluemix.net/openwhisk>
- Google Cloud Functions <https://cloud.google.com/functions>
- Iron.io <https://www.iron.io>
- Webtask.io <https://webtask.io>

Other than the providers there are also several open source FaaS offerings that are available to be used. One of the most complete and popular open source option would be Apache OpenWhisk, which was developed by IBM and later open sourced. IBM currently deploys OpenWhisk in IBM cloud and offers it as a IBM Cloud functions.

- OpenWhisk <https://github.com/apache/incubator-openwhisk>
- Funktion <https://github.com/funktionio/funktion>
- Iron Functions <https://github.com/iron-io/functions>
- Kubeless <https://github.com/kubeless/kubeless>
- Fission <https://github.com/fission/fission>
- FaaS-netes <https://github.com/alexellis/faas-netes>

There are many articles and tutorials online that provide very good information regarding FaaS. Given next are some such resources that provide introductions and some example usecase's of FaaS

#### 12.1.4 Resources

- <https://stackify.com/function-as-a-service-serverless-architecture>
- [https://en.wikipedia.org/wiki/Serverless\\_computing](https://en.wikipedia.org/wiki/Serverless_computing)
- <https://azure.microsoft.com/en-us/overview/serverless-computing>
- <https://aws.amazon.com/serverless>
- <https://aws.amazon.com/lambda>
- <https://www.infoworld.com/article/3093508/cloud-computing/what-serverless-computing-really-means.html>
- <https://techbeacon.com/aws-lambda-serverless-apps-5-things-you-need-know-about-serverless-computing>
- <https://blog.alexellis.io/introducing-functions-as-a-service>

## 12.1.5 Usage Examples

- <https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda>
- <https://blog.alexellis.io/first-faaS-python-function>

## 12.2 AWS LAMBDA

---

---



### Learning Objectives

- Learn about AWS lambda
  - Try out AWS Lambda practically
- 
- 

AWS Lambda is considered as one of the earliest FaaS implementations made available to end users. AWS Lambda provides a rich set of features that can be leveraged by users to build programs and applications on top of the FaaS framework. AWS Lambda supports many event types that developers can use to orchestrate their FaaS applications. And in line with the FaaS model AWS Lambda only charges users for actual execution time of the functions. For example if you host and deploy a function on AWS Lambda and only execute it for 1 minute every day you will only be charged for the 1 minute execution time.

AWS does not share how the internals of AWS Lambda work in detail but as with any general FaaS framework it should be leveraging various container technologies underneath. You can get a better understanding on how the internals of a FaaS framework is organized by looking at the [OpenWhisk Section](#)

### 12.2.1 AWS Lambda Features

AWS Lambda provides many features that allows the creation of an FaaS application to be straight forward. An FaaS application normally consist of a set of functions and a set of events that activate or invoke those functions. AWS Lambda supports several programming languages that allow developers to develop the function they require in any of the programming languages that are supported. The following list show the programming languages that are currently supported by AWS Lambda for function creation.

- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Go

Other than the functions the most important requirement for a good FaaS framework is a rich set of function invocation methods which allow users to tie together different events that happen in the echo system with the FaaS application. In this regard AWS Lambda supports many event sources, mainly from the AWS echo system. AWS documentation provides a complete list of supported event sources at [AWS Lambda event sources](#). For example a developer can configure an function to be invoked when a S3 bucket is updated, or configure an function to be invoked based on inputs received by Amazon Alexa, etc.

⊗ use also bibtex

## 12.2.2 Understanding Function limitations

Before you start working on FaaS frameworks it is important to understand the limitations and restrictions that are applied to functions. The limits and restrictions discussed in the section are applicable to most FaaS frameworks but the exact values may differ based on the FaaS vendor. However the reason for most of the limitations are to maintain performance requirements. We will discuss several major limits next. For a complete list of limits in AWS Lambda please refer to the limits documentation [AWS Lambda Function Limits](#)

⊗ use also bibtex

### 12.2.2.1 Execution Time

AWS Lambda limits the execution length of a function. Currently it is set to 15 minutes but was set at 300s previously, so the function limits have and may change with time. Other FaaS provides have different values for execution time limits, but in general each FaaS provider does define a execution time limit

### 12.2.2.2 Function size

AWS Lambda also sets several memory and storage limits for functions. Currently the maximum memory allocated to a function is 3008MB and the maximum allocated storage space is 512MB. However it is good to keep in mind that monetary charge for function execution increases with the amount of memory that is specified for the function.

### 12.2.3 Understanding the free Tier

If new users want to experiment with AWS lambda, AWS does provide a free tier for AWS Lambda, which include 1 million function invocations per month. You can find a more detailed description of the free tier in the AWS docs [AWS Lambda Pricing](#).

⊗ use also bibtex

### 12.2.4 Writing your fist Lambda function

With the GUI interface it is relatively easy to try out your first Serverless function with AWS Lambda. Please follow the steps defined at [How to run your first AWS Lambda function in the cloud](#) (this link does not exist any longer)

⊗ use also bibtex

### 12.2.5 AWS Lambda Usecases

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. It is a computing service that runs code in response to events, runs the code that has been loaded into the system and automatically manages the computing resources required by that code. According to the the Lambda product page

*“AWS Lambda lets you run code without provisioning or managing servers.”* [Aws](#) (⊗ refernce to bibtex)

For example, one of the use-cases would be that everytime AWS Lambda could resize the picture, after it is uploaded onto AWS S3 system and rendered on different devices like phone, ipad or desktop. The event that triggers the Lambda function is the file being uploaded to S3. Lambda then executes the function of resizing the image. The Seattle Times uses the AWS Lambda to automatically resize the images.

One key point to note here is that Amazon charges only when the functions are executed. So, The Seattle Times is charged for this service only when the images are been resized. Lambda can be used for Analytics. So lets say, there has been a purchase of a house on zillow, this data can be saved into a NoSQL database and this entry into the database is an event which can trigger Lambda function to load the order information into Amazon Redshift. Then we can run Analytics on top of this data. We can also build serverless applications composed of functions that are triggered by events and automatically deploy them using AWS CodePipeline and AWS CodeBuild. For more information, see Deploying Lambda-based Applications.

There are development groups or companies mainly startups, where they want to just focus on their application development without wanting to care about their infrastructure and they also want that they pay for what they use. Hence, AWS Lambda comes into play which satisfies all their needs.

Ironically, Lambda could be a threat to one of the Amazon's most popular EC2. Developers can build apps that run entirely on Lambda functions instead of spinning up EC2 VMs. Amazon may be out-innovating itself with Lambda.

In AWS Lambda, we have triggers. Lambda Functions can be triggered in different ways: an HTTP request, a new document upload to S3, a scheduled Job, an AWS Kinesis data stream, or a notification from AWS Simple Notification Service (SNS).

## **12.2.6 AWS Lambda Example**

Let us create our first Lambda function.

Step 1: The very first thing we need is an AWS account. (There is already a section on this, please go through that to understand how to create an AWS

account - [Creating AWS account](#)

Step 2: We will be writing a function that we call `isPalindrome`, which will check if the string is palindrome or not.

```
function isPalindrome(string) {
 const reverse = string.split('').reverse().join('');
 const isPalindrome = (string === reverse);
 const result = isPalindrome ? `${string} is a Palindrome` : `${string} is not a Palindrome`;
 return result;
}

document.write(isPalindrome('abcd'));
```

This example we store in a file as javascript named `isPalindrome.js`

Step 3: Let's see how to create an AWS Lambda function - `isPalindrome`. Firstly, go to AWS Console. (see [Figure 137](#)).

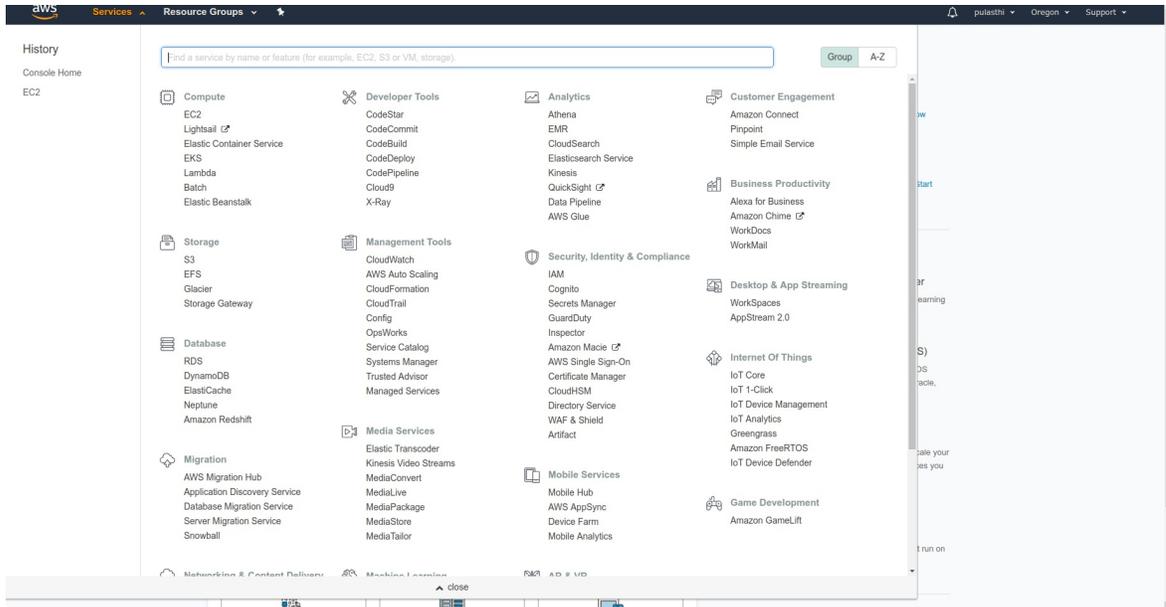


Figure 137: AWS Console

Step 4: Now we will select AWS Lambda from console and then click on **Get Started Now** (see [Figure 138](#))

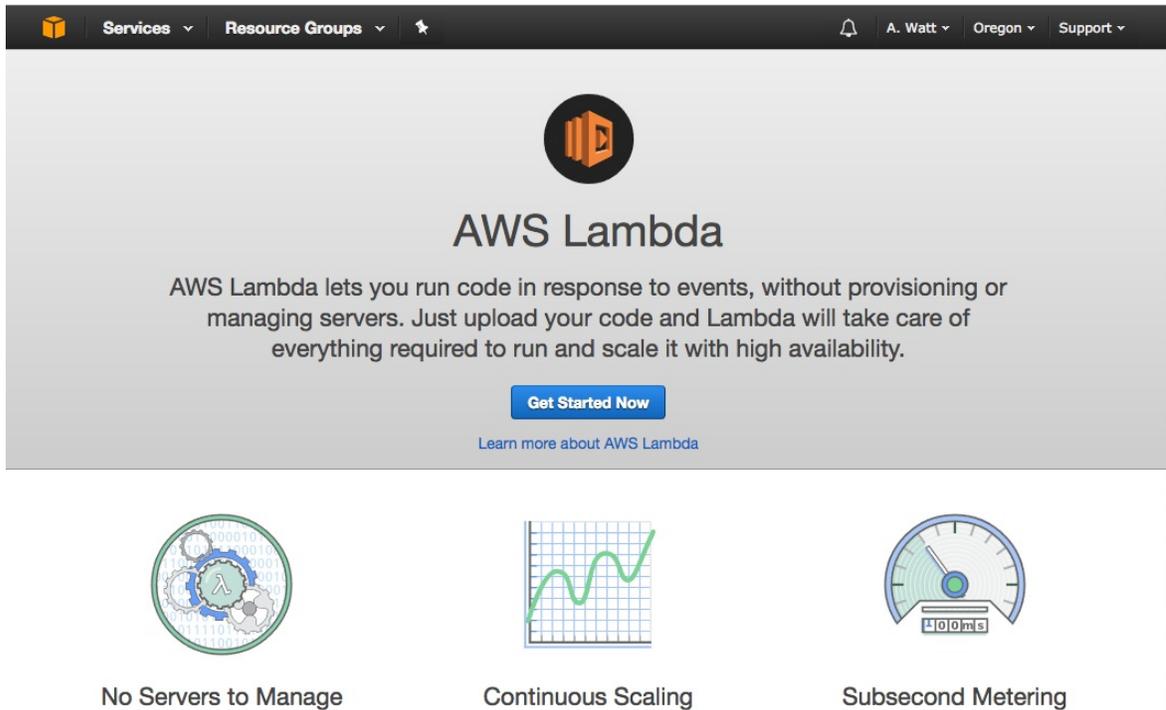


Figure 138: AWS Lambda

Step 5: For runtime, we will select Node.js 6.10 and then press “Blank Function.” (see [Figure 139](#)).

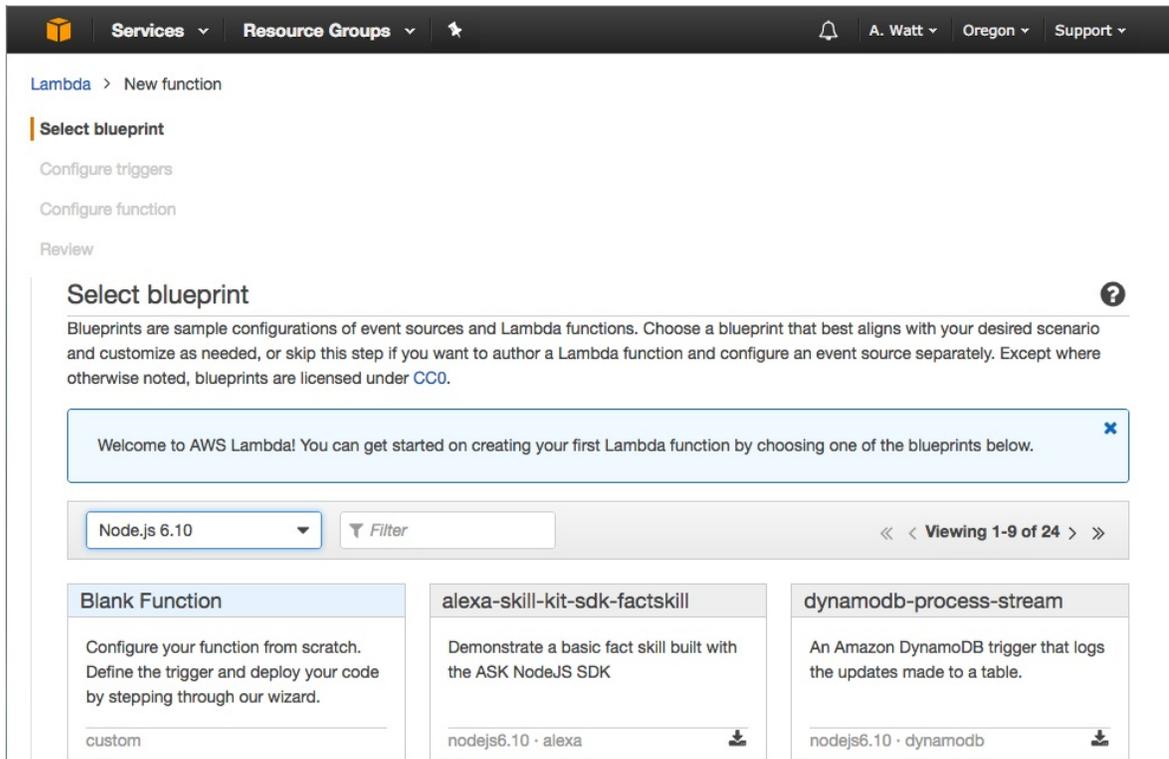


Figure 139: Blank Function

Step 6: We will skip this step and press **Next**. (see [Figure 140](#))

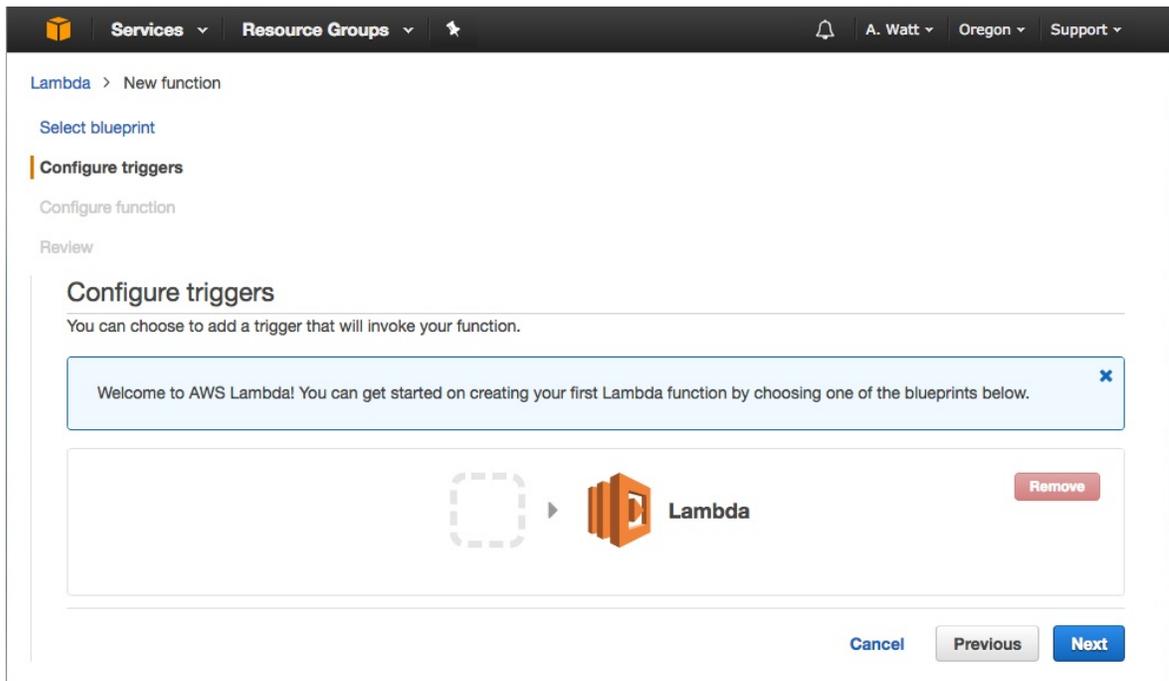
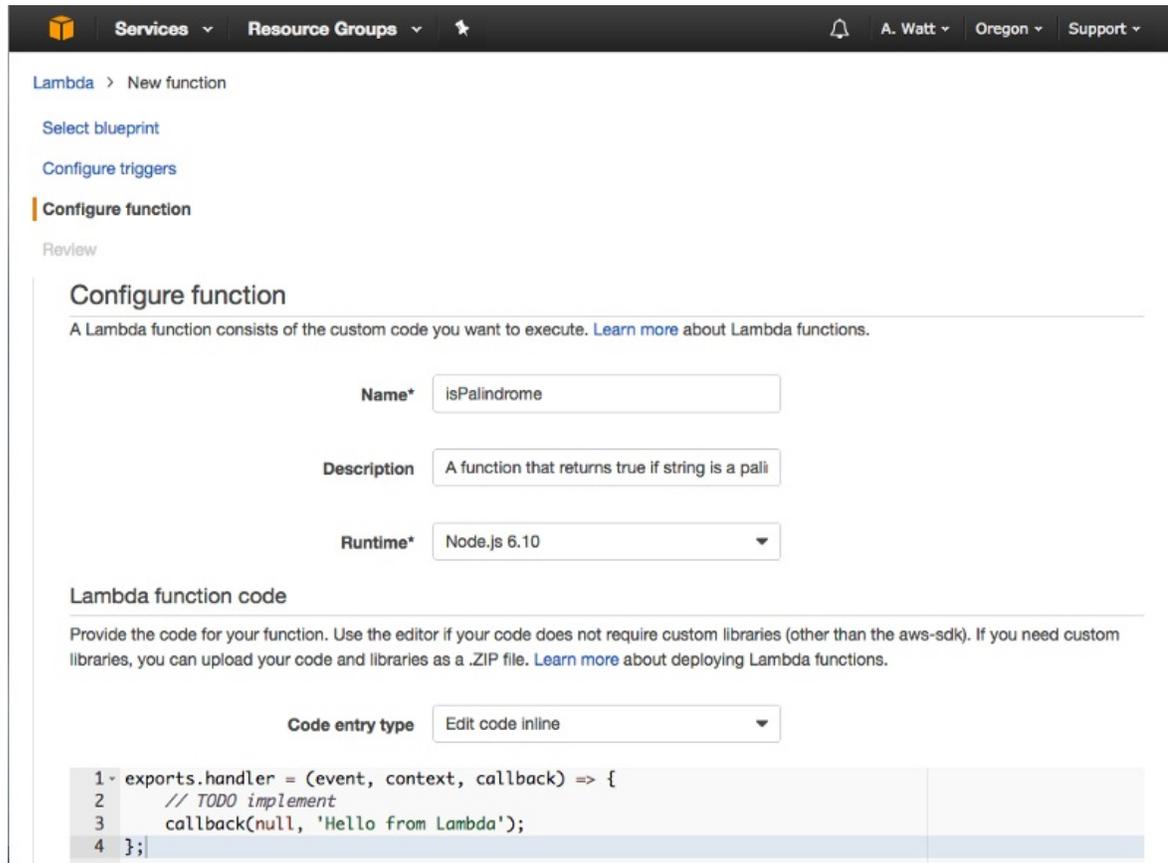


Figure 140: Next

Step 7: Let's give the Name as isPalindrome and put in a description of our new Lambda Function, or we can leave it blank. (see [Figure 141](#))



The screenshot shows the AWS Lambda console interface for configuring a new function. The breadcrumb navigation is 'Lambda > New function'. The main heading is 'Configure function'. Below this, there are three input fields: 'Name\*' with the value 'isPalindrome', 'Description' with the value 'A function that returns true if string is a pali', and 'Runtime\*' with a dropdown menu showing 'Node.js 6.10'. Below these fields is a section for 'Lambda function code' with a text area containing a JavaScript code snippet. The code entry type is set to 'Edit code inline'.

```
1 exports.handler = (event, context, callback) => {
2 // TODO implement
3 callback(null, 'Hello from Lambda');
4 };
```

Figure 141: Description

Lambda function is just a function, named as *handler* here and the function takes three parameter - event, context and a callback function. The callback will run when the Lambda function is done and will return a response or an error message. For the Blank Lambda blueprint, response is hard-coded as the string

Hello from Lambda.

Step 8: Please scroll down for choosing the Role “Create new Role from template”, and for Role name we are going to use isPalindromeRole in our case. For Policy templates, we will choose “Simple Microservice” permissions. (see [Figure 142](#))

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#). For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.

Enable encryption helpers

Environment variables 

Key	Value

 ✕

#### Lambda function handler and role

Handler\*  ⓘ

Role\*  ⓘ

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

Role name\*  ⓘ

Policy templates  ⓘ

▸ Tags

▸ Advanced settings

\* These fields are required.

Cancel

Previous

Next

Figure 142: Policy

Step 9: For Memory, 128 megabytes is more than enough for our simple function. As for the 3 second timeout, this means that—should the function not return within 3 seconds- AWS will shut it down and return an error. Three seconds is also more than enough. Leave the rest of the advanced settings unchanged (see [aws-lambda-settings](#)).

▼ **Advanced settings**

These settings allow you to control the code execution performance and costs for your Lambda function. Changing your resource settings (by selecting memory) or changing the timeout may impact your function cost. [Learn more](#) about how Lambda pricing works.

**Memory (MB)\*** 128 ▼ ⓘ

**Timeout\*** 0 min 3 sec

---

AWS Lambda will automatically retry failed executions for asynchronous invocations. You can additionally optionally configure Lambda to forward payloads that were not processed to a dead-letter queue (DLQ), such as an SQS queue or an SNS topic. [Learn more](#) about Lambda's [retry policy](#) and [DLQs](#). **Please ensure your role has appropriate permissions to access the DLQ resource.**

**DLQ Resource** Select resource ▼ ⓘ

---

All AWS Lambda functions run securely inside a default system-managed VPC. However, you can optionally configure Lambda to access resources, such as databases, within your custom VPC. [Learn more](#) about accessing VPCs within Lambda. **Please ensure your role has appropriate permissions to configure VPC.**

**VPC** No VPC ▼ ⓘ

---

**AWS X-Ray** provides tracing and monitoring capabilities for your Lambda function. [Click here](#) to learn more.

**Enable active tracing**  ⓘ

---

Environment variables are encrypted at rest using a default Lambda service key. You can change the key below to one of your account's keys or paste in a full KMS key ARN.

**KMS key** (default) aws/lambda ▼ ⓘ

\* These fields are required.

[Cancel](#) [Previous](#) [Next](#)

Figure 143: Advanced Settings [Source](#)

Step 10: Let's click on the "Create function" button now to create our first Lambda function. (see [Figure 144](#))

## Review

### Review

Please review your Lambda function details. You can go back to edit changes for each section. When you are ready, click **Create function** to complete the setup process.

#### Lambda function

[Edit](#)

**Name** isPalindrome

**Description** A function that returns true if string is a palindrome

**Runtime** Node.js 6.10

#### Environment variables

**Handler** index.handler

**Role name\*** isPalindromeRole

**Policy templates** Simple Microservice permissions

**Tags**

**DLQ Resource**

**Memory (MB)** 128

**Timeout** 3

**VPC** No VPC

**Enable active tracing**

**KMS key** (default) aws/lambda

[Cancel](#)[Previous](#)[Export function](#)[Create function](#)

Figure 144: Create

Step 11: Now that we have created our first Lambda function, let's test it by clicking **Test** (see [Figure 145](#))

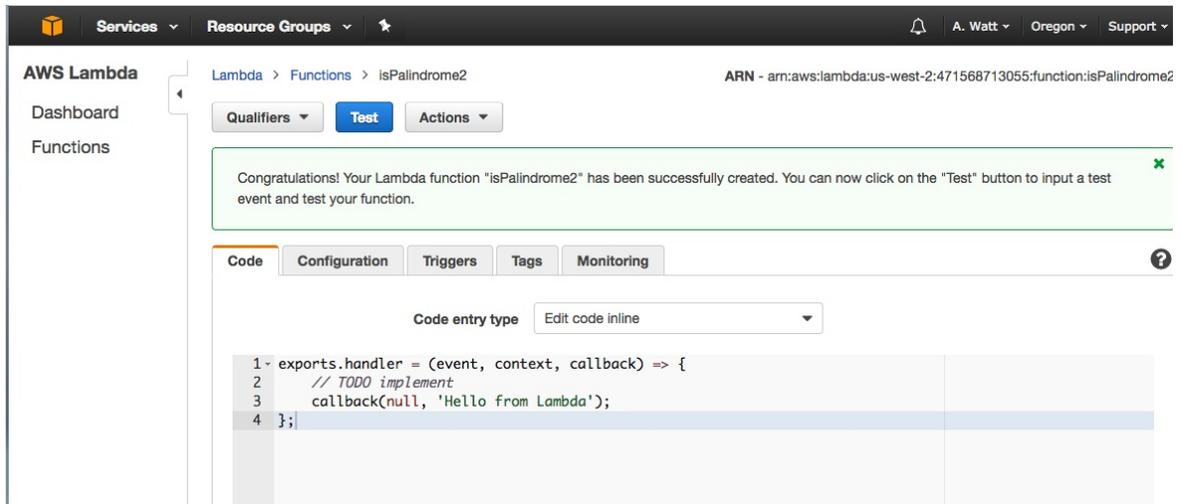


Figure 145: Test

The output will be the hard-coded response of `Hello from Lambda.` from the created Lambda function. (see [Figure 146](#))

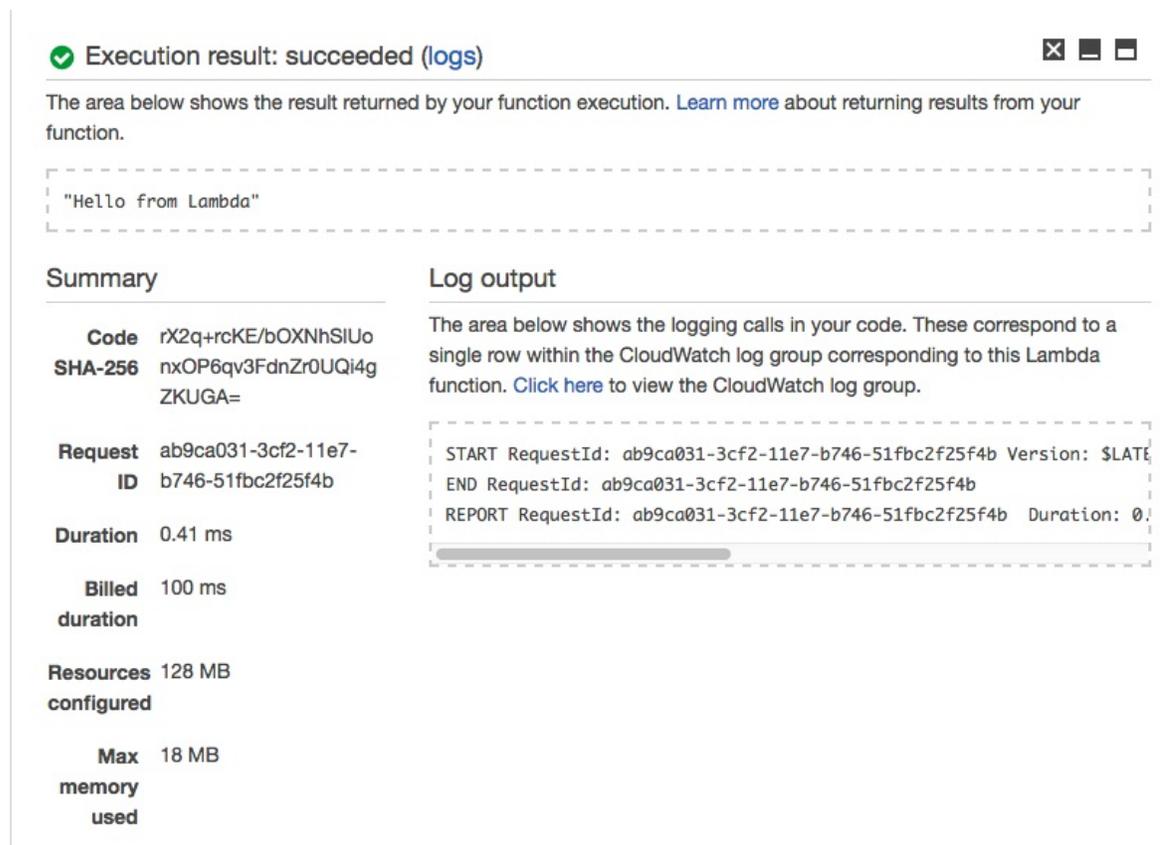


Figure 146: Hello

Step 12: Now let us add our `isPalindrome.js` function code here to Lambda function

but instead of return result use `callback(null, result)`. Then add a hard-coded string value of `abcd` on line 3 and press `Test`. (see [Figure 147](#))

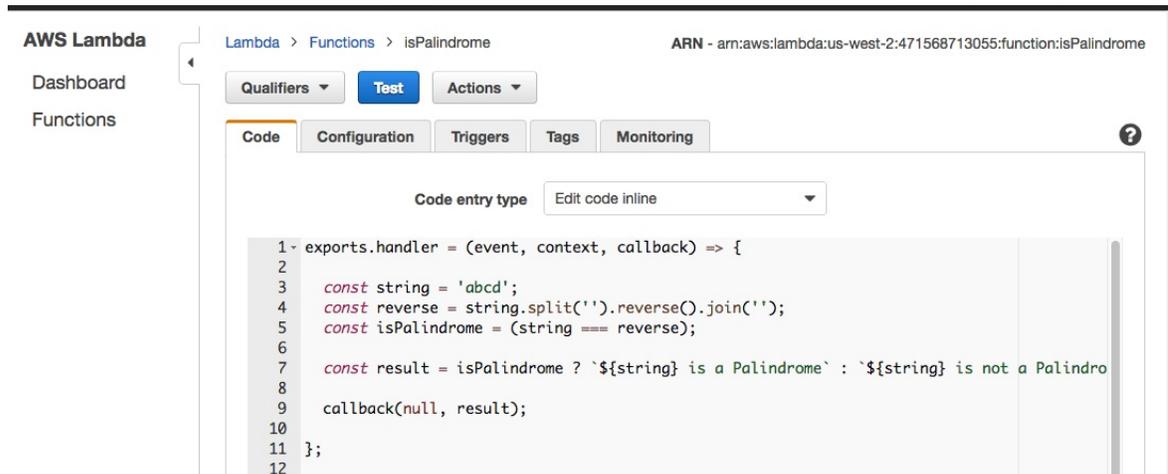


Figure 147: Press Test

The output will be `abcd is not a Palindrome` (see [Figure 148](#))



Figure 148: Output

Similarly, let us try with string `abcdcba` and in this case output should return `abcdcba is a Palindrome`. Thus, our Lambda function is behaving as expected.

## 12.3 APACHE OPENWHISK

 this section includes many refernces to other tools, that need bibtex refernces.

Apache OpenWhisk is a Function as a Service (FaaS), aka Serverless computing, platform used to execute code in response of an events via triggers by managing the infrastructure, servers and scaling. The advantage of OpenWhisk over traditional long-running VM or container approach is that there is lack of resiliency-related overhead in OpenWhisk. OpenWhisk is inherently scalable since the actions are executed on demand. OpenWhisk also helps the developers to focus only on coding by taking care of infrastructure-related tasks like monitoring and patching.

The developers provide the code written in the desired programming language for the desired action and this code will be executed in response to the events. The *triggering* can be invoked using HTTP requests or external feeds. The events invoking the triggers ranges from database modification to new variables in IoT sensors. Actions that response to these events could also range from a Python code snippet to a binary code in a container and it is as well possible to chain the actions. Note that these actions are deployed and executed instantaneously and can be invoked not only by triggers but also using the OpenWhisk API or CLI.

### **12.3.1 OpenWhisk Workflow**

OpenWhisk uses Nginx, Kafka, Docker and CouchDB as internal components. To understand the role of each of these components, let's review an action invocation trace in the system. Remember the main outcome of OpenWhisk (or Serverless architecture in general) is to execute the user's code inside the system and return the result. The workflow of the OpenWhisk is illustrated in the figure [Figure 149](#)

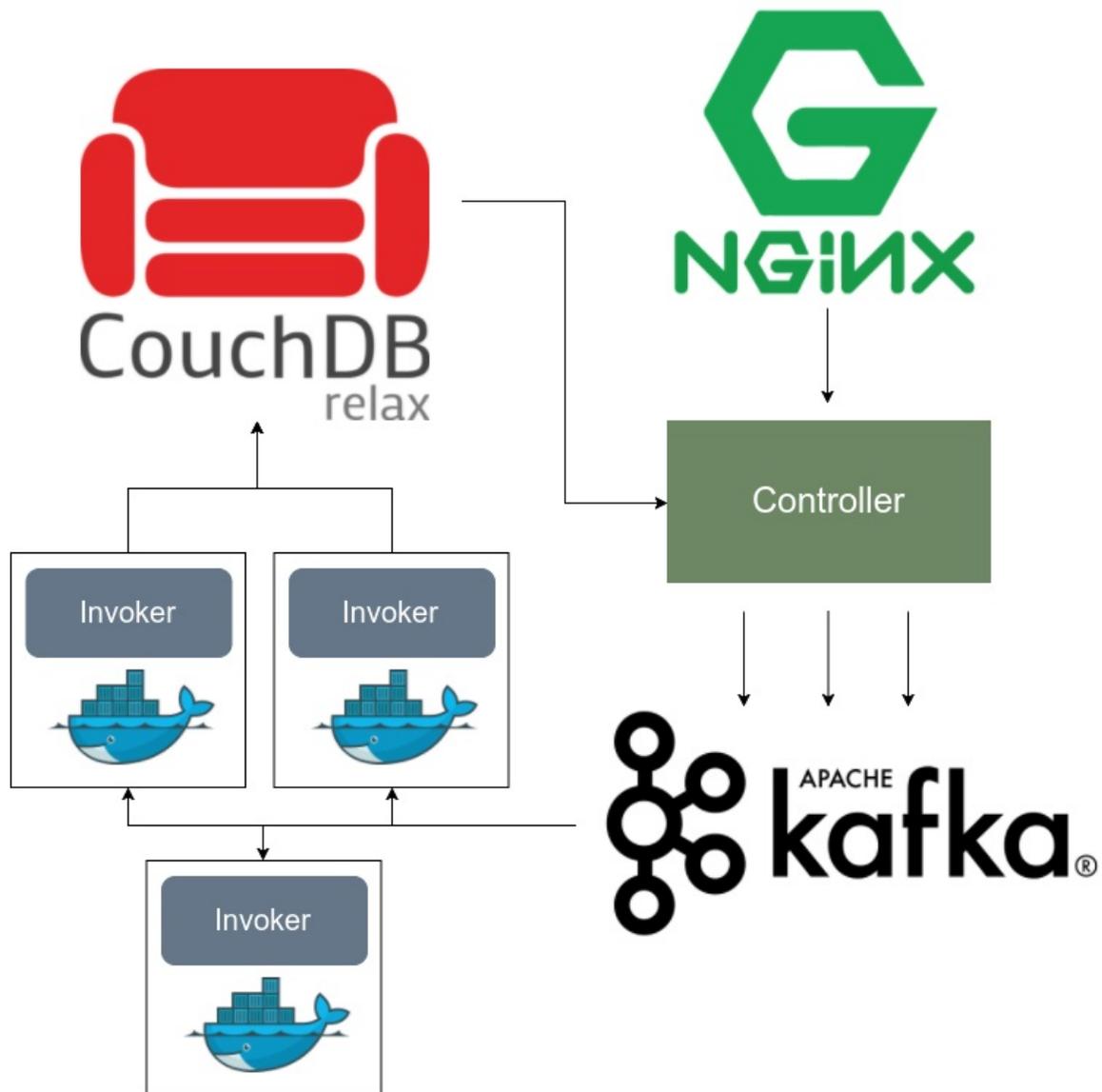


Figure 149: OpenWhisk workFlow

We will review the role of each components in the OpenWhisk workflow.

### 12.3.1.1 The Action and Nginx

As mentioned prior, the action is the response of the OpenWhisk to triggers. Consider the following JavaScript function:

```
function main() {
 return { hello: 'world' };
}
```

This is the Hello World example of the OpenWhisk action where the action returns a JSON object with the key `hello` which has a value of `world`. After saving this function in a `.js` file, e.g. `action.js` then the action could be created using the following command:

```
$ wsk action create HelloAction action.js
```

Then, the `HelloAction` can be invoked using:

```
$ wsk action invoke HelloAction --result
```

The `wsk` command is what is known as OpenWhisk CLI, which we will show how to install in the next sections. Note that OpenWhisk's API is RESTful and fully HTTP based. In other words, the previously-mentioned `wsk action` command is basically a HTTP request equivalent to the following:

```
POST /api/v1/namespaces/$userNamespace/actions/HelloAction
Host: $openwhiskEndpoint
```

The `userNamespace` variable defines the namespace in which the `HelloAction` is put into. Accordingly, `nginx` is the entering point of the OpenWhisk system and it plays an important role as a HTTP server as well as a reverse proxy server, mainly used for SSL termination and HTTP request forwarding.

### 12.3.1.2 Controller: The System's Interface

We learned that `nginx` does not do any processing on the HTTP request except decrypting it (SSL Termination). The main processing of the request starts in the Controller. The controller plays the role of the interface for user both for actions and Create, Read, Update, and Delete (CRUD) requests, translating the user's POST request to action invocation. The controller has an essential role in OpenWhisk workflow and its role is not finished here and is partially involved in next steps as well.

### 12.3.1.3 CouchDB

Naturally some notion of authentication is essentially required for the system. This authentication is performed by the Controller via CouchDB. The CouchDB instance has a specific database, namely `subjects` which contains the credentials and corresponding privileges. The credentials that corresponds to a request are

verified against the `subjects` database and if the user's privileges satisfies the permissions required for the requested `HelloAction`, the action will be invoked. In our example, we are assuming that the `HelloAction` is in a namespace owned by the user, meaning that the user has the required permission to invoke the action.

After authentication and authorization using the `subjects` database, the record for the action `HelloAction` is load from `whisks` database. This record contains the code, the parameters consist of default parameters merged with user parameters, as well as the resource limits, e.g. maximum memory. The `HelloAction` record in `whisk` contains its code (listed previously) and no parameters as the code does not get any parameters.

#### **12.3.1.4 Load Balancer**

Next comes the load balancer which is technically part of the controller and it is load balancer's responsibility to check the health status of the executors, known as `Invokers`, continuously. Load balancer is aware of the available invokers and select them for the actions accordingly.

#### **12.3.1.5 Kafka**

For a request user sends, there are two scenarios where things can go bad:

- Invocation is lost due to a crash
- Invocation has to wait for invokers to be available

Both of this scenarios can be handled with Kafka distributed messaging system. The action invocation mechanism with Kafka is as follows:

The controller "publishes" a message to Kafka. This message contains the required action and corresponding parameters and is addressed to an Invoker chosen by the controller. Kafka responds to the HTTP request of the user with an `ActivationId` which could be used later by the user to get the result. OpenWhisk supports both synchronous and asynchronous invocation models. In the former model, the user's HTTP request is terminated as the system accepts it. The latter model, known as blocking invocation, is otherwise.

### 12.3.1.6 Invoker

As the heart of the OpenWhisk, the Invoker's responsibility is to invoke the action. Invoker is implemented in Scala but it uses Docker for a safe and isolated execution. For each invoked actions, a container is spawned and the code as well as the parameters are passed to it. As soon as the result is obtained, the container is terminated.

The `Action` example is a node.js action and therefore the invoker will start a node.js container, inject our previously-mentioned code to it, runs the code and gets the results, save the logs and terminates the node.js container.

### 12.3.1.7 CouchDB again

The result of the Invoker is saved in another database in CouchDB, namely `activations`, under same `ActivationId` that was sent back to the user. The result of the `HelloAction` example containing the log in JSON format, would look like this:

```
{
 "activationId": "31809ddca6f64cfc9de2937ebd44fbb9",
 "response": {
 "statusCode": 0,
 "result": {
 "hello": "world"
 }
 },
 "end": 1474459415621,
 "logs": [
 "2016-09-21T12:03:35.619234386Z stdout: Hello World"
],
 "start": 1474459415595,
}
```

Similar to the same API call used for submitting the action, we can use OpenWhisk's API to retrieve the result using the `ActivationId`:

```
wsk activation get 31809ddca6f64cfc9de2937ebd44fbb9
```

## 12.3.2 Setting Up OpenWhisk Locally

There are several approaches to starting the OpenWhisk platform:

- Directly running the service
- Running using Kubernetes and Mesos
- Running with Vagrant using a pre-configured VM

But an easier approach is using [OpenWhisk Devtools](#) which is purposed for local development and testing of OpenWhisk. Using OpenWhisk Devtools, you can quickly start OpenWhisk on any machine using `docker compose`. Accordingly, make sure the `docker compose` is already installed on your machine. Then to start the platform, clone the OpenWhisk Devtools and navigate to its folder, then:

```
$ cd docker-compose
$ make quick-start
```

Make sure you do not have any services running on the following ports otherwise the docker compose will fail starting some of the containers:

- 5984 for CouchDB
- 2181, 2888, 3888 for Zookeeper
- 9092 for Kafka
- 8888, 2551 for the Controller
- 8085 for the Invoker
- 9001 for Minio
- 6379 for Redis
- 8080, 443, 9000, 9090 for apigateway
- 8001 Kafka-UI

In case you have services running on any of the previous ports, you can either stop the local services that are using these ports or alternatively you can modify the `docker-compose.yml` and change the source port number in the port number mapping. The latter option is, however, more tricky because you have to make sure the change does not affect the communication between the containers. For instance if you have `Apache` service running on Port 80, then open `docker-compose.yml`, search for the keyword `80:` to find the port mapping with source port of 80: `bash ports: - "80:80"`

then change it to another port:

```
ports:
- "8080:80"
```

After that, you should be able to run `make quick-start` successfully and then you can check the status of the running docker containers using:

```
$ docker ps --format "{{.ID}}: {{.Names}} {{.Image}}"
16e7746c4af1: wsk0_9_prewarm_nodejs6 openwhisk/nodejs6action:latest
dd3c4c2d4947: wsk0_8_prewarm_nodejs6 openwhisk/nodejs6action:latest
6233ae715cf7: openwhisk_apigateway_1 openwhisk/apigateway:latest
3ac0938aecdd: openwhisk_controller_1 openwhisk/controller
e1bb7272a3fa: openwhisk_kafka-topics-ui_1 landoop/kafka-topics-ui:0.9.3
```

```
6b2408474282: openwhisk_kafka-rest_1 confluentinc/cp-kafka-rest:3.3.1
9bab823a891b: openwhisk_invoker_1 openwhisk/invoker
98ebd5b4d605: openwhisk_kafka_1 wurstmeister/kafka:0.11.0.1
65a3b2a7914f: openwhisk_zookeeper_1 zookeeper:3.4
9b817a6d2c40: openwhisk_redis_1 redis:2.8
e733881d0004: openwhisk_db_1 apache/couchdb:2.1
6084aec44f03: openwhisk_minio_1 minio/minio:RELEASE.2018-07-13T00-09-07Z
```

Note that 12 containers should be up and running:

```
$ docker ps --format "{{.ID}}: {{.Names}} {{.Image}} | wc -l"
12
```

### 12.3.2.1 Debugging quick-start

It is always possible that something goes wrong in the process of deploying the 12 dockers. If the `make quick-start` process stuck at some point, the best way to find the issue is to use the `docker ps -a` command to check which of the containers is causing the issue. Then you can try to fix the issue of that container separately. This fix could possibly happen in `docker-compose.yml` file. For instance, there was some issue with the `openwhisk/controller` docker at some point and it turns out the issue was the following line in the `docker-compose.yml`:

```
command: /bin/sh -c "exec /init.sh --id 0 >> /logs/controller-local_logs.log 2>&1"
```

this line is indicating that the following command should be run after the container is started:

```
$ /init.sh --id 0 >> /logs/controller-local_logs.log 2>&1
```

However, starting another instance of the docker image with this command outputted a `Permission Denied` error which could be fixed either by changing the logs folder permission in the docker image or container (followed by a commit) or saving the log file in another folder. In this case replacing that line with the following line would temporarily fix the issue:

```
command: /bin/sh -c "exec /init.sh --id 0 >> /home/owuser/controller-local_logs.log 2>&1"
```

## 12.3.3 Hello World in OpenWhisk

OpenWhisk provides a command line tool called [openwhisk-cli](#) which is used for controlling the platform. As part of the `make quick-start` command that we previously used for starting the platform, the account credentials will automatically be written into the configuration of the CLI. You can either install

the CLI directly from the repository or install it using `linuxbrew`. Alternatively, use the binary available in this path in OpenWhisk Devtools folder:

```
[PATH_TO_DEVTOOLS]/docker-compose/openwhisk-src/bin/wsk
```

Running the `wsk` without any command or flag, will print its help:

```
~/incubator-openwhisk-devtools/docker-compose/openwhisk-src/bin$./wsk
```



```
Usage:
wsk [command]
```

```
Available Commands:
action work with actions
activation work with activations
package work with packages
rule work with rules
trigger work with triggers
sdk work with the sdk
property work with whisk properties
namespace work with namespaces
list list entities in the current namespace
api work with APIs
```

For instance, you can get the host address using:

```
$ wsk property get | grep host
whisk API host 192.168.2.2
```

You can then re-invoke the built-in `Hello World` example using:

```
~/incubator-openwhisk-devtools/docker-compose$ make hello-world
creating the hello.js function ...
invoking the hello-world function ...
adding the function to whisk ...
ok: created action hello
invoking the function ...
invocation result: { "payload": "Hello, World!" }
{ "payload": "Hello, World!" }
creating an API from the hello function ...
ok: updated action hello
invoking: http://192.168.2.2:9090/api/23bc46b1-71f6-4ed5-8c54-816aa4f8c502/hello/world
 "payload": "Hello, World!"
ok: APIs
Action Verb API Name URL
/guest/hello get /hello http://192.168.2.2:9090/api/23bc46b1-71f6-4ed5-8c54-816aa4f8c502/hello/world
deleting the API ...
ok: deleted API /hello
deleting the function ...
ok: deleted action hello
```

## 12.3.4 Creating a custom action

We already invoked the built-in hello world action. Now, we try to build a new custom action. First create a file called `greeter.js`:

```
function main(input) {
 return {payload: 'Hello, ' + input.user.name + ' from ' + input.user.location + '!'};
}
```

Now we can create an action called `greeter` using the `greeter.js`:

```
$ wsk -i action create greeter greeter.js
ok: created action greeter
```

Note that the `-i` option is to prevent the following error:

```
$ wsk action create greeter greeter.js
error: Unable to create action 'summer': Put https://192.168.2.2/api/v1/namespaces/guest/actions/summer?overwrite=false:
Run 'wsk --help' for usage.
```

Afterwards you can get the list of actions to make sure your desired action is created:

```
$ wsk -i action list
actions
/guest/greeter private nodejs:6
```

Afterwards, we can invoke the action by passing a `json` parameter including a name and location and receive the result:

```
$ wsk -i action invoke -r greeter -p user '{"name": "Vafa", "location": "Indiana"}'
{
 "payload": "Hello Vafa from Indiana!"
}
```

Now we can retrieve the list of activation records:

```
$ wsk activation list -i
activations
976a7d02dab7460eaa7d02dab7760e9a greeter
02e0e12118af43b0a0e12118afd3b038 hello
10c2cddb0d2c4c1f82cddb0d2c1c1feb hello
```

The result of the previous command is showing that the `hello` action has been invoked twice and the `greeter` action was invoked once. You can get more information about each of the activation using the `wsk -i activation get [ACTIVATION_ID]`:

```
$ wsk -i activation get 976a7d02dab7460eaa7d02dab7760e9a
ok: got activation 976a7d02dab7460eaa7d02dab7760e9a
{
 "namespace": "guest",
 "name": "greeter",
 "version": "0.0.1",
 "subject": "guest",
 "activationId": "976a7d02dab7460eaa7d02dab7760e9a",
 "start": 1539980284774,
 "end": 1539980284886,
 "duration": 112,
 "response": {
 "status": "success",
 "statusCode": 0,
 "success": true,
 "result": {
```

```
 "payload": "Hello Vafa from Indiana!"
 },
 ...
```

Finally, after you are finished using the OpenWhisk Devtools, you can stop platform using:

```
~/incubator-openwhisk-devtools/docker-compose$ make destroy
Stopping openwhisk_apigateway_1 ... done
Stopping openwhisk_controller_1 ... done
Stopping openwhisk_kafka-topics-ui_1 ... done
Stopping openwhisk_kafka-rest_1 ... done
Stopping openwhisk_invoker_1 ... done
Stopping openwhisk_kafka_1 ... done
Stopping openwhisk_zookeeper_1 ... done
Stopping openwhisk_redis_1 ... done
Stopping openwhisk_db_1 ... done
Stopping openwhisk_minio_1 ... done
...
```

## 12.4 KUBELESS

---

 add bibtex

### 12.4.1 Introduction

Kubeless is an Serverless or FaaS framework that has been developed as a Kubernetes native framework. Kubeless is designed using services and features that are provided natively on the Kubernetes framework. It uses Kubernetes Custom Resource Definition to define functions

### 12.4.2 Programing model

Similar to other serverless frameworks, the programming model of Kubeless is an event-driven model. The two main components that need to be understood are functions and events. Kubeless currently supports 3 types of functions runtimes Python, NodeJS and Ruby. These runtimes can be used to create and deploy functions. For each function an event type is defined which specifies the type of trigger for the event. Kubeless currently supports 3 types of triggers which are, HTTP based, scheduled and event-based (pubsub).

### 12.4.3 System Architecture

The system architecture is based completely on Kubernetes primitives which

were discussed to some extent in the previous section. The Kubless architecture has 3 main components, Functions API, Kubeless-controller, and Kafka. Additionally, they provide Kubeless command-line client which can be used to perform CRUD operations for function more easily.

The Functions API provides a REST Endpoint to create, read, update and delete functions. This is developed as a Kubernetes Custom Resource Definitions (CRD). CRD is an extension point provided by Kubernetes that can be used to create custom resources. A custom resource exposes a REST endpoint and makes it available as any other REST API that is embedded with Kubernetes. Once created, Functions custom resource exposes the REST API that can be used for function CRUD operations. The Kubeless-controller is a custom controller that is deployed with the Kubernetes installation. This controller continuously monitors invocations that occur at the functions REST API and performs the required tasks according to the invocation. For example, if the invocation is for a function creation, the controller will create a Deployment for the function and a Service to expose the function. The Deployment will contain information on what runtime the function is intended to use, therefore the deployment will make sure to spin up Pods which will host containers of that runtime when a function execution is requested. Kafka is deployed within the Kubernetes installation as an event source which can be used to trigger the functions.

Because the container image that is used to execute the function is generic, it does not have any specific dependencies that are required by the function and the function code itself. These two need to be injected into the Pod when the Pod is created. For the application logic/function code, Kubless uses a configuration resource provided by Kubernetes API named ConfigMap. The code segment is attached to the ConfigMap which can be read from within a Pod once the Pod is created. In order to install all the required dependencies, another Kubernetes resource named Init containers are utilized. Init containers are a special kind of container which can be configured to run when a Pod is created. Kubernetes also guarantees that all init containers specified for a Pod will run till completion before the application containers ( in this case function container) are executed. Kubless runs an init container which will install all the required dependencies for the function before invoking the function. The function dependencies must be specified at function creation time.

## 12.5 MICROSOFT AZURE FUNCTION

---



fa18-516-08



TODO students can contribute this section

## 12.6 GOOGLE CLOUD FUNCTIONS

---

---



### Learning Objectives

- Introduction to Google Cloud Function
  - Practical example using console mode
- 

Google Cloud Function is Google's offering for Function as a Service. It enables serverless computing and offers functions as services on Google Cloud Platform driven by trigger events from Cloud Pub/Sub, Cloud Storage, HTTP or changes in log in Stackdriver logging. Cloud functions can also be invoked for real time mobile changes. Google Cloud page on cloud functions

- <https://cloud.google.com/functions/use-cases/>

gives more detail about the use cases such as Serverless application backends, Real-time data processing, Intelligent applications - all without the need of provisioning a server instance or the overhead of managing a server instance. The functions are invoked as services whenever needed for a business requirement and the cost is billed as per the minutes of usage just for the function execution time. The Google Cloud Functions can be written in Node.js or Python. For Python runtime environment refer to the page

- <https://cloud.google.com/functions/docs/concepts/python-runtime>

For Node.js 6 runtime environment refer to the page

- <https://cloud.google.com/functions/docs/concepts/nodejs-6-runtime>

For Node.js 8 runtime environment refer to the page

- <https://cloud.google.com/functions/docs/concepts/nodejs-8-runtime>>

### **12.6.1 Google Cloud Function Example**

Following from the example as presented in AWS Lambda section, we will look into a simple example of building a Google Cloud Function to check if a string is Palindrome or not. The implementation will be in Python and we will use Python runtime environment. We will use HTTP trigger to invoke the function using HTTP request. We will also use the Google Cloud Console to build, deploy and test the function. Finally we will use HTTP url to send request to the function to get the result of our query.

Let us begin:

**Step 1:** Login to Google Cloud Platform with your GCP account. We are using free tier for this demonstration. Refer to the section for Google Cloud in the epub for creting a free tier GCP account.

**Step 2:** Select or create a Project and go to dashboard (see [Figure 150](#))

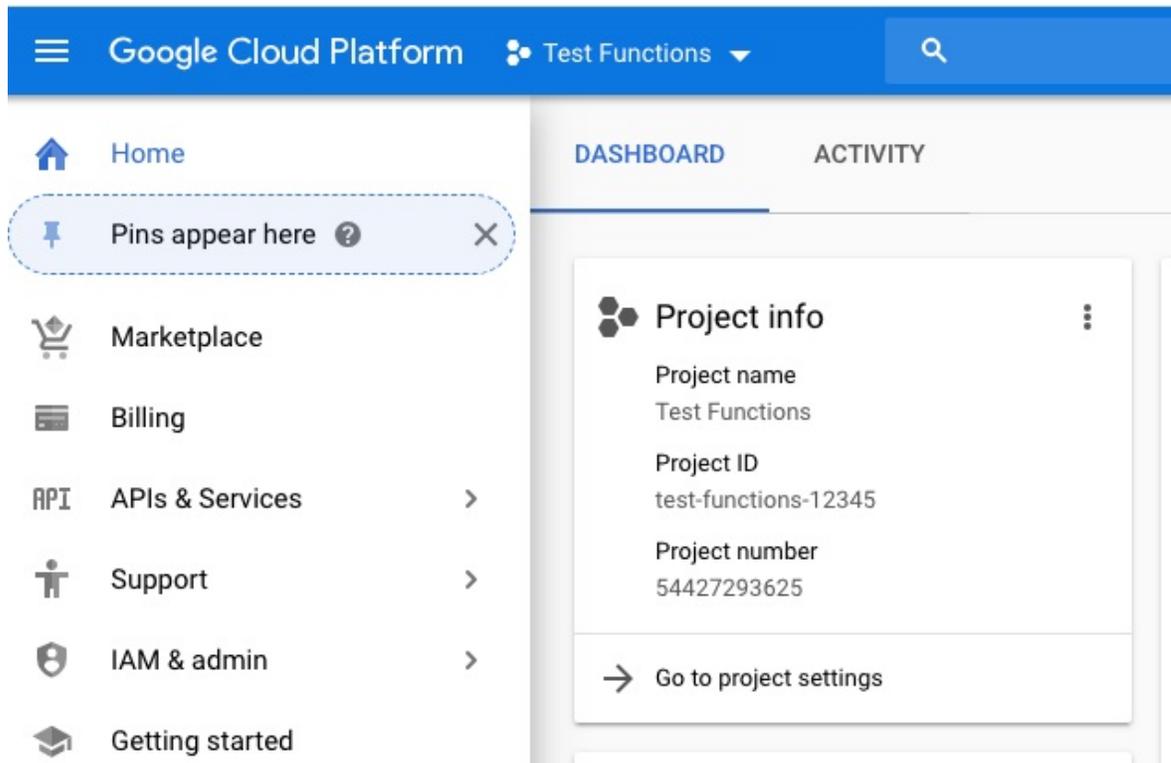


Figure 150: Login to Project and Dashboard

**Step 3:** Click “Create a Cloud Function” (see [Figure 151](#))

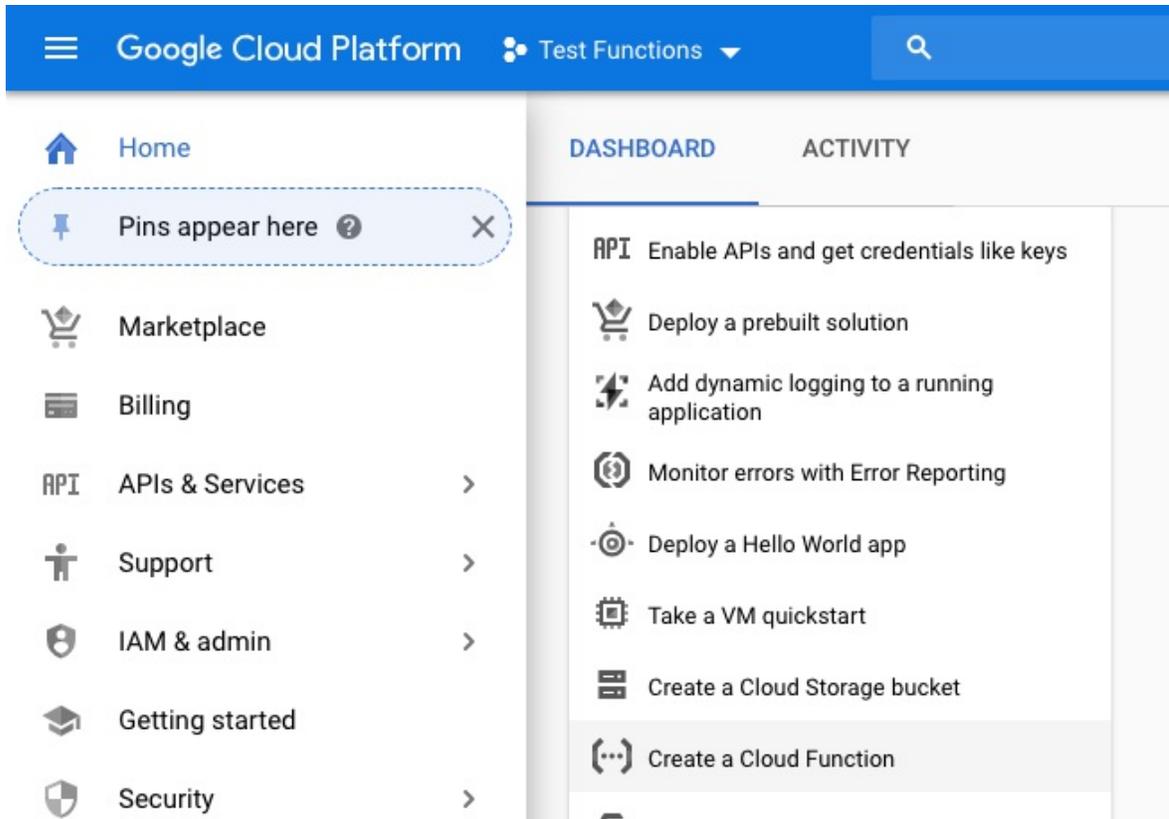


Figure 151: Create a Function

**Step 4:** Enable cloud function API if it is not enabled:(see [Figure 152](#))

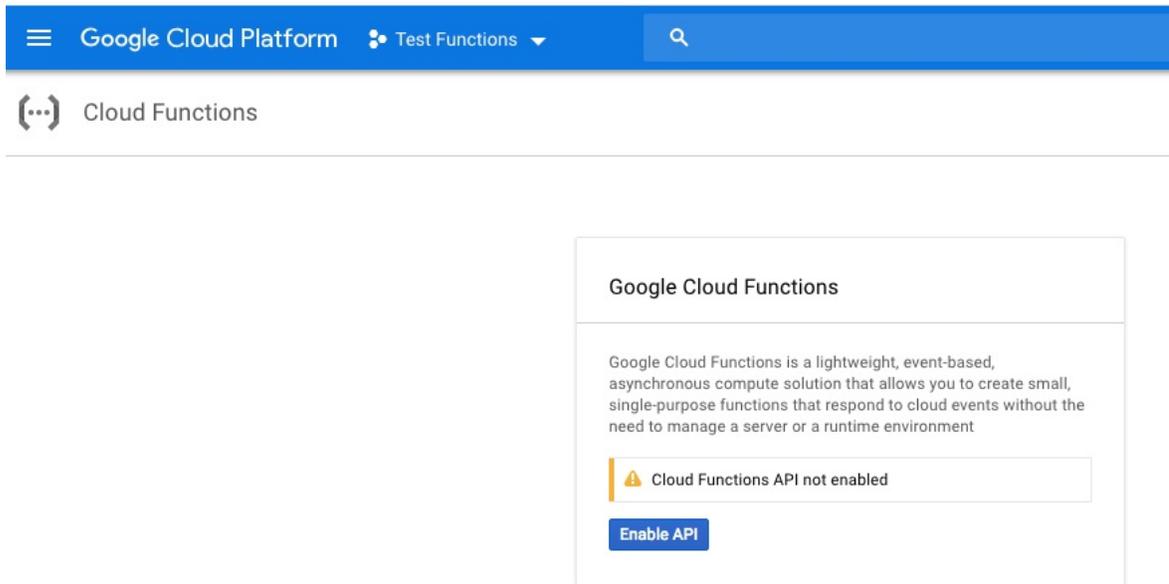


Figure 152: Enable the API

**Step 5:** Click Create Function (see [Figure 153](#))

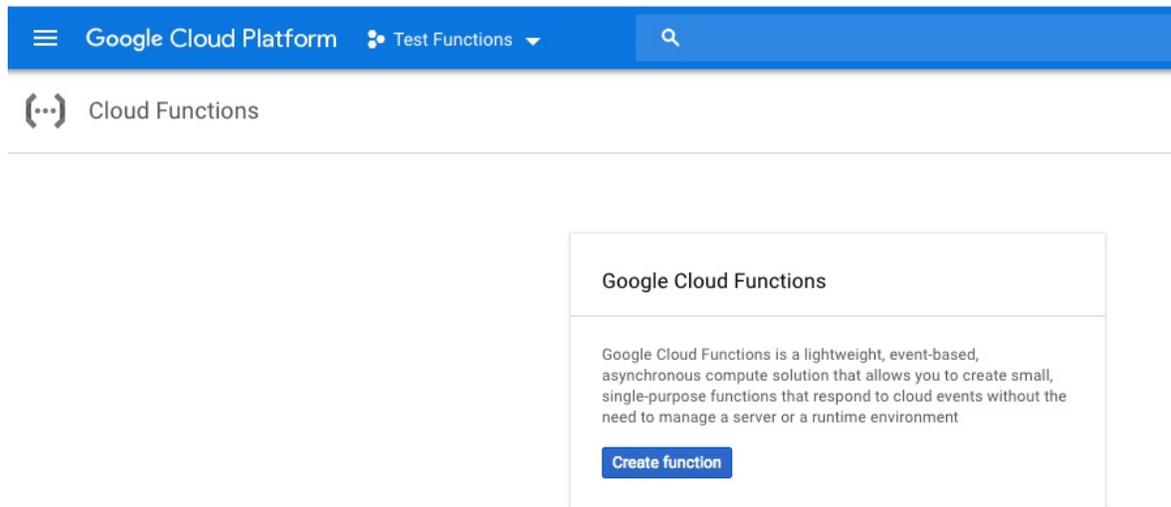


Figure 153: Select Create Function

**Step 6:** In the next page, give a name to the function. In our case we are giving function name as isPalindrome. Specify the memory (128 mb is good for this demo). Select the function trigger as HTTP. Choose inline editor for the source code and finally Python 3.7 as the run time environment.(see [Figure 154](#))

Google Cloud Platform Test Functions

Cloud Functions Create function

Name ?  
isPalindrome

Memory allocated  
128 MB

Trigger  
HTTP

URL  
https://us-central1-test-functions-12345.cloudfunctions.net/isPalindrome

Source code

- Inline editor
- ZIP upload
- ZIP from Cloud Storage
- Cloud Source repository

Runtime  
Python 3.7 (Beta)

Figure 154: Name Function

**Step 7:** In the inline source editor, write a Python function and then click Create. We have written a Python function to check for Palindrome string. NOTE: This is not an optimized Python code, it is just used here for demonstration purpose. This function can be optimized further with Python standards style writing.(see [Figure 155](#))

Google Cloud Platform Test Functions

Cloud Functions Create function

Runtime  
Python 3.7 (Beta)

main.py requirements.txt

```
1 def isPalindrome(request):
2 """Responds to any HTTP request.
3 Args:
4 request (flask.Request): HTTP request object.
5 Returns:
6 The response text or any set of values that can be
7 Response object using
8 `make_response` <http://flask.pocoo.org/docs/0.12/ag
9 """
10 request_json = request.get_json()
11 if request.args and 'message' in request.args:
12 string=request.args.get('message')
13 if(string==string[::-1]):
14 return f'The string is a palindrome'
15 else:
16 return f'The string isn not a palindrome'
17 elif request_json and 'message' in request_json:
18 string=request_json['message']
19 if(string==string[::-1]):
20 return f'The string is a palindrome'
21 else:
22 return f'The string isn not a palindrome'
23 else:
24 return f'Not Valid Request'
25
```

Function to execute ?

Figure 155: :Write Python Function

**Step 8:** The function is created and deployed in the next page (see [Figure 156](#)) and [Figure 157](#))

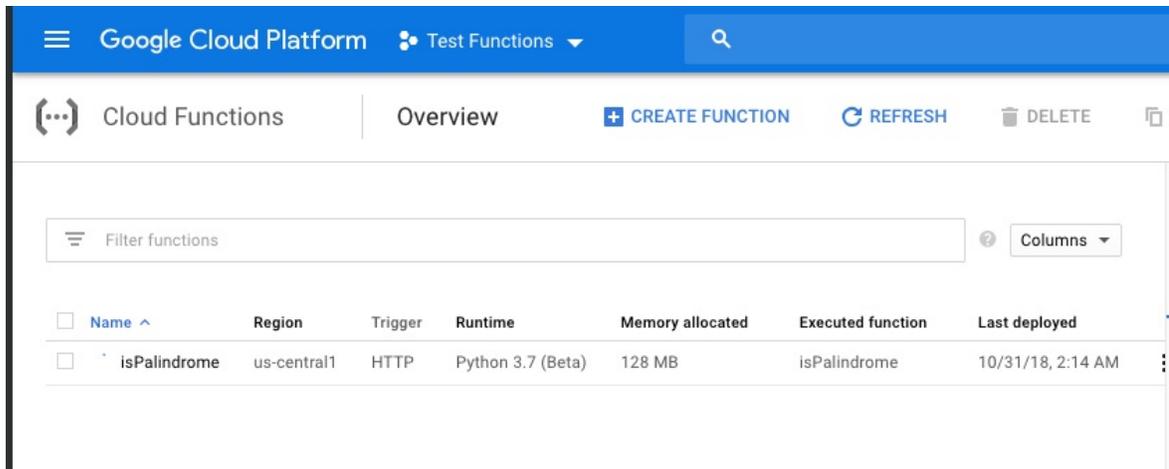


Figure 156: Function is Deployed

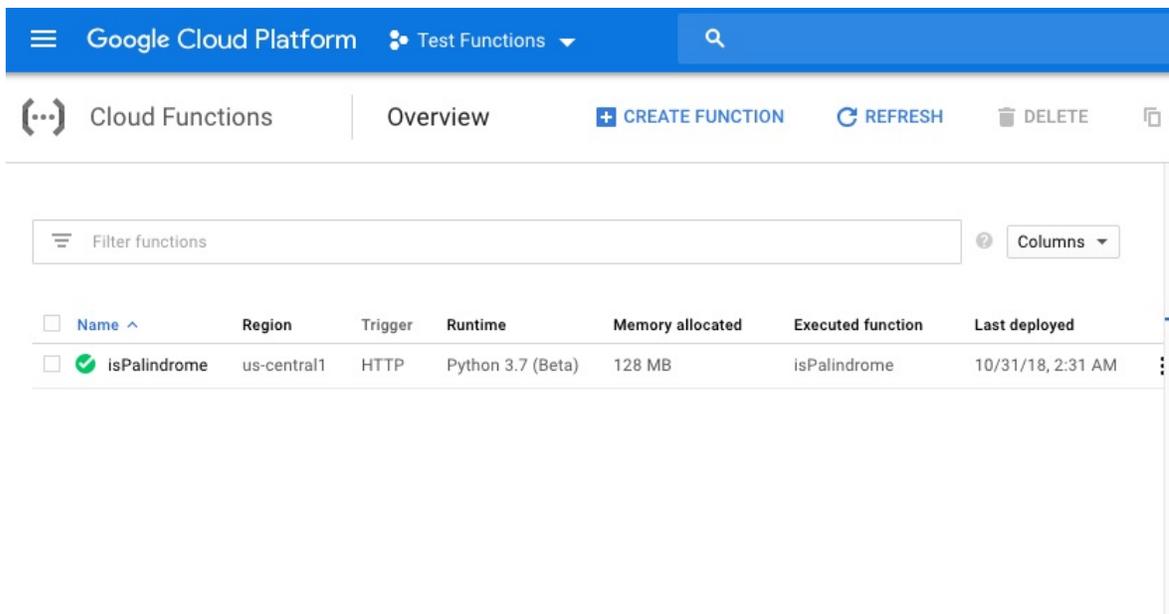


Figure 157: Function is Deployed

**Step 9:** Finally we will test the function (see [Figure 158](#))

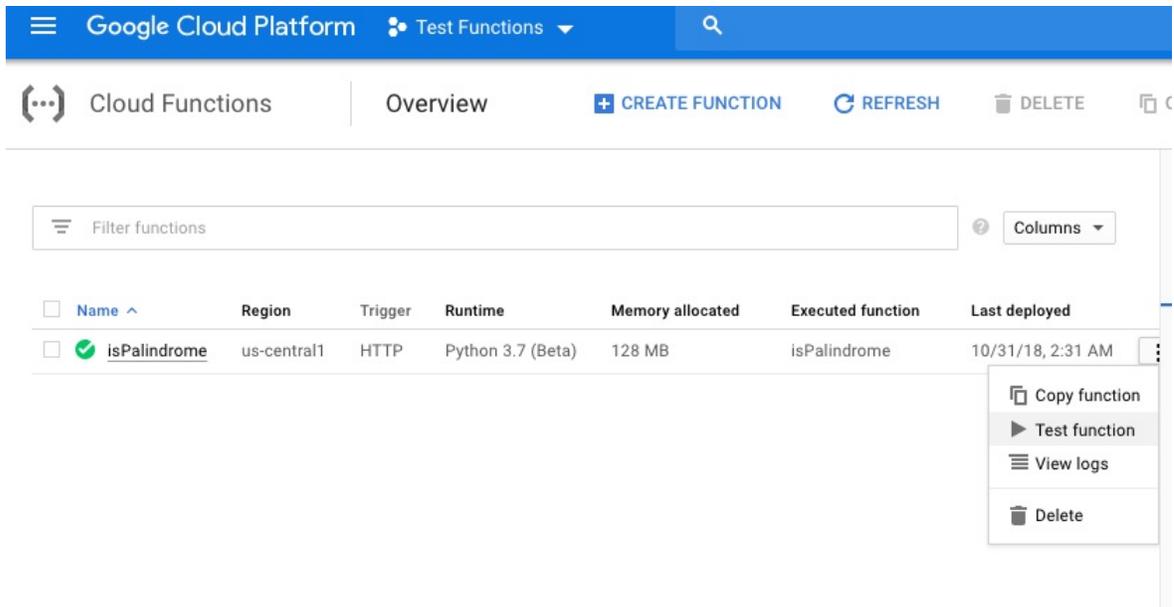


Figure 158: Test The Function

**Step 10:** In the Trigger event box, write a HTTP message request in JSON format and click Test the Function (see [Figure 159](#))

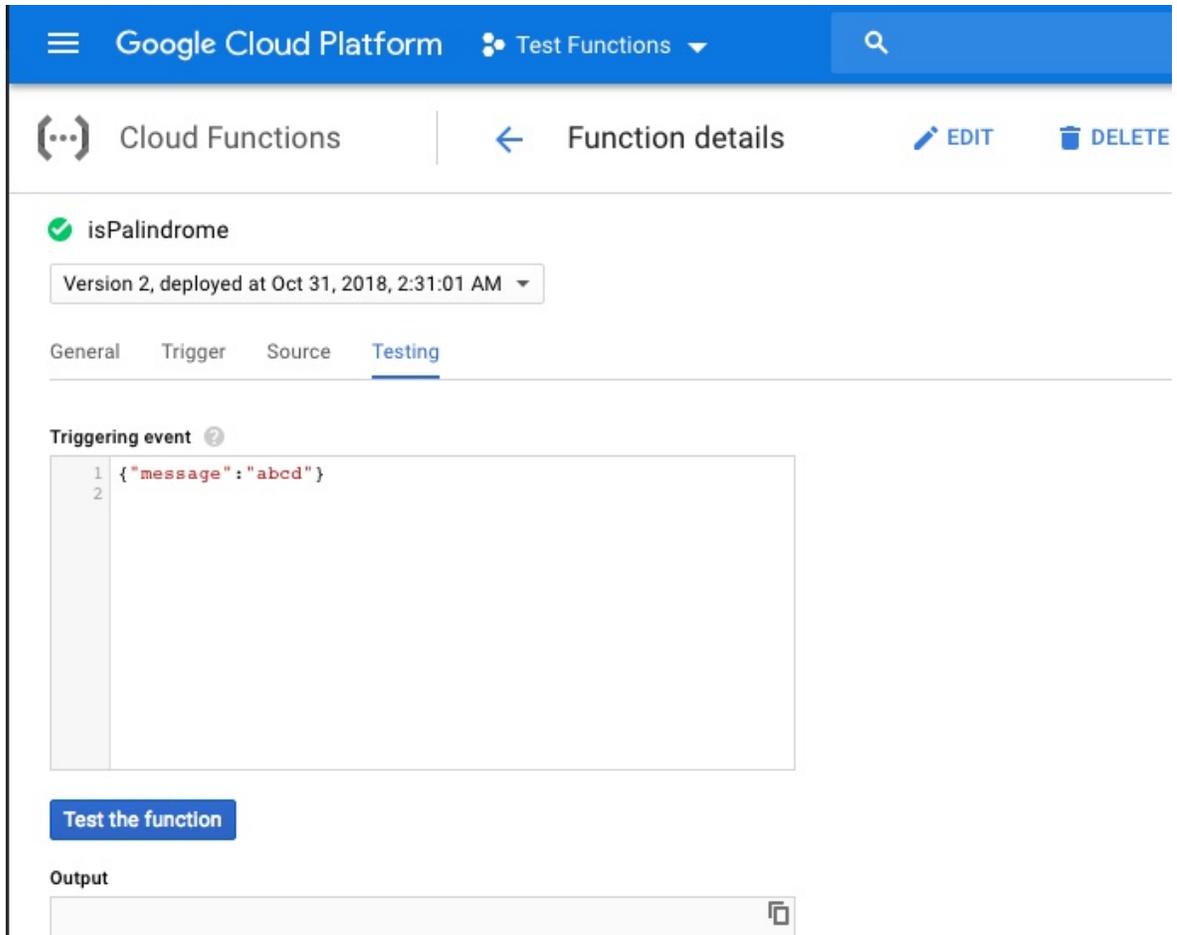


Figure 159: Trigger Event

**Step 11:** The response box will show the result of the test as expected (see [Figure 160](#))

The screenshot shows the Google Cloud Platform interface for testing a function. At the top, there is a blue header with the Google Cloud Platform logo, 'Test Functions' dropdown, and a search icon. Below the header, the breadcrumb navigation shows 'Cloud Functions' and 'Function details'. The function name 'isPalindrome' is displayed with a green checkmark icon. A dropdown menu shows 'Version 2, deployed at Oct 31, 2018, 2:31:01 AM'. There are four tabs: 'General', 'Trigger', 'Source', and 'Testing', with 'Testing' being the active tab. Under the 'Triggering event' section, a code editor shows a JSON object: 

```
1 { "message": "abcd" }
2
```

. Below the code editor is a blue button labeled 'Test the function'. Under the 'Output' section, a text box displays the result: 'The string isn not a palindrome'. A copy icon is visible in the top right corner of the output box.

Figure 160: Result

**Step 12:** Let us run one more Test (see [Figure 161](#))

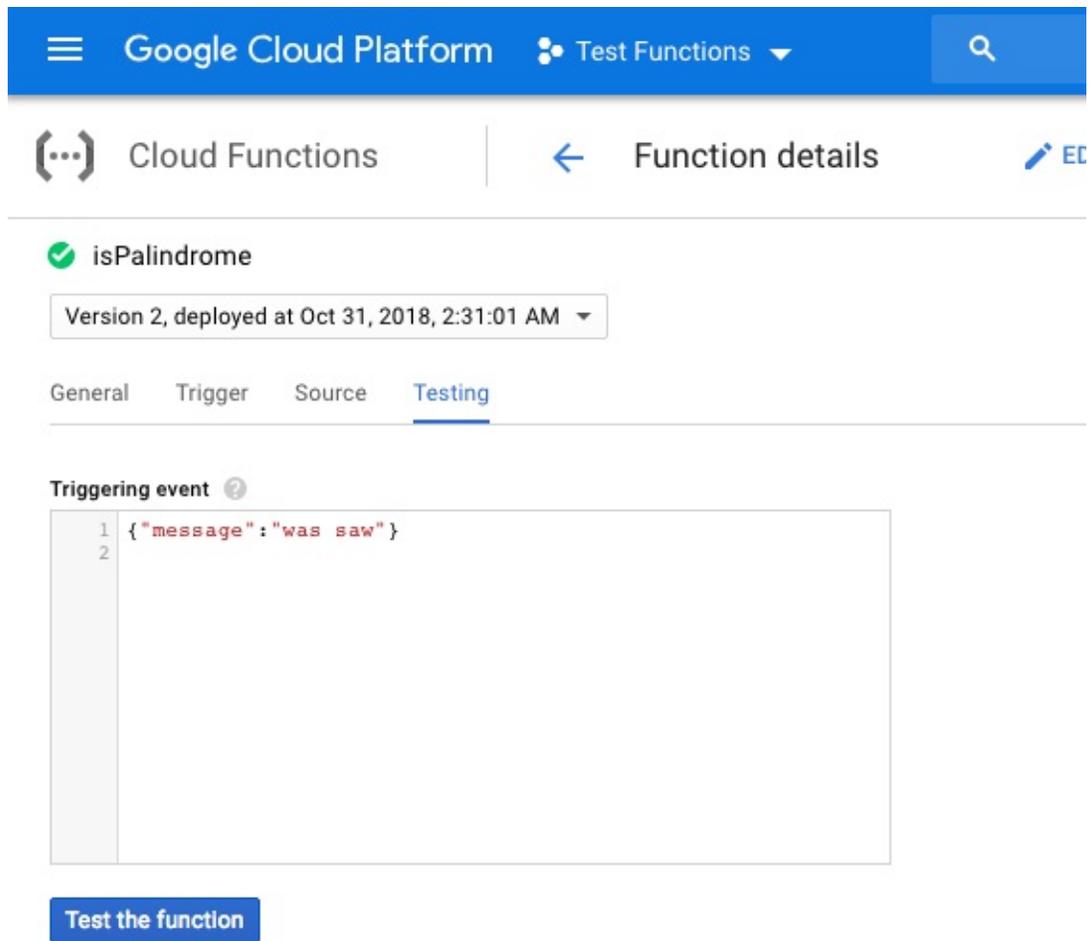


Figure 161: Another Test

**Step 13:** You will get the expected result (see [Figure 162](#))

Triggering event ?

```
1 {"message": "was saw"}
2
```

[Test the function](#)

Output

```
The string is a palindrome
```

Figure 162: Expected Result For Test

**Step 14:** Let us test our function deployment using url. Click on the function name (see [Figure 163](#))

Google Cloud Platform Test Functions

Cloud Functions Overview [+ CREATE FUNCTION](#) [REFRESH](#)

Filter functions

<input type="checkbox"/>	Name ^	Region	Trigger	Runtime	Memory allocated	Executed function	
<input type="checkbox"/>	<input checked="" type="checkbox"/> <a href="#">isPalindrome</a>	us-central1	HTTP	Python 3.7 (Beta)	128 MB	isPalindrome	1

Figure 163: Deployment Url

**Step 15:** In the next page, click on Trigger page and copy the url (see [Figure 164](#))

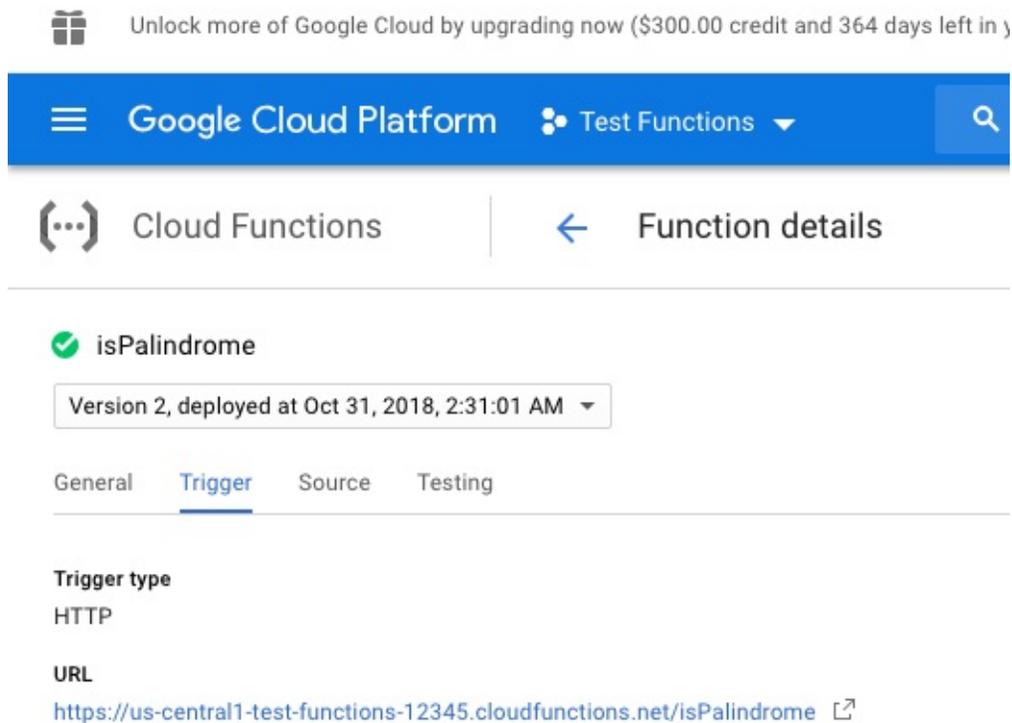


Figure 164: Trigger Url

**Step 16:** In a web browser type the url and add the HTTP request to it and hit enter

- <https://us-central1-test-functions-12345.cloudfunctions.net/isPalindrome?message=abcd>

**Step 17:** You will get a response back from the function(see [Figure 165](#))

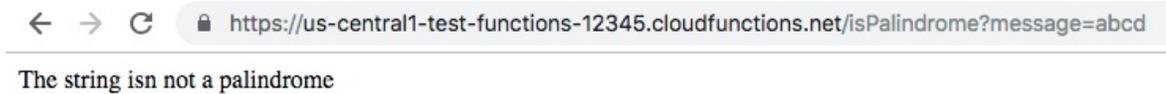


Figure 165: Test Http

**Step 18:** Another test (see [Figure 166](#))



Figure 166: Another Http Test

This completes our demo for Google Cloud Function offered as Function as Service. To learn more about Google Cloud Functions and trigger options available alongwith triggers using command line - visit

<https://cloud.google.com/functions/>

To learn about creating and deploying functions using command line instead of GCP console - visit

- <https://cloud.google.com/functions/docs/quickstart>

## 12.7 OPENFAAS

---

OpenFaas is a framework for building serverless functions on docker containers and follows the same workflow as micro services. Since, OpenFaas uses Docker and Kubernetes technologies, it will give lot of hosting options ranging from a laptop to large-scale cloud systems Any program written in any language can be packaged as a function within in a container which gives a best approach to

convert all the old code to run on cloud-based infrastructure

Few benefits of OpenFaas

- Easy to Use
- Deployable to private or public clouds in container
- Simplicity in architecture and design
- Open and extensible platform
- Language agnostic

### 12.7.1 OpenFaas Components and Architecture

There are three components which include API Gateway, Function Watchdog and the instance of Prometheus. All the functions run on Docker containers orchestrated by either Docker Swarm or Kubernetes. The function watchdog is part of the function containers, whereas the API Gateway and Prometheus instance are services.

## Functions as a Service

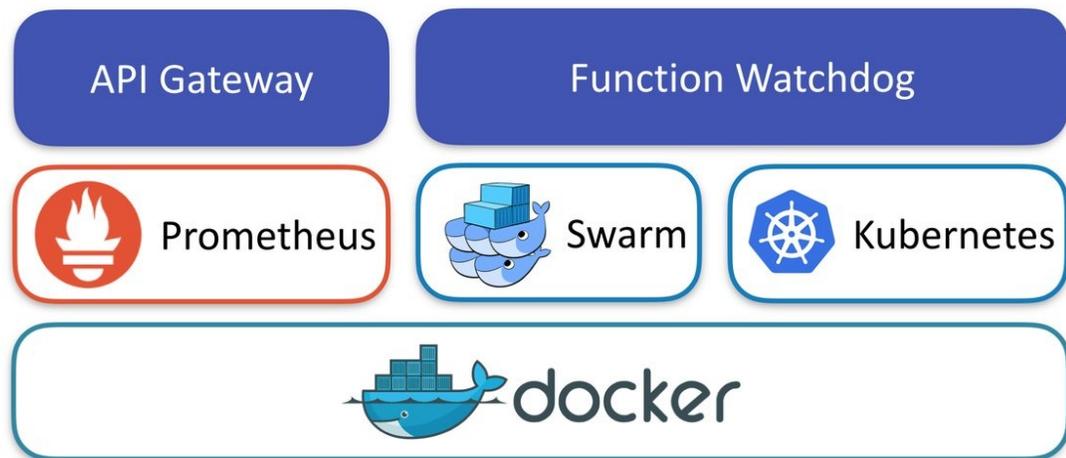


Figure 167: faas - OpenFaas - Arch [91]

#### 12.7.1.1 API Gateway

Routes inbound requests to the functions and collects metrics through

Prometheus. It autoscales modifying service replicas counts. Offers a convenient UI and endpoints for the CLI

### **12.7.1.2 Function Watchdog**

It is a tiny HTTP server, enclosed along with the app in the docker image. It receives request from the API Gateway, triggers the app. It provide args and catch result through STDIN/STDOUT

### **12.7.1.3 OpenFaas CLI**

The OpenFaas CLI provides mechanism to deploy the functions in the containders

### **12.7.1.4 Monitoring**

OpenFaas makes monitoring simple with the use of Prometheus. The end users can install Grafana Dashboard and connect point to the Promotehus data source. This provides quick access to the dashboard to monitor the OpenFaas functions

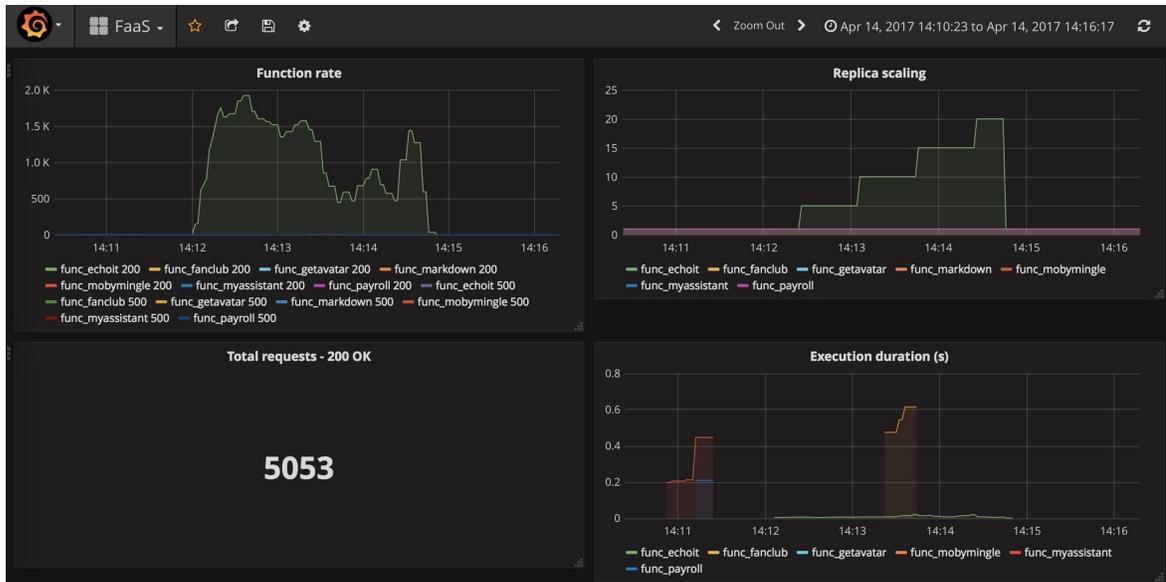


Figure 168: faas - OpenFaas - Grafana [91]

## 12.7.2 OpenFaas in Action

### 12.7.2.1 Prerequisites

1. Docker
2. Git Bash (for Windows)

### 12.7.2.2 Single Node Cluster

```
$ docker swarm init
```

Using a Terminal on Mac or Linux:

```
$ curl -sL cli.openfaas.com | sudo sh
```

For windows faas-cli.exe need to be downloaded from this link <https://github.com/openfaas/faas-cli/releases>

### 12.7.2.3 Deploy OpenFaas

OpenFaas gives the option to use yaml(.yml) file for configuring the functions and the image will be built by OpenFaas automatically. Alternatively, custom docker image can be built and passed as an argument to the OpenFaas CLI. This

gives the flexibility for the developers to extend further which is not in the standard yaml file.

```
$ git clone https://github.com/openfaas/faas
$ cd faas
$ git checkout master
$./deploy_stack.sh --no-auth
$ cd <test function folder>
$ docker build -t <test function image>
$ faas-cli deploy --image <test function image> --name <test function name>
```

### 12.7.2.4 To Run OpenFaaS

OpenFaaS can be tested via curl, faas-cli, or any HTTP-based client to connect to the API gateway to invoke a function

Once the function is deployed, the functions can be verified in the following url

- <http://127.0.0.1:8080>

The screenshot shows the OpenFaaS Portal interface. On the left, there is a sidebar with the 'OpenFaaS Portal' logo and a 'Deploy New Function' button. Below this is a search bar and a list of functions: 'func\_markdown', 'func\_echoit', 'func\_wordcount', 'func\_hubstats', and 'func\_base64'. The 'func\_markdown' function is selected and highlighted. The main content area displays details for 'func\_markdown', including 'Replicas: 1' and 'Invocation count: 11'. Below this, there is an 'Invoke function' section with an 'INVOKE' button and radio buttons for 'Text', 'JSON', and 'Download'. The 'Text' option is selected. The 'Request body' field contains the text '## The \*\*OpenFaaS\*\* \_workshop\_'. Below the 'Invoke function' section, there is a table showing the 'Response status' as '200' and the 'Round-trip (s)' as '0.038'. The 'Response body' is displayed as HTML: '<h2>The <strong>OpenFaaS</strong> <em>workshop</em></h2>'. The 'Image' field shows the full image name: 'functions/markdown-render:latest@sha256:fbac7549ad7a598385c40df9e16d25c51a8786865'.

Figure 169: faas-OpenFaaS-Portal [91]

### 12.7.3 OpenFaaS Function with Python

This section illustrates how to create a simple Python function with OpenFaaS.

Following are the the steps involved in creating and deploying a function with OpenFaaS

- Install OpenFaaS
- Install the OpenFaaS CLI
- Build the function
- Deploy the function

Installing OpenFaaS:

OpenFaaS installation guide can be viewed on this web page:

- <https://docs.openfaas.com/deployment>

Installing CLI:

For Linux, type the following

```
$ curl -sSL https://cli.openfaas.com | sudo sh
```

For Mac, type the following

```
$ brew install faas-cli
```

Developing a Python function:

First, scaffold a new Python function using the CLI

```
$ faas-cli new --lang python func-python
```

Following 3 files will be created in the current directory

```
func-python/handler.py
func-python/requirements.txt
func-python.yml
```

Edit the handler.py

```
def handle(req):
 print("Python Function: " + req)
```

Functions need to be specified in a YAML file created to indicate what to build and deploy onto the OpenFaaS cluster. YAML file should be created as follows

```
provider:
```

```
name: faas
gateway: http://127.0.0.1:8080

functions:
 func-python:
 lang: python
 handler: ./func-python
 image: func-python
```

YAML file description is as follows

- *gateway*- Location to specify a remote gateway, the programming language, and location of the handler within the filesystem.
- *functions* - This block defines the functions in our stack.
- *lang* - Programming language used.
- *handler* - This is the folder / path fo the handler.py file and any other source code
- *image* - This is the Docker image name. If it is being pushed to the Docker Hub, prefix should include Docker Hub accountn

Build the function:

```
$ faas-cli build -f ./func-python.yml
...
```

```
Successfully tagged func-python:latest
Image: func-python built.
```

Docker engine builds the function into an image in the docker library and images will appear as follows

```
$ docker images | grep func-python
func-python latest <image ID> one minute ago
```

Deploy the function:

```
$ faas-cli deploy -f ./func-python.yml
Deploying: func-python.
No existing service to remove
Deployed.
200 OK
URL: http://127.0.0.1:8080/function/func-python
```

Function can be tested either through the OpenFaas portal UI or with curl command

```
$ curl 127.0.0.1:8080/function/func-python -d "Test Successfull"
Python Function: Test Successfull
```

faas-cli commands can also be used to list and invoke the functions

```
faas-cli list
```

In case third party dependencies are required, they can be specified in a requirements.txt file along with the function handler and the function can be deployed.

## 12.8 OPENLAMBDA

---

Cloud computing is evolving. All major public cloud providers now support serverless computing such as AWS Lambda, Google Cloud Functions (Alpha) and Azure Function. Serverless computing introduces many new research challenges in the areas of sandboxing, session management, load balancing, and databases. To facilitate work in these areas, OpenLambda is building an open-source serverless computing platform.

Serverless Computation with OpenLambda PDF slide material is available at:

- <https://open-lambda.org/resources/slides/ol-first-meeting.pdf>.

Communication is available at the following Slack Development Channel (You will need to create an account if you do not already have one):

- <https://open-lambda.slack.com/>.

### 12.8.1 Suggested Materials

- Wat: <https://www.destroyallsoftware.com/talks/wat>.
- History of Containers: <https://www.youtube.com/watch?v=hgN8pCMLI2U>.
- AFS benchmarking: <http://www.cs.cmu.edu/~coda/docdir/s11.pdf>.

### 12.8.2 Development

OpenLambda source code is available on github (all material below have been sourced from github): <https://github.com/open-lambda/open-lambda>.

### 12.8.3 OpenLambda

OpenLambda is an Apache-licensed serverless computing project, written in Go

and based on Linux containers. The primary goal of OpenLambda is to enable exploration of new approaches to serverless computing. Our research agenda is described in more detail in a [HotCloud '16 paper][https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16\\_he](https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_he)

All detail about getting started, installation, configuration, administration and licensing has been sourced from <https://github.com/open-lambda/open-lambda>

## 12.8.4 Getting Started

OpenLambda relies heavily on operations that require root privilege. To simplify this, we suggest that you run all commands as the root user (i.e., run `sudo -s` before building or running OpenLambda). Additionally, OpenLambda is only actively tested on Ubuntu 14.04 & 16.04.

### 12.8.4.1 Install Dependencies

First, run the dependency script to install necessary packages (e.g., Golang, Docker, etc.)

```
$./quickstart/deps.sh
```

Now, build the OpenLambda worker & its dependencies and run integration tests. These tests will spin up clusters in various configurations and invoke a few lambdas.

```
$ make test-all
```

If these pass, congratulations! You now have a working OpenLambda installation.

### 12.8.4.2 Start a Test Cluster

To manage a cluster, we will use the `admin` tool. This tool manages state via a `cluster` directory on the local file system. More details on this tool can be found [below](#).

First, we need to create a cluster. Ensure that the path to the `cluster` directory exists, and it does not.

```
$./bin/admin new -cluster my-cluster
```

Now start a local worker process in your cluster:

```
$./bin/admin workers -cluster=my-cluster
```

Confirm that the worker is listening and responding to requests:

```
$./bin/admin status -cluster=my-cluster
```

This should return something similar to the following:

```
Worker Pings:
http://localhost:8080/status => ready [200 OK]

Cluster containers:
```

The default configuration uses a local directory to store handler code, so creating new lambda functions is as simple as writing files to the `./my-cluster/registry` directory.

Copy an example handler (`hello`) to this directory:

```
$ cp -r ./quickstart/handlers/hello ./my-cluster/registry/hello
```

Now send a request for the `hello` lambda to the worker via `curl`. Handlers are passed a Python dictionary corresponding to the JSON body of the request. `hello` will echo the `"name"` field of the payload to show this.

```
$ curl -X POST localhost:8080/runLambda/hello -d '{"name": "Alice"}'
```

The request should return the following:

```
"Hello, Alice!"
```

To create your own lambda named `<NAME>`, write your code in `./my-cluster/registry/<NAME>/lambda_func.py`, then invoke it via `curl`:

```
$ curl -X POST localhost:8080/runLambda/<NAME> -d '<JSON_STRING>'
```

Now, kill the worker process and (optionally) remove the `cluster` directory.

```
$./bin/admin kill -cluster=my-cluster
$ rm -r my-cluster
```

## 12.8.5 Administration

The `admin` tool is used to manage OpenLambda clusters. This tool manages state

via a `cluster` directory on the local file system. Note that only a single OpenLambda worker per machine is currently supported.

The simplest admin command, `worker-exec`, allows you to launch a foreground OpenLambda process. For example:

```
$ admin worker-exec --config=worker.json
```

The above command starts running a single worker with a configuration specified in the `worker.json` file (described in detail later). All log output goes to the terminal (i.e., `stdout`), and you can stop the process with `ctrl-C`.

Suppose `worker.json` contains the following line:

```
"worker_port": "8080"
```

While the process is running, you may ping it from another terminal with the following command:

```
$ curl http://localhost:8080/status
```

If the worker is ready, the status request will return a “ready” message.

Of course, you will typically want to run one (or maybe more) workers as servers in the background on your machine. Most of the remaining admin commands allow you to manage these long-running workers.

An OpenLambda worker requires a local file-system location to store handler code, logs, and various other data. Thus, when starting a new local cluster, the first step is to indicate where the cluster data should reside with the `new` command:

```
$ admin new --cluster=<ROOT>
```

For OpenLambda, a local cluster’s name is the same as the file location. Thus, `<ROOT>` should refer to a local directory that will be created for all OpenLambda files. The layout of these files in the `<ROOT>` directory is described in detail below. You will need to pass the cluster name/location to all future admin commands that manage the cluster.

The `<ROOT>/config/template.json` file in the cluster located at `<ROOT>` will contain many configuration options specified as keys/values in JSON. These setting will be

used for every new OpenLambda worker. You can modify these values by specifying override values (again in JSON) using the `setconf` command. For example:

```
$./admin setconf --cluster=<ROOT> '{"sandbox": "sock", "registry": "local"}'
```

In the above example, the configuration is modified so that workers will use the local registry and the “sock” sandboxing engine.

Once configuration is complete, you can launch a specified number of workers (currently only one is supported?) using the following command:

```
$./admin workers --cluster=<NAME> --num-workers=<NUM> --port=<PORT>
```

This will create a specified number of workers listening on ports starting at the given value. For example, suppose `<NUM>=3` and `<PORT>=8080`. The `workers` command will create three workers listening on ports 8080, 8081, and 8082. The `workers` command is basically a convenience wrapper around the `worker-exec` command. The `workers` command does three things for you: (1) creates a config file for each worker, based on `template.json`, (2) invokes `worker-exec` for each requested worker instance, and (3) makes the workers run in the background so they continue executing even if you exit the terminal.

When you want to stop a local OpenLambda cluster, you can do so by executing the `kill` command:

```
$./admin kill --cluster=<NAME>
```

This will halt any processes or containers associated with the cluster.

In addition to the above commands for managing OpenLambda workers, two admin commands are also available for managing an OpenLambda handler store. First, you may launch the OpenLambda registry with the following `registry` command:

```
$./admin registry --port=<PORT> --access-key=<KEY> --secret-key=<SECRET>
```

The registry will start listening on the designated port. You may generate the KEY and SECRET randomly yourself if you wish (or you may use some other hard-to-guess SECRET). Keep these values handy for later uploading handlers.

The `<ROOT>/config/template.json` file specifies registry mode and various registry options. You may manually set these, but as a convenience, the `registry` command will automatically populate the configuration file for you when you launch the registry process. Thus, to avoid manual misconfiguration, we recommend running `./admin registry` before running `./admin workers`. Or, if you wish to use the local-directory mode for your registry, simply never run `./admin registry` (the default configs use local-directory mode).

After the registry is running, you may upload handlers to it via the following command:

```
$./admin upload --cluster=<NAME> --handler=<HANDLER-NAME> \
 --file=<TAR> --access-key=<KEY> \
 --secret-key=<SECRET>
```

The above command should use the KEY/SECRET pair used when you launched the registry earlier. The `<TAR>` can refer to a handler bundle. This is just a `.tar.gz` containing (at a minimum) a `lambda_func.py` file (for the code) and a `packages.txt` file (to specify the Python dependencies).

### 12.8.5.1 Writing Handlers

**O** describe how to write and upload handlers

### 12.8.5.2 Cluster Directory

Suppose you just ran the following:

```
$ admin new --cluster=./my-cluster
```

You will find six subdirectories in the `my-cluster` directory: `config`, `logs`, `base`, `packages`, `registry`, and `workers`.

The `config` directory will contain, at a minimum, a `template.json` file. Once you start workers, each worker will have an additional config file in this directory named `worker-<N>.json` (the admin tool creates these by copying first copying `template.json`, then populating additional fields specific to the worker).

Each running worker will create two files in the `logs` directory: `worker-<N>.out` and `worker-<N>.pid`. The “.out” files contain the log output of the workers; this is a good place

to start if the workers are not reachable or if they are returning unexpected errors. The “.pid” files each contain a single number representing the process ID of the corresponding worker process; this is mostly useful to the admin kill tool for identifying processes to halt.

All OpenLambda handlers run on the same base image, which is dumped via Docker into the `my-cluster/base` directory. This contains a standard Ubuntu image with additional OpenLambda-specific components. This base is accessed on a read-only basis by every handler when using SOCK containers.

The `./my-cluster/packages` directory is mapped (via a read-only bind mount) into all containers started by the worker, and contains all of the PyPI packages installed to workers on this machine.

As discussed earlier, OpenLambda can use a separate registry service to store handlers, or it can store them in a local directory; the latter is more convenient for development and testing. Unless configured otherwise, OpenLambda will treat the `./my-cluster/registry` directory as a handler store. Creating a handler named “X” is as simple as creating a directory named `./my-cluster/registry/X` and writing your code therein. No compression is necessary in this mode; the handler code for “X” can be saved here: `./my-cluster/registry/X/lambda_func.py`.

Each worker has its own directory for various state. The storage for worker N is rooted at `./my-cluster/workers/worker-<N>`. Within that directory, handler containers will have scratch space at `./handlers/<handler-name>/<instance-number>`. For example, all containers created to service invocations of the “echo” handler will have scratch space directories inside the `./handlers/echo` directory. Additionally, there is a directory `./my-cluster/workers/worker-<N>/import-cache` that contains the communication directory mapped into each import cache entry container.

Suppose there is an instance of the “echo” handler with ID “3”. That container will have its scratch space at `./handlers/echo/3` (within the worker root, `./my-cluster/workers/worker-<N>`). The handler may write temporary files in that directory as necessary. In addition to these, there will be three files: `server_pipe` sock file (used by the worker process to communicate with the handler) and `stdout` and `stderr` files (handler output is redirected here). When debugging a handler, checking these output files can be quite useful.

Note that the same directory can appear at different locations in the host and in a guest container. For example, containers for two handlers named “function-A” and “function-B” might have scratch space on the host allocated at the following two locations:

```
$./my-cluster/workers/worker-0/handlers/function-A/123
$./my-cluster/workers/worker-0/handlers/function-B/321
```

As a developer debugging the functions, you may want to peek in the above directories to look for handler output and generated files. However, in order to write code for a handler that generates output in the above locations, you will need to write files to the `/host` directory (regardless of whether you’re writing code for function-A or function-B) because that is where scratch space is always mapped within a lambda container.

## 12.8.6 Configuration

❶ document the configuration parameters and how they interact. Also describe how to use the `packages.txt` file in a handler directory to specify dependencies.

## 12.8.7 Architecture

❶ concise description of the architecture.

## 13 MESSAGING

### 13.1 MQTT

---

---



#### Learning Objectives

- Understand Message systems for the cloud
  - Learn about MQTT
- 
- 

With the increase importance of cloud computing and the increased number of edge devices and their applications, such as sensor networks, it is crucial to enable fast communication between the sensing devices and actuators, which may not be directly connected, as well as cloud services that analyze the data. To allow services that are built on different software and hardware platforms to communicate, a data agnostic, fast and service is needed. In addition to communication, the data generated by these devices, services, and sensors must be analyzed. Security aspects to relay this data is highly important. We will introduce a service called MQTT, which is a common, easy to use, queuing protocol that helps meet these requirements.

#### 13.1.1 Introduction

As Cloud Computing and Internet of Things (IoT) applications and sensor networks become commonplace and more and more devices are being connected, there is an increased need to allow these devices to communicate quickly and securely. In many cases these edge devices have very limited memory and need to conserve power. The computing power on some of these devices is limited so that the sensory data need to be analyzed remotely. Furthermore, they may not even have enough computing capacity to process traditional HTTP web requests efficiently [92][93] or these traditional Web-based services are too resource hungry. Monitoring the state of a remotely located sensor using HTTP would require sending requests and receiving responses to and from the device frequently, which may not be efficient on small

circuits or embedded chips on edge computing sensors [92].

*Message Queue Telemetry Transport* (MQTT) is a lightweight machine to machine (M2M) messaging protocol, based on a client/server publish-subscribe model. It provides a simple service allowing us to communicate between sensors, and services based on a subscription model.

MQTT was first developed in 1999 to connect oil pipelines [93]. The protocol has been designed to be used on top of TCP/IP protocol in situations where network bandwidth, and available memory are limited allowing low power usage. However, as it is implemented on top of TCP/IP it is reliable in contrast to other protocols such as UDP. It allows efficient transmission of data to various devices listening for the same event, and is scalable as the number of devices increase [94][95].

MQTT is becoming more popular than ever before with the increasing use of mobile device and smartphone applications such as Facebook's Messenger and Amazon Web Services. This protocol is used in WIFI or low bandwidth network. MQTT does not require any connection with the content of the message.

The current support for MQTT is conducted as part of the Eclipse Phao project [96]. As MQTT is a protocol many different clients in various languages exist. This includes languages such as Python, C, Java, Lua, and many more.

The current standard of MQTT is available at

<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>

### **13.1.2 Publish Subscribe Model**

MQTT works via a publish-subscribe model that contains 3 entities: (1) a Raspberry Pi publisher, that sends a message, (2) a broker, that maintains queue of all messages based on topics and (3) multiple subscribers that subscribe to various topics they are interested in [97]. See [Figure 170](#)

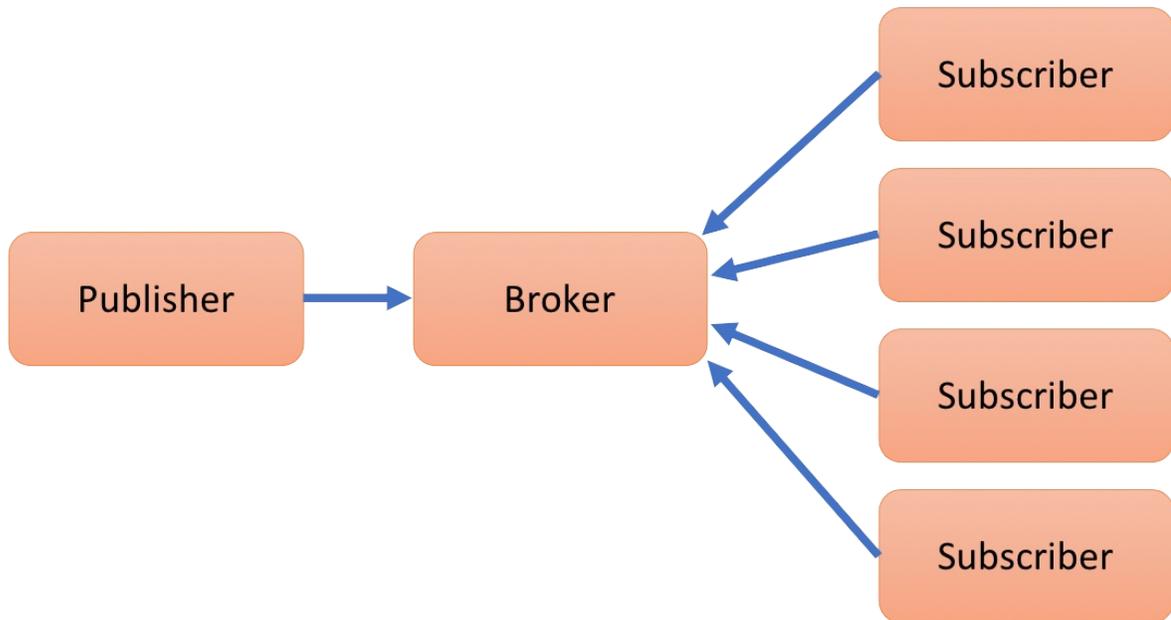


Figure 170: MQTT publish subscriber model

This allows for decoupling of functionality at various levels. The publisher and subscriber do not need to be close to each other and do not need to know each others identity. They need only to know the broker, as the publisher and the subscribers do not have to be running either at the same time nor on the same hardware [98].

Ready to use implementation exist to be deployed as brokers in the users application frameworks. A broker is a service that relays information between the client and servers. Common brokers include the open source Mosquito broker [95] and the Eclipse Phao MQTT Broker [96].

### 13.1.2.1 Topics

MQTT implements a hierarchy of topics that are relates to all messages. These topics are recognised by strings separated by a forward-slash (/), where each part represents a different topic level. This is a common model introduced in file systems but also in internet URLs.

A topic looks therefore as follows:

```
topic-level0/topic-level1/topic-level2.
```

Subscribers can subscribe to different topics via the broker. Subscribing to `topic-`

`level0` allows the subscriber to receive all messages that are associated with topics that start with `topic-level0`. This allows subscribers to filter what messages to receive based on the topic hierarchy. Thus, when a publisher publishes a message related to a topic to the broker, the message is forwarded to all the clients that have subscribed to the topic of the message or a topic that has a lower depth of hierarchy [98] [97].

This is different from traditional point-to-point message queues as the message is forwarded to multiple subscribers, and allows for flexibility of dealing with subscribed topics not only on the server but also on the subscriber side [98]. The basic steps in a MQTT client subscriber application include to (1) connect to the broker, (2) subscribe to some topics, (3) wait for messages and (4) perform the appropriate action when a certain message is received [94].

### 13.1.2.2 Callbacks

One of the advantages of using MQTT is that it supports asynchronous behaviour with the help of callbacks. Both the publisher and subscriber can use non-blocking callbacks to act upon message exchanges. [98][99].

For example, the `paho-mqtt` package for python provides callbacks methods including `on-connect()`, `on-message()` and `on-disconnect()`, which are invoked when the connection to the broker is complete, a message is received from the broker, and when the client is disconnected from the broker respectively. These methods are used in conjunction with the `loop-start()` and `loop-end()` methods which start and end an asynchronous loop that listens for these events invoking the relevant callbacks. Hence it frees the services to perform other tasks [99] when no messages are available, thus reducing overhead.

### 13.1.2.3 Quality of Service

MQTT has been designed to be flexible allowing for the change of quality of service (QoS) as desired by the application. Three basic levels of QoS are supported by the protocol: Atmost-once (QoS level 0), Atleast-once (QoS level 1) and Atmost-once (QoS level 2) [99], [100].

QoS level 0:

The QoS level of 0 is used in applications where some dropped messages may not affect the application. Under this QoS level, the broker forwards a message to the subscribers only once and does not wait for any acknowledgement [100][99].

#### QoS Level 1:

The QoS level of 1 is used in situations where the delivery of all messages is important and the subscriber can handle duplicate messages. Here the broker keeps on resending the message to a subscriber after a certain timeout until the first acknowledgement is received.

#### QoS Level 3:

A QoS level of 3 is used in cases where all messages must be delivered and no duplicate messages should be allowed. In this case the broker sets up a handshake with the subscriber to check for its availability before sending the message [99], [100].

The various levels of quality of service allow the use of this protocol with different service level expectations.

### **13.1.3 Secure MQTT Services**

MQTT specification uses TCP/IP to deliver the messages to the subscribers, but it does not provide security by default to enable resource constrained IoT devices. “It allows the use of username and password for authentication, but by default this information is sent as plain text over the network, making it susceptible to man-in-the middle attacks” [101], [102]. Therefore, to support sensitive applications additional security measures need to be integrated through other means. This may include for example the use of Virtual Private Networks (VPNs), Transport Layer Security, or application layer security [102].

#### **13.1.3.1 Using TLS/SSL**

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are cryptographic protocols that establish the identity of the server and client with the help of a handshake mechanism which uses trust certificates to establish

identities before encrypted communication can take place [103]. If the handshake is not completed for some reason, the connection is not established and no messages are exchanged [102]. “Most MQTT brokers provide an option to use TLS instead of plain TCP and port 8883 has been standardized for secured MQTT connections” [101].

Using TLS/SSL security however comes at an additional cost. If the connections are short-lived then most of the time is spent in verifying the security of the handshake itself, which in addition to using time for encryption and decryption, may take up few kilobytes of bandwidth. In case the connections are short-lived, temporary session IDs and session tickets can be used as alternative to resume a session instead of repeating the handshake process. If the connections are long term, the overhead of the handshake is negligible and TLS/SSL security should be used [101], [102].

### **13.1.3.2 Using OAuth**

OAuth is an open protocol that allows access to a resource without providing unencrypted credentials to the third party. Although MQTT protocol itself does not include authorization, many MQTT brokers include authorization as an additional feature [103]. OAuth2.0 uses JSON Web Tokens which contain information about the token and the user and are signed by a trusted authorization server [104].

When connecting to the broker this token can be used to check whether the client is authorised to connect at this time or not. Additionally the same validations can be used when publishing or subscribing to the broker. The broker may use a third party resource such as LDAP (lightweight directory access protocol) to look up authorizations for the client [104]. Since there can be a large number of clients and it can become impractical to authorize everyone, clients may be grouped and the authorizations may be checked for each group [103].

### **13.1.4 Integration with Other Services**

As the individual IoT devices perform their respective functions in the sensor network, a lot of data is generated which needs to be processed. MQTT allows easy integration with other services, that have been designed to process this data.

Let us provide some examples of MQTT integration into other Services.

Apache Storm.

Apache storm is a distributed processing system that allows real time processing of continuous data streams, much like Hadoop works for batch processing [105]. Apache storm can be easily integrated with MQTT as shown in [106] to get real time data streams and allow analytics and online machine learning in a fault tolerant manner [107].

ELK stack.

ELK stack (elastic-search, logstash and kibana) is an opensource project designed for scalability which contains three main software packages, the *elastic-search* search and analytics engine, *logstash* which is a data collection pipeline and *kibana* which is a visualization dashboard [108]. Data from an IoT network can be collected, analysed and visualized easily with the help of the ELK stack as shown in [109] and [110].

### 13.1.5 MQTT in Production

When using optimized MQTT broker services, MQTT can be utilized for enterprise and production environments. A good example is the use of EMQ (Erlang MQTT Broker) that provides a highly scalable, distributed and reliable MQTT broker for enterprise-grade applications [111].

### 13.1.6 Installation

The installation of an MQTT server based on paho is very simple.

#### 13.1.6.1 MacOS install

On OSX yo need to first install mosquito, which is easiest to install with `brew`

Step 1: Installing Mosquito clients

Open a terminal and use homebrew to install mosquito and than you can install paho with pip

```
brew install mosquitto
pip install paho-mqtt
```

You need to start the mosquito service buy hand to use it.

### 13.1.6.2 MacOS Advanced Service install

 *We recommend that this is only be done if you truly need a production system. For our class you will not need this.*

You can integrate mosquito service on boot, while adding it via LaunchAgents. This can be achieved by linking it as follows:

```
ln -sfv /usr/local/opt/mosquitto/*.plist ~/Library/LaunchAgents
```

Next you need to restart the server as follows:

```
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.mosquitto.plist
```

Now you can test the installation and ensure the server is running successfully. Open a new command window and start a listener.

```
mosquitto_sub -t topic/state
```

To test teh setup you can in another window, send a message to the listener.

```
mosquitto_pub -t topic/state -m "Hello World"
```

This ensures the server is running.

### 13.1.6.3 Ubuntu install

On ubuntu you need to first install mosquito, than with pip you install `paho-mqt`

```
$ sudo apt-get install mosquitto mosquitto-clients
$ pip install paho-mqtt
```

### 13.1.6.4 Raspberry Pi Setup

If you have experimented with the Raspberry P and MQTT, you can contribute to this section.

#### 13.1.6.4.1 Broker

You will need to add the mosquito repository to the known repositories as follows:

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list
apt-get update
```

Mosquito is installed by implementing the following command:

```
apt-get install mosquito
```

#### 13.1.6.4.2 Client

The MQTT client needs to be installed on raspberry pi by running the following command:

```
apt-get install mosquitto-clients
```

### 13.1.7 Server Usecase

In this example we are demonstrating how to set up a MQTT broker, a client and a subscriber while just using regular servers and clients. The code of this example is located at:

- <https://github.com/bigdata-i523/sample-hid000/tree/master/experiment/mqtt>

A test program that starts a MQTT broker and client showcases how simple the interactions between the publisher and subscribers are while using a higher level API such as provided through the python client library of Paho.

```
import paho.mqtt.client as mqtt
import time

def on_message(client, userdata, message):
 print("message received ",
 str(message.payload.decode("utf-8")))
 print("message topic=", message.topic)
 print("message qos=", message.qos)
 print("message retain flag=", message.retain)

def on_log(client, userdata, level, buf):
 print("log: ",buf)

broker_address="localhost"
broker_address="test.mosquitto.org"
broker_address="broker.hivemq.com"
broker_address="iot.eclipse.org"
```

```
print("creating new instance")
client = mqtt.Client("i523") #create new instance
client.on_log=on_log
client.on_message=on_message #attach function to callback

print("connecting to broker")
client.connect(broker_address) #connect to broker
client.loop_start() #start the loop

print("Subscribing to topic","robot/leds/led1")
client.subscribe("robot/leds/led1")

print("Publishing message to topic","robot/leds/led1")
client.publish("robot/leds/led1","OFF")

time.sleep(4) # wait
client.loop_stop() #stop the loop
```

## 13.1.8 IoT Use Case with a Raspberry PI

MQTT can be used in a variety of applications. This section explores a particular use case of the protocol. A small network was set up with three devices to simulate an IoT environment, and actuators were controlled with the help of messages communicated over MQTT.

The code for the project is available at

- <https://github.com/bigdata-i523/hid201/tree/master/experiment/mqtt>

### 13.1.8.1 Requirements and Setup

The setup used three different machines. A laptop or a desktop running the MQTT broker, and two raspberry pis configured with raspbian operating system. Eclipse Paho MQTT client was setup on each of the raspberry pis [99]. Additionally all three devices were connected to an isolated local network.

GrovePi shields for the raspberry pis, designed by Dexter Industries were used on each of the raspberry pi to connect the actuators as they allow easy connections to the raspberry pi board [112]. The actuators used were Grove relays [113] and Grove LEDs [114] which respond to the messages received via MQTT.

To control the leds and relays, the python library cloudmesh-pi [115], developed at Indiana University was used. The library consists of interfaces for various IoT sensors and actuators and can be easily used with the grove modules.

### **13.1.8.2 Results**

The two Raspberry Pis subscribe connect to the broker and subscribe with different topics. The raspberry pis wait for any messages from the broker. A publisher program that connects to the broker publishes messages to the broker for the topics that the two raspberry pi had registered. Each raspberry pi receives the corresponding message and turns the LEDs or relays on or off as per the message.

On a local network this process happens in near real time and no delays were observed. Eclipse IoT MQTT broker (*iot.eclipse.org*) was also tried which also did not result in any significant delays.

Thus it is observed that two raspberry pi can be easily controlled using MQTT. This system can be extended to include arbitrary number of raspberry pi and other devices that subscribe to the broker. If a device fails, or the connection from one device is broken, other devices are not affected and continue to perform the same.

This project can be extended to include various other kinds of sensors and actuators. The actuators may subscribe to topics to which various sensors publish their data and respond accordingly. The data of these sensors can be captured with the help of a data collector which may itself be a different subscriber, that performs analytics or visualizations on this data.

### **13.1.9 Conclusion**

We see that as the number of connected devices increases and their applications become commonplace, MQTT allows different devices to communicate with each other in a data agnostic manner. MQTT uses a publish-subscribe model and allows various levels of quality of service requirements to be fulfilled. Although MQTT does not provide data security by default, most brokers allow the use of TLS/SSL to encrypt the data. Additional features may be provided by the broker to include authorization services. MQTT can be easily integrated with other services to allow collection and analysis of data. A small environment was simulated that used MQTT broker and clients running on raspberry pi to control actuators

## 13.1.10 Exercises

E.MQTT.1:

*Develop a temperature broker, that collects the temperature from a number of machines and clients can subscribe to the data and visualize it.*

E.MQTT.2:

*Develop a CPU load broker, that collects the cpu load from a number of machines and clients can subscribe to the data and visualize it.*

E.MQTT.3:

*Develop install instructions and examples on how to use MQTT on Raspberry Pi.*

E.MQTT.4:

*Develop a broker with a variety of topics that collects data from a Raspberry Pi or Raspberry Pi cluster and visualize it.*

E.MQTT.5:

*Explore hosted services for MQTT while at the same time remembering that they could pose a security risk. Can any of the online services be used to monitor a cluster safely?*

## 13.2 PYTHON APACHE AVRO

---

Although Apache Avro is not directly a messaging system, it uses messaging to communicate between components while serializing and deserializing object defined with a schema. In addition it provides data structures, remote procedure call (RPC), a container file to store persistent data and simple integration with dynamic languages [116]. Avro depends on schemas, which are defined with JSON. This facilitates implementation in other languages that have the JSON libraries. The key advantages of Avro are schema evolution - Avro will handle

the missing/extra/modified fields, dynamic typing - serialization and deserialization without code generation, untagged data - data encoding and faster data processing by allowing data to be written without overhead.

The following steps illustrate using Avro to serialize and deserialize data with example modified from Apache Avro 1.8.2 Getting Started (Python) [117].

## 13.2.1 Download, Unzip and Install

Please download the following zipped file [avro-python3-1.8.2.tar.gz](http://avro-python3-1.8.2.tar.gz).

Unzip it and conduct the install

```
$ tar xvf avro-1.8.2.tar.gz
$ cd avro-1.8.2
$ python setup.py install
```

To check successful installation, import avro in python without error message:

```
$ python
>>> import avro
```

## 13.2.2 Defining a schema

Use a simple schema for students contributed in cloudmesh as an example: paste the following lines into an empty text file with the name it `student.avsc`

```
{ "namespace": "cloudmesh.avro",
 "type": "record",
 "name": "Student",
 "fields": [
 { "name": "name", "type": "string" },
 { "name": "hid", "type": "string" },
 { "name": "age", "type": ["int", "null"] },
 { "name": "project_name", "type": ["string", "null"] }
]
}
```

This schema defines a record representing a hypothetical student, which is defined to be a record with the name `Student` and 4 fields, namely name, hid, age and project name. The type of each of the field needs to be provided. If any field is optional, one could use the list including `null` to define the type as shown in age and project name in the example schema. Further, a namespace `cloudmesh.avro` is also defined, which together with the name attribute defines the full name of the schema (cloudmesh.avro.Student in this case).

## 13.2.3 Serializing

The following piece of python code illustrates serialization of some data

```
import avro.schema
from avro.datafile import DataFileWriter
from avro.io import DatumWriter

schema = avro.schema.parse(open("student.avsc", "rb").read())

writer = DataFileWriter(open("students.avro", "wb"), DatumWriter(), schema)
writer.append({"name": "Albert Zweistein",
 "hid": "hid-sp18-405",
 "age": 99,
 "project_name": "hadoop with docker"})
writer.append({"name": "Ben Smith",
 "hid": "hid-sp18-309",
 "project_name": "spark with docker"})
writer.append({"name": "Alice Johnson",
 "hid": "hid-sp18-208",
 "age": 27})
writer.close()
```

The code does the following:

- Imports required modules
- Reads the schema `student.avsc` (make sure that the schema file is placed in the same directory as the python code)
- Create a `DataFileWriter` called `writer`, for writing serialized items to a data file on disk
- Use `DataFileWriter.append()` to add data points to the data file. Avro records are represented as Python dicts.
- The resulting data file saved on the disk is named `students.avro`
- This instruction is for Python2. If one is using Python3, change

```
schema = avro.schema.parse(open("student.avsc", "rb").read())
```

to:

```
schema = avro.schema.Parse(open("student.avsc", "rb").read())
```

since the method name has a different case in Python3.

## 13.2.4 Deserializing

The following python code illustrates deserialization

```
from avro.datafile import DataFileReader
from avro.io import DatumReader
```

```
reader = DataFileReader(
 open("students.avro", "rb"), DatumReader())
for student in reader:
 print (student)
reader.close()
```

The code does the following:

- Imports required modules
- Use *DatafileReader* to read the serialized data file *students.avro*, it is an iterator
- Returns the data in a python dict

The output should look like:

```
{'name': 'Albert Zweistein',
 'hid': 'hid-sp18-405',
 'age': 29,
 'project_name': 'hadoop with docker'
}
{'name': 'Ben Smith',
 'hid': 'hid-sp18-309',
 'age': None,
 'project_name': 'spark with docker'
}
{'name': 'Alice Johnson',
 'hid': 'hid-sp18-208',
 'age': 27,
 'project_name': None
}
```

## 13.2.5 Resources

- The steps and instructions are modified from [Apache Avro 1.8.2 Getting Started \(Python\)](#) [117].
- The Avro Python library does not support code generation, while Avro used with Java supports code generation, see [Apache Avro 1.8.2 Getting Started \(Java\)](#) [118].
- Avro provides a convenient way to represent complex data structures within a Hadoop MapReduce job. Details about Avro are documented in [Apache Avro 1.8.2 Hadoop MapReduce guide](#) [119].
- For more information on schema files and how to specify name and type of a record can be found at [record specification](#) [120].

## 14 GO

### 14.1 INTRODUCTION TO GO FOR CLOUD COMPUTING

---



#### Learning Objectives

- Learn quickly Go under the assumption you know a programming language
  - Work with Go modules
  - Conduct some Go examples
  - Learn about REST services in Go
  - Learn how access virtual machines from Go
  - Learn how to interface with kubernetes in Go
- 



Go Logo

Go is a programming language that has been introduced by Google to replace the C++ language. Online documentation about go is available also from the official Go [Documentation](#) [121] Web page from which our material is derived.

The language Go has at its goal to be expressive, concise, clean, and efficient. It includes concurrency mechanisms with the goal to make it easy to write programs that can utilize multicore and networked features of modern computer systems and infrastructure easily with language features. However in contrast to languages such as python and ruby, it introduces in addition to static types explicitly types supporting concurrent programming such as channels that have already been used in the early days of programming for example as part of CSP [122] and OCCAM [123] [124].

In contrast to languages such as Python, Go is designed to compiled to machine code. However garbage collection and run-time reflection are build in, exposing this functionality similar to languages such as python. Hence, it is designed to provide the programmer a fast, statically typed, compiled language that feels like

a dynamically typed, interpreted language.

According to the [TIOBE \[125\]](#) index for programming languages Go has reached for November 2018 the 16th spot. However it is rated only with 1.081% with a declining rating but increase in the ranking. This trend is even more prominently depicted when looking at google trends in [Figure 171](#).

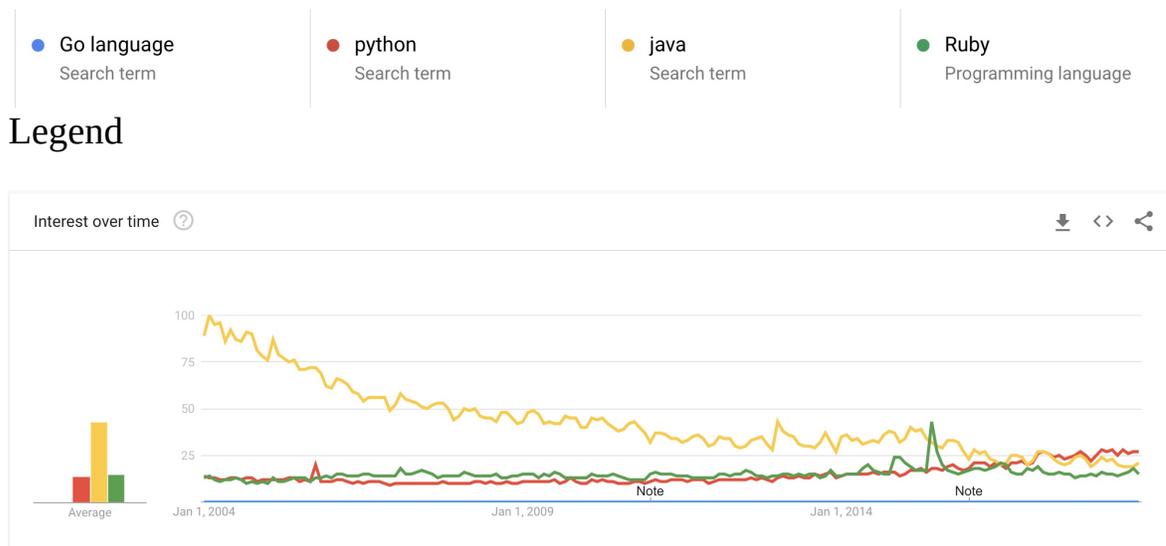


Figure 171: Google trends for selected programming languages

### 14.1.1 Organization of the chapter

The material presented in this chapter introduces the reader first to the basic concepts and features of the Go language and system. This includes installation (see [Section 14.2](#)) and compiling (see [Section 14.4](#)), have a basic understanding of the programming language (see [Section 14.4](#)), use standard library and become familiar with package management (see [Section 14.5](#)). Next we will focus on aspects of the Go language that are especially useful for Cloud computing. This includes the review of how to develop REST services with various frameworks such as Gorilla (see ) and OpenAPI (see [Section 14.8](#)). You will than be introduced on how to access virtual machines (see [Section 14.10](#)) and containers (see [Section 14.10](#)).

In order to to use Go we recommend that you have a computer fulfilling the following requirements:

- Have the most up to date version of Go installed

- Be familiar with the Linux command line as showcased in [126]
- Familiarity with a text editor such as emacs, which we prefer as it supports nicely not only Go but any other language or document format we typically use in our activities. Alternatives are discussed in [Section 1.5](#).

## 14.1.2 References

The following references may be useful for you to find out more about go. We have not gone to the list in detail, but want to make you are of some of them that we found through simple searches in Google search. If you find others or you have a favorite, let us know and we will add them and mark them appropriately.

- [golang.org](#) [121].
- [Go cheat sheet](#) [127].
- [The Little Go Book](#) [128].
- [Learn Go in an Hour - Video](#) 2015-02-15
- [Learning to Program in Go](#), a multi-part video training class.
- [Go By Example](#) provides a series of annotated code snippets [129].
- [Learn Go in Y minutes](#) is a top-to-bottom walk-through of the language [130].
- [Workshop-Go](#) - Startup Slam Go Workshop - examples and slides [131].
- [Go Fragments](#) - A collection of annotated Go code examples [132].
- [50 Shades of Go: Traps, Gotchas, Common Mistakes for New Golang Devs](#) [133]
- [Golang Tutorials](#) - A free online class [134].
- [The Go Bridge Foundry](#) [135] - A member of the [Bridge Foundry](#) [136] family, offering a complete set of free Go training materials with the goal of bringing Go to under-served communities.
- [Golangbot](#) - Tutorials to get started with programming in Go [137].
- [Algorithms to Go](#) - Texts about algorithms and Go, with plenty of code examples [138].
- [Go Language Tutorials](#) - List of Go sites, blogs and tutorials for learning Go language [139].
- [Golang Development Video Course](#) - A growing list of videos focused purely on Go development 2019-02-10.

## 14.2 INSTALLATION

---

In case Go is not installed on your computer it is lease to install. Up to date informtion about the install process for your acitecture is available from <https://golang.org/doc/install>

We list now some brief instalation notes

We recommend that you use the tarball for Linux, and MacOS, that you can obtain for your platform here:

- <https://golang.org/dl/>

Once downloaded, unpack it with

```
$ sudo tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

Packaged installers are also available for macOS, and Windows. They may provide you with the familiar platform specific instalation methods.

## 14.3 EDITORS SUPPORTING GO

---

A large number of editor compatibilities and plugins are listed at

- <https://github.com/golang/go/wiki/IDEsAndTextEditorPlugins>

We recommend that you identify an editor form that list that will work for you. Due to the universality of emacs and its use for managing LaTeX as ewll as bibtex, we recommend that you use emacs, Important is that the editor supports lin breaks at the 80 character limit so the code is no=icely formatted for github. If your editor does not support this feature use emacs.

If your version of emacs does not yet have support for go, yo ucan find the go mode at

- <https://www.emacswiki.org/emacs/GoMode> [?]

The documentation to it is provided at

- [http://dominik.honnef.co/posts/2013/03/writing\\_go\\_in\\_emacs/](http://dominik.honnef.co/posts/2013/03/writing_go_in_emacs/) [?]

Other editors may include

- [GoLand](#) [?] which however in contrast to PyCharm Community edition is not free. However as student and faculty one can get a free license via <https://www.jetbrains.com/student/>
- [Atom](#) [?]
- [vim](#) [?]



Please help us complete this section while letting us know how each editor supports 80 character line wrap mode.

## 14.4 GO LANGUAGE

---

Go is a computer language developed by Google with the goal to “build simple, reliable, and efficient software”. The language is open source and the main Web page is <https://golang.org/>

Go is specifically a systems-level programming language for large, distributed systems and highly-scalable network servers. It is meant to replace C++ and Java in terms of Google’s needs. Go was meant to alleviate some of the slowness and clumsiness of development of very large software systems.

- slow compilation and slow execution
- programmers that collaborate using different subsets of languages
- readability and documentation
- language consistency
- versioning issues
- multi-language builds
- dependencies being hard to maintain

The following program from the <https://golang.org/> web page shows the customary Hello World example:

```
package main

import "fmt"

func main() {
 /* This is a very easy program. */
 fmt.Println("Hello World!")
}
```

## 14.4.1 Concurrency in Go

Making a program to be able to run multiple tasks simultaneously is known as concurrency. Go language supports concurrency with `GoRoutines`, `Channels`, and `select` statements.

### 14.4.1.1 GoRoutines (execution)

A GoRoutine in the Go programming language is a lightweight thread that is managed by Go runtime. If you just put ‘go’ before a function, it means that it will execute concurrently with the rest of the code.

To create a goroutine we use the keyword `go` followed by a function invocation:

```
package main

import "fmt"

func f(n int) {
 for i := 0; i < 10; i++ {
 fmt.Println(n, ":", i)
 }
}

func main() {
 go f(0)
 var input string
 fmt.Scanln(&input)
}
```

This program consists of two `goroutines`. The first `goroutine` is implicit and is the main function itself. The second `goroutine` is created when we call `go f(0)`. Normally when we invoke a function our program will execute all the statements in a function and then return to the next line following the invocation. With a `goroutine` we return immediately to the next line and don’t wait for the function to complete.

Goroutines are lightweight and we can easily create thousands of them. We can modify our program to run 10 goroutines by doing this:

```
func main() {
 for i := 0; i < 10; i++ {
 go f(i)
 }
 var input string
 fmt.Scanln(&input)
}
```

### 14.4.1.2 Channels (communication)

Channels are pipes that connect concurrent GoRoutines. You are able to send values and signals over Channels from GoRoutine to GoRoutine. This allows for synchronizing execution.

Here is an example program using channels:

```
package main

import (
 "fmt"
 "time"
)

func pinger(c chan string) {
 for i := 0; ; i++ {
 c <- "ping"
 }
}

func printer(c chan string) {
 for {
 msg := <- c
 fmt.Println(msg)
 time.Sleep(time.Second * 1)
 }
}

func main() {
 var c chan string = make(chan string)

 go pinger(c)
 go printer(c)

 var input string
 fmt.Scanln(&input)
}
```

This program will print “ping” forever (hit enter to stop it). A channel type is represented with the keyword `chan` followed by the type of the things that are passed on the channel (in this case we are passing strings). The `<-` (left arrow) operator is used to send and receive messages on the channel. `c <- "ping"` means send “ping”. `msg := <- c` means receive a message and store it in `msg`.

### 14.4.1.3 Select (coordination)

The Select statement in Go lets you wait and watch multiple operations on a channel. Combining GoRoutines and channels will show off the true power of concurrency in Go.

Take the following code as an example:

```
func main() {
 c1 := make(chan string)
 c2 := make(chan string)

 go func() {
```

```

 for {
 c1 <- "from 1"
 time.Sleep(time.Second * 2)
 }
}()

go func() {
 for {
 c2 <- "from 2"
 time.Sleep(time.Second * 3)
 }
}()

go func() {
 for {
 select {
 case msg1 := <- c1:
 fmt.Println(msg1)
 case msg2 := <- c2:
 fmt.Println(msg2)
 }
 }
}()

var input string
fmt.Scanln(&input)
}

```

`select` picks the first channel that is ready and receives from it (or sends to it). If more than one of the channels are ready then it randomly picks which one to receive from. If none of the channels are ready, the statement blocks until one becomes available.

## 14.5 LIBRARIES

Golang comes with a list of standard libraries in the following table, and more libraries can be found on this page: <https://golang.org/pkg/>

Name	Synopsis
archive	
tar	Package tar implements access to tar archives.
zip	Package zip provides support for reading and writing ZIP archives.
bufio	Package bufio implements buffered I/O. It wraps an <code>io.Reader</code> or <code>io.Writer</code> object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.

builtin	Package builtin provides documentation for Go's predeclared identifiers.
bytes	Package bytes implements functions for the manipulation of byte slices.
compress	
bzip2	Package bzip2 implements bzip2 decompression.
flate	Package flate implements the DEFLATE compressed data format, described in RFC 1951.
gzip	Package gzip implements reading and writing of gzip format compressed files, as specified in RFC 1952.
lzw	Package lzw implements the Lempel-Ziv-Welch compressed data format, described in T. A. Welch, "A Technique for High-Performance Data Compression", Computer, 17(6) (June 1984), pp 8-19.
zlib	Package zlib implements reading and writing of zlib format compressed data, as specified in RFC 1950.
container	
heap	Package heap provides heap operations for any type that implements heap.Interface.
list	Package list implements a doubly linked list.
ring	Package ring implements operations on circular lists.
context	Package context defines the Context type, which carries deadlines, cancelation signals, and other request-scoped values across API boundaries and between processes.

crypto	Package crypto collects common cryptographic constants.
aes	Package aes implements AES encryption (formerly Rijndael), as defined in U.S. Federal Information Processing Standards Publication 197.
cipher	Package cipher implements standard block cipher modes that can be wrapped around low-level block cipher implementations.
des	Package des implements the Data Encryption Standard (DES) and the Triple Data Encryption Algorithm (TDEA) as defined in U.S. Federal Information Processing Standards Publication 46-3.
dsa	Package dsa implements the Digital Signature Algorithm, as defined in FIPS 186-3.
ecdsa	Package ecdsa implements the Elliptic Curve Digital Signature Algorithm, as defined in FIPS 186-3.
elliptic	Package elliptic implements several standard elliptic curves over prime fields.
hmac	Package hmac implements the Keyed-Hash Message Authentication Code (HMAC) as defined in U.S. Federal Information Processing Standards Publication 198.
md5	Package md5 implements the MD5 hash algorithm as defined in RFC 1321.
rand	Package rand implements a cryptographically secure random number generator.

rc4	Package rc4 implements RC4 encryption, as defined in Bruce Schneier's Applied Cryptography.
rsa	Package rsa implements RSA encryption as specified in PKCS#1.
sha1	Package sha1 implements the SHA-1 hash algorithm as defined in RFC 3174.
sha256	Package sha256 implements the SHA224 and SHA256 hash algorithms as defined in FIPS 180-4.
sha512	Package sha512 implements the SHA-384, SHA-512, SHA-512/224, and SHA-512/256 hash algorithms as defined in FIPS 180-4.
subtle	Package subtle implements functions that are often useful in cryptographic code but require careful thought to use correctly.
tls	Package tls partially implements TLS 1.2, as specified in RFC 5246.
x509	Package x509 parses X.509-encoded keys and certificates.
pkix	Package pkix contains shared, low level structures used for ASN.1 parsing and serialization of X.509 certificates, CRL and OCSP.
database	
sql	Package sql provides a generic interface around SQL (or SQL-like) databases.
driver	Package driver defines interfaces to be implemented by database drivers as used by package sql.
debug	
	Package dwarf provides access to

dwarf	DWARF debugging information loaded from executable files, as defined in the DWARF 2.0 Standard at <a href="http://dwarfstd.org/doc/dwarf-2.0.0.pdf">http://dwarfstd.org/doc/dwarf-2.0.0.pdf</a>
elf	Package elf implements access to ELF object files.
gosym	Package gosym implements access to the Go symbol and line number tables embedded in Go binaries generated by the gc compilers.
macho	Package macho implements access to Mach-O object files.
pe	Package pe implements access to PE (Microsoft Windows Portable Executable) files.
plan9obj	Package plan9obj implements access to Plan 9 a.out object files.
encoding	Package encoding defines interfaces shared by other packages that convert data to and from byte-level and textual representations.
ascii85	Package ascii85 implements the ascii85 data encoding as used in the btoa tool and Adobe's PostScript and PDF document formats.
asn1	Package asn1 implements parsing of DER-encoded ASN.1 data structures, as defined in ITU-T Rec X.690.
base32	Package base32 implements base32 encoding as specified by RFC 4648.
base64	Package base64 implements base64 encoding as specified by RFC 4648.
binary	Package binary implements simple translation between numbers and byte sequences and encoding and decoding

of variants.

csv	Package csv reads and writes comma-separated values (CSV) files.
gob	Package gob manages streams of gobs - binary values exchanged between an Encoder (transmitter) and a Decoder (receiver).
hex	Package hex implements hexadecimal encoding and decoding.
json	Package json implements encoding and decoding of JSON as defined in RFC 7159.
pem	Package pem implements the PEM data encoding, which originated in Privacy Enhanced Mail.
xml	Package xml implements a simple XML 1.0 parser that understands XML name spaces.
errors	Package errors implements functions to manipulate errors.
expvar	Package expvar provides a standardized interface to public variables, such as operation counters in servers.
flag	Package flag implements command-line flag parsing.
fmt	Package fmt implements formatted I/O with functions analogous to C's printf and scanf.
go	
ast	Package ast declares the types used to represent syntax trees for Go packages.
build	Package build gathers information about Go packages.

constant	Package constant implements Values representing untyped Go constants and their corresponding operations.
doc	Package doc extracts source code documentation from a Go AST.
format	Package format implements standard formatting of Go source.
importer	Package importer provides access to export data importers.
parser	Package parser implements a parser for Go source files.
printer	Package printer implements printing of AST nodes.
scanner	Package scanner implements a scanner for Go source text.
token	Package token defines constants representing the lexical tokens of the Go programming language and basic operations on tokens (printing, predicates).
types	Package types declares the data types and implements the algorithms for type-checking of Go packages.
hash	Package hash provides interfaces for hash functions.
adler32	Package adler32 implements the Adler-32 checksum.
crc32	Package crc32 implements the 32-bit cyclic redundancy check, or CRC-32, checksum.
crc64	Package crc64 implements the 64-bit cyclic redundancy check, or CRC-64, checksum.
	Package fnv implements FNV-1 and

fnv	FNV-1a, non-cryptographic hash functions created by Glenn Fowler, Landon Curt Noll, and Phong Vo.
html	Package html provides functions for escaping and unescaping HTML text.
template	Package template (html/template) implements data-driven templates for generating HTML output safe against code injection.
image	Package image implements a basic 2-D image library.
color	Package color implements a basic color library.
palette	Package palette provides standard color palettes.
draw	Package draw provides image composition functions.
gif	Package gif implements a GIF image decoder and encoder.
jpeg	Package jpeg implements a JPEG image decoder and encoder.
png	Package png implements a PNG image decoder and encoder.
index	
suffixarray	Package suffixarray implements substring search in logarithmic time using an in-memory suffix array.
io	Package io provides basic interfaces to I/O primitives.
ioutil	Package ioutil implements some I/O utility functions.
log	Package log implements a simple logging package.

syslog	Package syslog provides a simple interface to the system log service.
math	Package math provides basic constants and mathematical functions.
big	Package big implements arbitrary-precision arithmetic (big numbers).
bits	Package bits implements bit counting and manipulation functions for the predeclared unsigned integer types.
cmplx	Package cmplx provides basic constants and mathematical functions for complex numbers.
rand	Package rand implements pseudo-random number generators.
mime	Package mime implements parts of the MIME spec.
multipart	Package multipart implements MIME multipart parsing, as defined in RFC 2046.
quotedprintable	Package quotedprintable implements quoted-printable encoding as specified by RFC 2045.
net	Package net provides a portable interface for network I/O, including TCP/IP, UDP, domain name resolution, and Unix domain sockets.
http	Package http provides HTTP client and server implementations.
cgi	Package cgi implements CGI (Common Gateway Interface) as specified in RFC 3875.
cookiejar	Package cookiejar implements an in-memory RFC 6265-compliant <code>http.CookieJar</code> .

fcgi	Package fcgi implements the FastCGI protocol.
httptest	Package httptest provides utilities for HTTP testing.
httptrace	Package httptrace provides mechanisms to trace the events within HTTP client requests.
httputil	Package httputil provides HTTP utility functions, complementing the more common ones in the net/http package.
pprof	Package pprof serves via its HTTP server runtime profiling data in the format expected by the pprof visualization tool.
mail	Package mail implements parsing of mail messages.
rpc	Package rpc provides access to the exported methods of an object across a network or other I/O connection.
jsonrpc	Package jsonrpc implements a JSON-RPC 1.0 ClientCodec and ServerCodec for the rpc package.
smtp	Package smtp implements the Simple Mail Transfer Protocol as defined in RFC 5321.
textproto	Package textproto implements generic support for text-based request/response protocols in the style of HTTP, NNTP, and SMTP.
url	Package url parses URLs and implements query escaping.
os	Package os provides a platform-independent interface to operating system functionality.
exec	Package exec runs external commands.

signal	Package signal implements access to incoming signals.
user	Package user allows user account lookups by name or id.
path	Package path implements utility routines for manipulating slash-separated paths.
filepath	Package filepath implements utility routines for manipulating filename paths in a way compatible with the target operating system-defined file paths.
plugin	Package plugin implements loading and symbol resolution of Go plugins.
reflect	Package reflect implements run-time reflection, allowing a program to manipulate objects with arbitrary types.
regexp	Package regexp implements regular expression search.
syntax	Package syntax parses regular expressions into parse trees and compiles parse trees into programs.
runtime	Package runtime contains operations that interact with Go's runtime system, such as functions to control goroutines.
cgo	Package cgo contains runtime support for code generated by the cgo tool.
debug	Package debug contains facilities for programs to debug themselves while they are running.
msan	
pprof	Package pprof writes runtime profiling data in the format expected by the pprof visualization tool.

race	Package race implements data race detection logic.
trace	Package trace contains facilities for programs to generate traces for the Go execution tracer.
sort	Package sort provides primitives for sorting slices and user-defined collections.
strconv	Package strconv implements conversions to and from string representations of basic data types.
strings	Package strings implements simple functions to manipulate UTF-8 encoded strings.
sync	Package sync provides basic synchronization primitives such as mutual exclusion locks.
atomic	Package atomic provides low-level atomic memory primitives useful for implementing synchronization algorithms.
syscall	Package syscall contains an interface to the low-level operating system primitives.
js	Package js gives access to the WebAssembly host environment when using the js/wasm architecture.
testing	Package testing provides support for automated testing of Go packages.
iotest	Package iotest implements Readers and Writers useful mainly for testing.
quick	Package quick implements utility functions to help with black box testing.
text	

scanner	Package scanner provides a scanner and tokenizer for UTF-8-encoded text.
tabwriter	Package tabwriter implements a write filter (tabwriter.Writer) that translates tabbed columns in input into properly aligned text.
template	Package template implements data-driven templates for generating textual output.
parse	Package parse builds parse trees for templates as defined by text/template and html/template.
time	Package time provides functionality for measuring and displaying time.
unicode	Package unicode provides data and functions to test some properties of Unicode code points.
utf16	Package utf16 implements encoding and decoding of UTF-16 sequences.
utf8	Package utf8 implements functions and constants to support text encoded in UTF-8.
unsafe	Package unsafe contains operations that step around the type safety of Go programs.

---

## 14.6 Go CMD

### 14.6.1 CMD

In python we have the CMD5 package that allows us to create command shells with plugins. In Go we find a community developed package called `gosh` (or Go shell). It uses the Go plugin system to create interactive console-based shell

programs. A shell created with `gosh` contains a collection of Go plugins, each of which implement one or more commands. Upon start `gosh` starts, searches the directory `./plugins` and loads them so they become available within `gosh`.

- <https://github.com/vladimirvivien/gosh>

## 14.6.2 DocOpts

When we want to design commandline arguments for go programs we have many options. However, as our approach is to create documentation first, `docopts` provides also a good approach for Go. The code for it is located at

- <https://github.com/docopt/docopt.go>

It can be installed with

```
$ go get github.com/docopt/docopt-go
```

A sample programs are located at

- <https://github.com/docopt/docopt.go/blob/master/examples/options/>

A sample program of using doc opts for our purposes looks as follows.

```
package main

import (
 "fmt"
 "github.com/docopt/docopt-go"
)

func main() {
 usage := `cm-go.

Usage:
 cm-go vm start NAME [--cloud=CLOUD]
 cm-go vm stop NAME [--cloud=CLOUD]
 cm-go set --cloud=CLOUD
 cm-go -h | --help
 cm-go --version

Options:
 -h --help Show this screen.
 --version Show version.
 --cloud=CLOUD The name of the cloud.
 --moored Moored (anchored) mine.
 --drifting Drifting mine.

ARGUMENTS:
 NAME The name of the VM`

 arguments, _ := docopt.ParseDoc(usage)
 fmt.Println(arguments)
}
```

## 14.7 Go REST

---

Go is a new powerful language and there are many frameworks from lightweight to full featured that support building RESTful APIs.

1. [Revel](#) A high-productivity web framework for the Go language.
2. [Gin](#) The fastest full-featured web framework for Golang. Crystal clear.
3. [Martini](#) Classy web framework for Go
4. [Web.go](#) The easiest way to create web applications with Go

List here the rest services tutorials for frameworks

- <https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/>
- with mongo <https://hackernoon.com/build-restful-api-in-go-and-mongodb-5e7f2ec4be94>
- <https://tutorialedge.net/golang/consuming-restful-api-with-go/>
- <https://thenewstack.io/make-a-restful-json-api-go/>
- [Making a RESTful JSON API in Go](#)

### 14.7.1 Gorilla

- <https://www.codementor.io/codehakase/building-a-restful-api-with-golang-a6yivzqdo>

Gorilla is a web toolkit for the Go programming language. Currently these packages are available:

- gorilla/context stores global request variables.
- gorilla/mux is a powerful URL router and dispatcher.
- gorilla/reverse produces reversible regular expressions for regexp-based muxes.
- gorilla/rpc implements RPC over HTTP with codec for JSON-RPC.
- gorilla/schema converts form values to a struct.
- gorilla/securecookie encodes and decodes authenticated and optionally encrypted cookie values.

- gorilla/sessions saves cookie and filesystem sessions and allows custom session backends.
- gorilla/websocket implements the WebSocket protocol defined in RFC 6455.

## 14.7.2 REST, RESTful

REST is an acronym for Representational State Transfer. It is a web standards architecture and HTTP Protocol. The REST protocol, describes six (6) constraints:

1. Uniform Interface
2. Cacheable
3. Client-Server
4. Stateless
5. Code on Demand
6. Layered System

## 14.7.3 Router

Package gorilla/mux implements a request router and dispatcher for matching incoming requests to their respective handler.

The name mux stands for “HTTP request multiplexer”. Like the standard `http.ServeMux`, `mux.Router` matches incoming requests against a list of registered routes and calls a handler for the route that matches the URL or other conditions. The main features are:

We’ll need to use a mux to route requests, so we need a Go package for that (mux stands for HTTP request multiplexer which matches an incoming request to against a list of routes (registered)). In the rest-api directory, let’s require the dependency (package rather). More examples are here: <https://github.com/gorilla/mux#examples>

```
rest-api$ go get github.com/gorilla/mux
```

```
package main

import (
 "encoding/json"
 "log"
 "net/http"
```

```

 "github.com/gorilla/mux"
)

// our main function
func main() {
 router := mux.NewRouter()
 router.HandleFunc("/people", GetPeople).Methods("GET")
 router.HandleFunc("/people/{id}", GetPerson).Methods("GET")
 router.HandleFunc("/people/{id}", CreatePerson).Methods("POST")
 router.HandleFunc("/people/{id}", DeletePerson).Methods("DELETE")
 log.Fatal(http.ListenAndServe(":8000", router))
}

```

Packages are explained here: \* `fmt` is what we will be using to print to STDOUT (the console) \* `log` is used to log when the server exits \* `encoding/json` is for creating our JSON responses \* `net/http` will give us the representations of HTTP requests, responses, and be responsible for running our server \* `github.com/gorilla/mux` will be our router that will take requests and decide what should be done with them

## 14.7.4 Full code

```

package main

import (
 "encoding/json"
 "github.com/gorilla/mux"
 "log"
 "net/http"
)

// The person Type (more like an object)
type Person struct {
 ID string `json:"id,omitempty"`
 Firstname string `json:"firstname,omitempty"`
 Lastname string `json:"lastname,omitempty"`
 Address *Address `json:"address,omitempty"`
}

type Address struct {
 City string `json:"city,omitempty"`
 State string `json:"state,omitempty"`
}

var people []Person

// Display all from the people var
func GetPeople(w http.ResponseWriter, r *http.Request) {
 json.NewEncoder(w).Encode(people)
}

// Display a single data
func GetPerson(w http.ResponseWriter, r *http.Request) {
 params := mux.Vars(r)
 for _, item := range people {
 if item.ID == params["id"] {
 json.NewEncoder(w).Encode(item)
 return
 }
 }
 json.NewEncoder(w).Encode(&Person{})
}

// create a new item
func CreatePerson(w http.ResponseWriter, r *http.Request) {
 params := mux.Vars(r)
 var person Person
 _ = json.NewDecoder(r.Body).Decode(&person)
 person.ID = params["id"]
}

```

```

 people = append(people, person)
 json.NewEncoder(w).Encode(people)
}

// Delete an item
func DeletePerson(w http.ResponseWriter, r *http.Request) {
 params := mux.Vars(r)
 for index, item := range people {
 if item.ID == params["id"] {
 people = append(people[:index], people[index+1:]...)
 break
 }
 }
 json.NewEncoder(w).Encode(people)
}

// main function to boot up everything
func main() {
 router := mux.NewRouter()
 people = append(people, Person{ID: "1", Firstname: "John", Lastname: "Doe", Address: &Address{City: "City X", State: "State X"}},
 people = append(people, Person{ID: "2", Firstname: "Koko", Lastname: "Doe", Address: &Address{City: "City Z", State: "State Z"}},
 router.HandleFunc("/people", GetPeople).Methods("GET")
 router.HandleFunc("/people/{id}", GetPerson).Methods("GET")
 router.HandleFunc("/people/{id}", CreatePerson).Methods("POST")
 router.HandleFunc("/people/{id}", DeletePerson).Methods("DELETE")
 log.Fatal(http.ListenAndServe(":8000", router))
}

```

## 14.8 OPEN API

We have a large section previously on openapi, what needs to be done here is to showcase how to generate go from swagger codegen or other tool and use it. Please see [Section 6.6](#)

In this section, we introduce the `go-swagger`, which is an open source implementation for Swagger 2.0 (aka OpenAPI 2.0). Please follow this link for more details: <https://goswagger.io/>.

### 14.8.1 Install from Homebrew

The binary release version can be installed via Homebrew on macOS.

```

brew tap go-swagger/go-swagger
brew install go-swagger

```

### 14.8.2 serve specification UI

Most basic use-case: serve a UI for your spec:

```

swagger serve https://raw.githubusercontent.com/swagger-api/swagger-spec/master/examples/v2.0/json/petstore-expanded.json

```

### 14.8.3 validate a specification

```
swagger validate https://raw.githubusercontent.com/swagger-api/swagger-spec/master/examples/v2.0/json/petstore-expanded.j
```

This command should produce this content:

```
The swagger spec at "https://raw.githubusercontent.com/swagger-api/swagger-spec/master/examples/v2.0/json/petstore-expanc
```

## 14.8.4 Generate a Go OpenAPI server

```
swagger generate server [-f ./swagger.json] -A [application-name [--principal [principal-name]]]
```

## 14.8.5 generate a Go OpenAPI client

```
swagger generate client [-f ./swagger.json] -A [application-name [--principal [principal-name]]]
```

## 14.8.6 generate a spec from the source

```
swagger generate spec -o ./swagger.json
```

## 14.8.7 generate a data model

```
swagger generate model --spec={spec}
```

## 14.8.8 other editors

- [KaiZen-OpenAPI-Editor](#) - Full-featured Eclipse editor for OpenAPI 2.0 and 3.0, also available on Eclipse Marketplace.
- [Atom/linter-swagger](#) - This plugin for Atom Linter will lint Swagger 2.0 specifications or OpenAPI 3.0, both JSON and YAML using swagger-parser node package.
- [Swagger Editor](#) - Design, describe, and document your API on the first open source editor fully dedicated to OpenAPI-based APIs.
- [RepreZen API Studio](#) - RepreZen API Studio is an integrated workbench that brings API-first design into focus for your whole team, harmonizes your API designs, and generates APIs that click into client apps.
- [Apicurio Studio](#) A standalone API design studio that can be used to create new or edit existing API designs.
- [SwaggerHub](#) - API design and documentation platform to improve collaboration, standardize development workflow and centralize their API discovery and consumption.

- [Senya Editor](#) - Design API specifications fast and effectively in your favorite JetBrains IDE.

## 14.9 CREATE AN ECHO SERVICE USING SWAGGER AND GO

---

In this tutorial, we will create a micro service using Swagger and Go. This service does nothing but echos the message sent from users.

### 14.9.1 Dependencies

Some dependencies are required to install before proceeding.

- [github.com/go-swagger/go-swagger/cmd/swagger](https://github.com/go-swagger/go-swagger/cmd/swagger)
- [github.com/go-openapi/runtime](https://github.com/go-openapi/runtime)
- [github.com/docker/go-units](https://github.com/docker/go-units)
- [github.com/go-openapi/loads](https://github.com/go-openapi/loads)
- [github.com/go-openapi/validate](https://github.com/go-openapi/validate)

These dependencies can be installed via command lines:

```
go get -u -v github.com/go-swagger/go-swagger/cmd/swagger
go get -u -v github.com/go-openapi/runtime
go get -u -v github.com/docker/go-units
go get -u -v github.com/go-openapi/loads
go get -u -v github.com/go-openapi/validate
```

### 14.9.2 Initialize a Golang project

Create a new folder named `hello-swagger` under `~/go/src`, and a folder named `swagger` under `hello-swagger`. Create `main.go` under `hello-swagger` and `swagger.yml` under `swagger` folder. The structure of the project should look like this:

```
hello-swagger/
 swagger/
 swagger.yml
 main.go
```

### 14.9.3 Define APIs and generate code in Go

Here is the code of `swagger.yml`:

```
swagger: "2.0"
info:
 title: "Echo"
```

```

version: "0.0.1"
paths:
 /echo:
 get:
 operationId: echo
 produces:
 - "application/json"
 parameters:
 - name: "msg"
 in: "query"
 required: true
 type: "string"
 responses:
 200:
 description: "echo message"
 schema:
 type: object
 properties:
 msg:
 type: string

```

To generate Go code run this command:

```
~/go/bin/swagger generate server --target ./swagger --spec ./swagger/swagger.yml --exclude-main --name=echo
```

The command will generate Go code and put them under the `swagger` folder. Once there is a new folder named `restapi`, this step is successful.

## 14.9.4 Implement the functionality

Now we can create our `restapi` server and implement the request handler in Go.

Modify the `main.go` so the the content looks like this:

```

package main

import (
 "github.com/go-openapi/loads"
 "github.com/go-openapi/runtime/middleware"
 "hello-swagger/swagger/restapi"
 "hello-swagger/swagger/restapi/operations"
 "log"
)

func main() {
 log.Println("Starting...")

 swaggerSpec, err := loads.Analyzed(restapi.SwaggerJSON, "")

 if err != nil {
 log.Fatalln(err)
 }

 api := operations.NewEchoAPI(swaggerSpec)
 server := restapi.NewServer(api)
 defer server.Shutdown()

 server.Port = 8080

 api.EchoHandler = operations.EchoHandlerFunc(
 func(params operations.EchoParams) middleware.Responder {
 response := params.Msg
 payload := operations.EchoOKBody{Msg: response}
 return operations.NewEchoOK().WithPayload(&payload)
 })

```

```
 if err := server.Serve(); err != nil {
 log.Fatalf(err)
 }
}
```

## 14.9.5 Run and test the server

We use `go` command to run, which compile the code and run the program. Once the program is been started, some logs are print out in the console like:

```
2019/02/04 17:06:24 Starting...
2019/02/04 17:06:24 Serving echo at http://[::]:8080
```

Once the server is listening at 8080, we can run `curl` command to do some tests:

```
curl http://localhost:8080/echo?msg=Hello
{"msg":"Hello"}

curl http://localhost:8080/echo?msg=World
{"msg":"World"}
```

## 14.9.6 References

- [go-swagger documentation](#)
- [OpenAPI.Tools](#)

## 14.10 Go CLOUD

---

### 14.10.1 Golang Openstack Client

- <https://github.com/openstack/golang-client>
- [Authentication](#)
- [Images](#)
- [ObjectStore](#)
- This file reads in some configurations form a json file, however we want to develop one for our `~/cloudmesh/cloudmesh.yaml` file. For the json example see: [json setup](#)
- [Volume](#)

Portable Cloud Programming with Go Cloud:

- <https://blog.golang.org/go-cloud>

## 14.10.2 OpenStack from Go

There are multiple API interfaces that allow direct access to elementary functionality to control for example virtual machines in Go. This includes GopherCloud, and GolangClient. We describe them next.

### 14.10.2.1 GopherCloud

GopherCloud is located at

- <https://github.com/gophercloud/gophercloud>

#### 14.10.2.1.1 Authentication

To interact with OpenStack, you will need to first authenticate with your OpenStack cloud. You will need to know your username and password. However the example that we provide here is on intention wrong to showcase you a better way. The example includes a hard coded username and password, that actually is supposed to be either read in via an interactive process or from a `~/cloudmesh/cloudmesh.yaml` file as we have used in our python cloudmesh code. We will use the same format and obtain the information from that file. Your task will be to write a yaml file reader in go, get the information and modify the program accordingly while improving this section with a pull request.

The example copied from gopher cloud looks as follows:

```
import (
 "github.com/gophercloud/gophercloud"
 "github.com/gophercloud/gophercloud/openstack"
 "github.com/gophercloud/gophercloud/openstack/utils"
)

opts := gophercloud.AuthOptions{
 IdentityEndpoint: "https://openstack.example.com:5000/v2.0",
 Username: "{username}",
 Password: "{password}",
}
```

Naturally you can also obtain the values from environment variables as pointed out by gopher cloud:

```
import (
 "github.com/gophercloud/gophercloud"
```

```

 "github.com/gophercloud/gophercloud/openstack"
 "github.com/gophercloud/gophercloud/openstack/utils"
)

opts, err := openstack.AuthOptionsFromEnv()
provider, err := openstack.AuthenticatedClient(opts)

```

We will at this time not use the second approach.

To start a virtual machine you need to first identify the location of the client for the region you will use. This can be achieved with the command:

```

client, err := openstack.NewComputeV2(provider, gophercloud.EndpointOpts{
 Region: os.Getenv("OS_REGION_NAME"),
})

```

#### 14.10.2.1.2 Virtual machines

Now that we know we can authenticate to the cloud, we can create our first virtual machine, where `flavor_id` and `image_id` are the appropriate flavors and image ids:

```

import "github.com/gophercloud/gophercloud/openstack/compute/v2/servers"

server, err := servers.Create(client, servers.CreateOpts{
 Name: "gregor-001",
 FlavorRef: "flavor_id",
 ImageRef: "image_id",
}).Extract()

```

Additional information can be found in the source code of GoherClient which you can easily inspect. Some useful documentation is also provided in <https://github.com/gophercloud/gophercloud/blob/master/doc.go>

#### 14.10.2.1.3 Resources

Code examples are provided from <https://github.com/gophercloud/gophercloud/blob/master/doc.go>

As Openstack is providing REST interfaces, gopher cloud leverages this model. Hence, it provides interfaces to manage REST resources. These resources are bound to structs so they can easily be manipulated and interfaced with. To for example get the client with a specific `{serverId}` and extract its information we can use the following API call:

```

server, err := servers.Get(client, "{serverId}").Extract()

```

If we need just a subset of the information, we can get an intermediate result with just the get method. Then we can obtain specific information from the result

as needed.

```
result := servers.Get(client, "{serverId}")
// Attempt to extract the disk configuration from the OS-DCF disk config
// extension:
config, err := diskconfig.ExtractGet(result)
```

The previous example is based on a single resource. However, if we interact with a list of resources we need to use the `Page` struct so we can iterate over each page. A convenient example is provided next. Here we list all servers while iterating over all pages returned to us. While calling each page we can invoke special operations that are applied to each page.

```
err := servers.List(client, nil).EachPage(func (page pagination.Page) (bool, error) {
 s, err := servers.ExtractServers(page)
 if err != nil {
 return false, err
 }
 // Handle the []servers.Server slice.
 // Return "false" or an error to prematurely stop fetching new pages.
 return true, nil
})
```

However, if we just want to provide a list of all servers, we can simply use the `AllPages()` method as follows:

```
allPages, err := servers.List(client, nil).AllPages()
allServers, err := servers.ExtractServers(allPages)
```

Additional methods and resources can be found at

- <https://github.com/gophercloud/gophercloud/blob/master/doc.go>

## 14.11 GO LINKS

---

In this section we list some potentially useful as well as links to research Go further. If you find other very useful links, please let us know. YOU can share this section with us.

### 14.11.1 Introductory Material

Some Introductory Material about go can be found at

- [https://cse.sc.edu/~mgv/csce330f16/pres/330f15\\_BarnhartReevesLee\\_Go.pdf](https://cse.sc.edu/~mgv/csce330f16/pres/330f15_BarnhartReevesLee_Go.pdf)
- [http://www.cis.upenn.edu/~matuszek/cis554-2016/Talks/Golang\\_Presentation.ppt](http://www.cis.upenn.edu/~matuszek/cis554-2016/Talks/Golang_Presentation.ppt)

- <https://talks.golang.org/2015/go-for-java-programmers.slide#1>
- <https://github.com/golang/go/wiki/GoTalks>
- <http://courses.cs.vt.edu/cs5204/fall11-kafura/Overheads/Go.pptx>
- [https://en.wikipedia.org/wiki/Kahn\\_process\\_networks](https://en.wikipedia.org/wiki/Kahn_process_networks)

## 14.11.2 The GO Language

Information that deals with describing the Go language are

- <http://devcodegeek.com/best-cloud-programming-languages.html>
- <https://webdesignledger.com/top-4-cloud-computing-languages-learn-now/>
- <https://techlog360.com/top-10-cloud-programming-languages/>
- <https://www.techrepublic.com/article/10-of-the-coolest-cloud-programming-languages/>

## 14.11.3 How popular is Go?

Often we find references to how popular a programming language is. Some ways of identifying this is with analysis of the Tiobe index

- <http://www.zdnet.com/article/which-programming-languages-are-most-popular-and-what-does-that-even-mean/>
- <https://www.tiobe.com/tiobe-index/>

Another way is to look at e the PYPL Popularity of Programming Language

*“The PYPL Popularity of Programming Language Index is created by analyzing how often language tutorials are searched on Google. The more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from Google Trends.”*

- <http://pypl.github.io/PYPL.html>
- <https://www.infoworld.com/article/2610933/cloud-computing/article.html>

## 14.11.4 OpenAPI and Go

The following links are useful to research aspects related to creating REST

services or clients form specifications in Go.

- <https://swagger.io/>
- <https://nordicapis.com/top-specification-formats-for-rest-apis/>
- <https://www.npmjs.com/package/raml-python>
- <https://github.com/Jumpscale/go-raml>
- [https://en.wikipedia.org/wiki/Overview\\_of\\_RESTful\\_API\\_Description\\_Lan](https://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Lan)

## 14.12 EXERCISES

---

E.Go.1:

*Write a REST service with OpenAPI that exposes the cpu load*

E.GO.2:

*Write a REST service with Gorilla that exposes the cpu load*

E.GO.3:

*Locate goo Go Libraries for writing rest services.*

E.GO.4:

*Write an MQTT service in Go.*

## 15 REFERENCES



- [1] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, “Accessing Multiple Clouds with Cloudmesh,” in *Proceedings of the 2014 acm international workshop on software-defined ecosystems*, 2014, p. 8 [Online]. Available: <https://github.com/cyberaide/paper-cloudmesh/raw/master/vonLaszewski-cloudmesh.pdf>
- [2] domo.com, “Data never sleeps 7.0.” Image, Jun-2019 [Online]. Available: <https://www.domo.com/learn/data-never-sleeps-7>
- [3] domo.com, “Data never sleeps 6.0.” Image, Jun-2018 [Online]. Available: <https://www.domo.com/blog/wp-content/uploads/2018/06/18-domo-data-never-sleeps-6.png>
- [4] L. Lewis, “This is what happens in an internet minute.” Web Page, Apr-2018 [Online]. Available: <https://www.allaccess.com/merge/archive/28030/2018-update-what-happens-in-an-internet-minute>
- [5] visibletechnologies.com, “Big data 36 month.” Image, Mar-2012 [Online]. Available: [https://blogs-images.forbes.com/christopherfrank/files/2012/03/VI\\_BigData\\_Graphic\\_v3\\_low](https://blogs-images.forbes.com/christopherfrank/files/2012/03/VI_BigData_Graphic_v3_low)
- [6] K. Heslin, “Proper data center staffing is key to reliable operations.” Web Page, Mar-2015 [Online]. Available: <https://journal.uptimeinstitute.com/data-center-staffing/>
- [7] D. Bouley, “Estimating a data center’s electrical carbon footprint,” Schneider Electric – Data Center Science Center, Report 66, 2011 [Online]. Available: [http://www.apc.com/salestools/DBOY-7EVHLH/DBOY-7EVHLH\\_R0\\_EN.pdf](http://www.apc.com/salestools/DBOY-7EVHLH/DBOY-7EVHLH_R0_EN.pdf)
- [8] Indiana State, “Indiana - state energy profile overview - u.s. Energy information.” Web Page, Apr-2018 [Online]. Available: <https://www.eia.gov/state/?sid=IN>

- [9] Joe Powell and Associates, “4 steps to better data center cooling.” Web Page, Jan-2019 [Online]. Available: <https://www.joepowell.com/4-steps-to-better-data-center-cooling/>
- [10] Kevin Normandeau, “Approaches to data center containment.” Web Page, Nov-2012 [Online]. Available: <https://www.joepowell.com/4-steps-to-better-data-center-cooling/>
- [11] T. R. Furlani *et al.*, “Using xdmmod to facilitate xsede operations, planning and analysis,” in *Proceedings of the conference on extreme science and engineering discovery environment: Gateway to discovery*, 2013, p. 8 [Online]. Available: <http://doi.acm.org/10.1145/2484762.2484763>
- [12] F. Wang, G. von Laszewski, G. C. Fox, T. R. Furlani, R. L. DeLeon, and S. M. Gallo, “Towards a scientific impact measuring framework for large computing facilities - a case study on xsede,” in *Proceedings of the 2014 annual conference on extreme science and engineering discovery environment*, 2014, p. 8 [Online]. Available: <http://cgl.soic.indiana.edu/publications/Metrics2014.pdf>
- [13] G. von Laszewski, “FutureGrid cloud metrics.” Web Page, Sep-2014 [Online]. Available: <http://archive.futuregrid.org/metrics/html/results/2014-Q3/reports/rst/india-All.html>
- [14] Amazon, “Amazon data centers.” Web Page, Jan-2019 [Online]. Available: <https://aws.amazon.com/compliance/data-center/data-centers/>
- [15] Amazon, “AWS global infrastructure.” Web Page, Jan-2019 [Online]. Available: <https://aws.amazon.com/about-aws/global-infrastructure/>
- [16] Microsoft, “Azure regions.” Web Page, Jan-2019 [Online]. Available: <https://azure.microsoft.com/en-us/global-infrastructure/regions/>
- [17] Google, “Google locations.” Web Page, Jan-2019 [Online]. Available: <https://www.google.com/about/datacenters/inside/locations/index.html>
- [18] Google, “Efficiency: How we do it.” Web Page, Jan-2019 [Online]. Available: <https://www.google.com/about/datacenters/efficiency/internal/>
- [19] A. Shehabi *et al.*, “United states data center energy usage report,” Lawrence

Berkeley National Laboratory, Report DE-AC02-05CH1131, LBNL-1005775, Jun. 2016 [Online]. Available: <https://escholarship.org/uc/item/84p772fc>

[20] Microsoft, “Microsoft leona philpot.” Web Page, Aug-2019 [Online]. Available: <https://news.microsoft.com/features/microsoft-research-project-puts-cloud-in-ocean-for-the-first-time/>

[21] Microsoft, “Microsoft northern isles.” Web Page, Aug-2019 [Online]. Available: <https://news.microsoft.com/features/under-the-sea-microsoft-tests-a-datacenter-thats-quick-to-deploy-could-provide-internet-connectivity-for-years/>

[22] New York times, “Microsoft underwater datacenter.” Web Page, Aug-2019 [Online]. Available: <https://www.nytimes.com/2016/02/01/technology/microsoft-plumbs-oceans-depths-to-test-underwater-data-center.html>

[23] I. Foster and C. Kesselman, Eds., *The grid: Blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[24] S. Tata, “Cloud platforms: Concepts, definitions, architectures and open issues.” Presentation, Nov-2012 [Online]. Available: <http://www.lifl.fr/iwaise12/presentations/tata.pdf>

[25] J. Varia, “Architecting for the cloud: Best practices,” *Amazon Web Services*, vol. 1, pp. 1–21, 2010.

[26] I. Sun Microsystems, “Introduction to cloud computing architecture.” White Paper, Jun-2009 [Online]. Available: [http://staff.polito.it/alessandro.mantelero/cloud\\_computing/Sun\\_Wp2009.pdf](http://staff.polito.it/alessandro.mantelero/cloud_computing/Sun_Wp2009.pdf)

[27] G. Kaefer, “Cloud computing architecture.” Presentation, May-2010 [Online]. Available: [https://resources.sei.cmu.edu/asset\\_files/Presentation/2010\\_017\\_001\\_23337.pdf](https://resources.sei.cmu.edu/asset_files/Presentation/2010_017_001_23337.pdf)

[28] O. Orgeron, “Top 10 list for success in the cloud.” Oracle Corporation; Presentation, 2012 [Online]. Available: <https://www.oracle.com/technetwork/articles/entarch/orgeron-top-10-cloud-1957407.pdf>

- [29] A. K. Anbarasu, "Cloud reference architecture." Oracle Corporation; White Paper, Nov-2012 [Online]. Available: <https://www.oracle.com/technetwork/topics/entarch/oracle-wp-cloud-ref-arch-1883533.pdf>
- [30] "Topic : Cloud computing architecture." Presentation [Online]. Available: <https://pdfs.semanticscholar.org/cecd/c193b73ec1e7b42d132b3c340e6dd348d3f4>
- [31] National Institute of Standards, "NIST big data public working group." Aug-2019 [Online]. Available: <https://bigdatawg.nist.gov/>
- [32] LIGO, "Ligo data grid." Sep-2019 [Online]. Available: <https://www.lsc-group.phys.uwm.edu/lscdatagrid/overview.html>
- [33] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Doctoral dissertation, 2000.
- [34] Wikipedia, "Representational state transfer." Web Page, 2019 [Online]. Available: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)
- [35] OpenAPI Initiative, "The openapi specification." Web Page [Online]. Available: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>
- [36] OpenAPI Initiative, "The openapi specification." Web Page [Online]. Available: <https://github.com/OAI/OpenAPI-Specification>
- [37] RAML, "RAML version 1.0: RESTful api modeling language." Web Page [Online]. Available: <https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md>
- [38] R. H. Kevin Burke Kyle Conroy, "Flask-restful." Web Page [Online]. Available: <https://flask-restful.readthedocs.io/en/latest/>
- [39] E. O. Ltd, "Django rest framework." Web Page [Online]. Available: <https://www.django-rest-framework.org/>

- [40] S. Software, “API development for everyone.” Web Page [Online]. Available: <https://swagger.io>
- [41] S. Software, “Swagger codegen documentation.” Web Page [Online]. Available: <https://swagger.io/docs/open-source-tools/swagger-codegen/>
- [42] A. Y. W. Hate, “OpenAPI.Tools.” Web Page [Online]. Available: <https://openapi.tools/>
- [43] tinyspec, “Tinyspec.” Web Page [Online]. Available: <https://github.com/Ajaxy/tinyspec>
- [44] api blueprint, “API blueprint. A powerful high-level api description language for web apis.” Web Page [Online]. Available: <https://apiblueprint.org/>
- [45] OpenAPI Initiative, “Announcing the official release of openapi 3.0.” Web Page, 2017 [Online]. Available: <https://swagger.io/blog/news/announcing-openapi-3-0/>
- [46] OpenAPI Initiative, “The openapi docs.” Web Page [Online]. Available: <https://swagger.io/docs/specification/about/>
- [47] S. Software, “Swagger editor documentation.” Web Page [Online]. Available: <https://swagger.io/docs/open-source-tools/swagger-editor/>
- [48] S. Software, “Swagger ui.” Web Page [Online]. Available: <https://swagger.io/docs/open-source-tools/swagger-ui/usage/installation/>
- [49] RAML, “RAML.” Web Page [Online]. Available: <https://raml.org/>
- [50] Yehuda Katz, “JSON:API.” Web Page [Online]. Available: <https://jsonapi.org/>
- [51] Zalando SE, “Connexion.” Web Page [Online]. Available: <https://github.com/zalando/connexion>
- [52] Wikipedia, “Scikit-learn.” Web Page, Aug-2019 [Online]. Available: <https://en.wikipedia.org/wiki/Scikit-learn>

- [53] scikit-learn developers, Web Page [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
- [54] Facebook, “Introduction to graphql.” Web Page, Aug-2019 [Online]. Available: <https://graphql.org/learn/>
- [55] Django Software Foundation, “Django web framework.” Web Page, Aug-2019 [Online]. Available: <https://www.djangoproject.com/>
- [56] João Angelo, “JWT tokens.” Web Page, Aug-2019 [Online]. Available: <https://stackoverflow.com/a/39914013>
- [57] Clay Allsopp, “GraphQL and authentication.” Web Page, Aug-2019 [Online]. Available: <https://medium.com/the-graphqlhub/graphql-and-authentication-b73aed34bbeb>
- [58] Github, “Github api v4.” Web Page, Aug-2019 [Online]. Available: <https://developer.github.com/v4/>
- [59] Jonatas Baldin, “Introduction to graphql python implementation.” Web Page, Aug-2019 [Online]. Available: <https://www.howtographql.com/graphql-python/0-introduction/>
- [60] S. J. Bigelow, “Full virtualization vs. Paravirtualization: What are the key differences?” Web page, Sep-2018 [Online]. Available: <https://searchservirtualization.techtarget.com/answer/Full-virtualization-vs-paravirtualization-What-are-the-key-differences>
- [61] Wikipedia, “Input–output memory management unit.” Web Page, May-2019 [Online]. Available: [https://en.wikipedia.org/wiki/Input%E2%80%93output\\_memory\\_management\\_u](https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit)
- [62] Wikipedia, “Input–output memory management unit.” Web Page, Jul-2019 [Online]. Available: [https://en.wikipedia.org/wiki/X86\\_virtualization#I.2FO\\_MMU\\_virtualization\\_.28Vi\\_and\\_Intel\\_VT-d.29](https://en.wikipedia.org/wiki/X86_virtualization#I.2FO_MMU_virtualization_.28Vi_and_Intel_VT-d.29)
- [63] Suse, “Virtualization with kvm,” in *Virtualization with kvm*, Suse, 2016 [Online]. Available:

[https://www.suse.com/documentation/sles11/book\\_kvm/data/cha\\_libvirt\\_overvie](https://www.suse.com/documentation/sles11/book_kvm/data/cha_libvirt_overvie)

[64] J. Guerrag, “Difference between kvm and qemu.” Forum Post, Dec-2010 [Online]. Available: <https://serverfault.com/questions/208693/difference-between-kvm-and-qemu>

[65] Wikipedia, “VMWare.” Web Page, Sep-2018 [Online]. Available: <https://en.wikipedia.org/wiki/VMware>

[66] “Wine – wine is not an emulator.” Web Page, Sep-2018 [Online]. Available: <https://www.winehq.org/>

[67] Stackoverflow, “What are the differences between qemu and virtualbox?” Forum Post, Sep-2018 [Online]. Available: <https://stackoverflow.com/questions/43704856/what-are-the-differences-between-qemu-and-virtualbox>

[68] “Hadoop mapreduce.” Aug-2019 [Online]. Available: [https://www.edureka.co/blog/mapreduce-tutorial/?utm\\_source=youtube&utm\\_campaign=mapreduce-tutorial-161216-wr&utm\\_medium=description](https://www.edureka.co/blog/mapreduce-tutorial/?utm_source=youtube&utm_campaign=mapreduce-tutorial-161216-wr&utm_medium=description)

[69] “Hadoop mapreduce.” Aug-2019 [Online]. Available: <https://www.youtube.com/watch?v=SqvAaB3vK8U&list=WL&index=25&t=2547s>

[70] “Apache mapreduce.” Aug-2019 [Online]. Available: <https://www.ibm.com/analytics/hadoop/mapreduce>

[71] Wikipedia, “MapReduce.” Aug-2019 [Online]. Available: <https://en.wikipedia.org/wiki/MapReduce>

[72] “Hadoop mapreduce.” Aug-2019 [Online]. Available: [https://www.tutorialspoint.com/hadoop/hadoop\\_mapreduce.htm](https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm)

[73] A. Khan, “Hadoop and spark.” Aug-2019 [Online]. Available: <https://www.quora.com/What-is-the-difference-between-Hadoop-and-Spark>. [Accessed: 03-Sep-2017]

[74] “Apache spark vs hadoop mapreduce.” Aug-2019 [Online]. Available: <https://data-flair.training/blogs/apache-spark-vs-hadoop-mapreduce/>

[75] “HDFS architecture guide.” Aug-2019 [Online]. Available: [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)

[76] “Amazon emr - amazon web services.” Aug-2019 [Online]. Available: [https://aws.amazon.com/emr/?nc2=type\\_a](https://aws.amazon.com/emr/?nc2=type_a)

[77] AWS, “AWS.” Web Page, Aug-2019 [Online]. Available: <https://us-east-2.console.aws.amazon.com/elasticmapreduce/home?region=us-east-2#>

[78] Twister, “Twister2: Twister2 Big Data Hosting Environment: A composable framework for high-performance data analytics.” Web Page, Feb-2017 [Online]. Available: <https://twister2.gitbook.io/twister2/>

[79] Twister, “Twister2: Twister2 Big Data Hosting Environment: A composable framework for high-performance data analytics.” Web Page, Feb-2017 [Online]. Available: <https://github.com/DSC-SPIDAL/twister2/>

[80] Twister, “Twister2 word count example.” Aug-2019.

[81] Twister, “Task examples.” Web Page, Feb-2017 [Online]. Available: [https://twister2.gitbook.io/twister2/examples/task\\_examples](https://twister2.gitbook.io/twister2/examples/task_examples)

[82] Twister, “Communication Model.” Web Page, Feb-2017 [Online]. Available: <https://twister2.gitbook.io/twister2/concepts/communication/communication-model>

[83] S. Kamburugamuve *et al.*, “Twister: Net-communication library for big data processing in hpc and cloud environments,” in *2018 IEEE 11th International Conference on Cloud Computing (Cloud)*, 2018, pp. 383–391.

[84] Twister2, “Kmeans performance comparison.” Web Page, Jan-2019 [Online]. Available: <https://twister2.gitbook.io/twister2/>

[85] Twister, “Twister Examples.” Web Page, Feb-2017 [Online]. Available: <https://twister2.gitbook.io/twister2/examples>

[86] Docker, “Overview of docker hub.” Web Page, Mar-2018 [Online]. Available: <https://docs.docker.com/docker-hub/>

[87] S. Bhartiya, “How to use dockerhub.” Blog, Jan-2018 [Online]. Available: <https://www.linux.com/blog/learn/intro-to-linux/2018/1/how-use-dockerhub>

[88] Docker, “Repositories on docker hub.” Web Page, Mar-2018 [Online]. Available: <https://docs.docker.com/docker-hub/repos/>

[89] R. Irani, “Docker tutorial series-part 4-docker hub.” Blog, Jul-2015 [Online]. Available: <https://rominirani.com/docker-tutorial-series-part-4-docker-hub-b51fb545dd8e>

[90] G. M. Kurtzer, “Singularity Containers for Science.” Presentation, Jan-2019 [Online]. Available: [http://www.hpcadvisorycouncil.com/events/2017/stanford-workshop/pdf/GMKurtzer\\_Singularity\\_Keynote\\_Tuesday\\_02072017.pdf#43](http://www.hpcadvisorycouncil.com/events/2017/stanford-workshop/pdf/GMKurtzer_Singularity_Keynote_Tuesday_02072017.pdf#43)

[91] A. Ellis, “Introducing functions as a service (openfaas).” Web Page, Aug-2017 [Online]. Available: <https://blog.alexellis.io/introducing-functions-as-a-service/>

[92] P. Caponetti, “Why mqtt is the protocol of choice for the iot.” Blog, Nov-2017 [Online]. Available: <http://blog.xively.com/why-mqtt-is-the-protocol-of-choice-for-the-iot/>

[93] The HiveMQ Team, “Introducing the mqtt protocol - mqtt essentials: Part 1.” Web Page, Nov-2017 [Online]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt>

[94] Wikipedia, “MQTT.” Web Page, Nov-2017 [Online]. Available: <https://en.wikipedia.org/w/index.php?title=MQTT&oldid=808683219>

[95] Mqtt, “Mqtt official website.” mqtt official website, Nov-2017 [Online]. Available: <http://mqtt.org/>

[96] “Mosquito mqtt broker.” Web Page, Nov-2017 [Online]. Available: <https://mosquitto.org/>

[97] Random nerds tutorial, “What is mqtt and how it works.” Web Page, Nov-

2017 [Online]. Available: <https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/>

[98] H. MQ, "MQTT essentials part 2: Publish & subscribe." Web Page, Nov-2017 [Online]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe>

[99] E. Paho, "Python client - documentation." Web Page, Nov-2017 [Online]. Available: <https://www.eclipse.org/paho/clients/python/docs/>

[100] H. MQ, "MQTT essentials part 6: Quality of service 0, 1 and 2." Web Page, Nov-2017 [Online]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>

[101] T. Ouska, "Transport-level security tradeoffs using mqtt." Web Page, Nov-2017 [Online]. Available: <http://iotdesign.embedded-computing.com/guest-blogs/transport-level-security-tradeoffs-using-mqtt/>

[102] H. Mq, "MQTT security fundamentals: TLS / ssl." Web Page, Nov-2017 [Online]. Available: <https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl>

[103] I. Craggs, "MQTT security: Who are you? Can you prove it? What can you do?" Web Page, Nov-2013 [Online]. Available: [https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/mqtt\\_security?lang=en](https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/mqtt_security?lang=en)

[104] hive mq, "MQTT security fundamentals: OAuth 2.0 and mqtt." Web Page, Nov-2017 [Online]. Available: <https://www.hivemq.com/blog/mqtt-security-fundamentals-oauth-2-0-mqtt>

[105] apache, "Apache storm." Web Page, Nov-2017 [Online]. Available: <http://storm.apache.org/>

[106] A. storm, "Storm mqtt integration." Apache storm website, Nov-2017 [Online]. Available: <http://storm.apache.org/releases/1.1.0/storm-mqtt.html>

[107] Wikipedia, "Storm (event processor)." Nov-2017 [Online]. Available: <https://en.wikipedia.org/w/index.php?>

[title=Storm \(event\\_processor\)&oldid=808771136](#)

[108] elastic.io, “ELK stack.” elastic.io website, Nov-2017 [Online]. Available: <https://www.elastic.co/products>

[109] Smart Factory, “Storing iot data using open source. MQTT and elasticsearch - tutorial.” Web Page, Oct-2016 [Online]. Available: <https://smart-factory.net/mqtt-elasticsearch-setup/>

[110] Smart Factory, “MQTT and kibana - open source graphs and analysis for iot.” Web Page, Nov-2017 [Online]. Available: <https://smart-factory.net/mqtt-and-kibana-open-source-graphs-and-analysis-for-iot/>

[111] erlang-mqtt, “Erlang mqtt broker.” wmqtt website, Nov-2017 [Online]. Available: <http://emqtt.io/docs/v2/index.html>

[112] D. Industries, “Grovepi.” Dexter Industries website, Nov-2017 [Online]. Available: <https://www.dexterindustries.com/grovepi/>

[113] S. Studio, “Grove relay.” seed studio website, Nov-2017 [Online]. Available: <http://wiki.seeed.cc/Grove-Relay/>

[114] seed studio, “Grove led socket kit.” Seed studio website, Oct-2017 [Online]. Available: [http://wiki.seeed.cc/Grove-LED\\_Socket\\_Kit/](http://wiki.seeed.cc/Grove-LED_Socket_Kit/)

[115] G. von Laszewski, *Cloud computing with the raspberry pi*, Fall 2018. Bloomington, Indiana: Indiana University, 2018 [Online]. Available: <https://github.com/cloudmesh-community/book/vonLaszewski-pi.epub?raw=true>

[116] Apache Avro, “Apache avro 1.8.2 documentation.” Web Page, Feb-2017 [Online]. Available: <http://avro.apache.org/docs/1.8.2/index.html>

[117] Apache Avro, “Apache avro 1.8.2 getting started (python).” Web Page, Feb-2017 [Online]. Available: <http://avro.apache.org/docs/1.8.2/gettingstartedpython.html>

[118] Apache Avro, “Apache avro 1.8.2 getting started (java).” Web Page, Feb-2017 [Online]. Available: <http://avro.apache.org/docs/1.8.2/gettingstartedjava.html>

- [119] Apache Avro, “Apache avro 1.8.2 hadoop mapreduce guide.” Web Page, Feb-2017 [Online]. Available: <http://avro.apache.org/docs/1.8.2/mr.html>
- [120] Apache Avro, “Apache avro 1.8.2 specification.” Web Page, Feb-2017 [Online]. Available: [http://avro.apache.org/docs/1.8.2/spec.html#schema\\_record](http://avro.apache.org/docs/1.8.2/spec.html#schema_record)
- [121] Golang, “The go programming language.” Web Page, Sep-2018 [Online]. Available: <https://golang.org/doc/>
- [122] C. A. R. Hoare, *Communicating sequential processes*. Electronit version of Prentice Hall International, 1985 [Online]. Available: <http://www.usingcsp.com/cspbook.pdf>
- [123] D. Pountain and D. May, *A tutorial introduction to occam programming*. New York, NY, USA: McGraw-Hill, Inc., 1987 [Online]. Available: <http://www.transputer.net/obooks/72-occ-046-00/tuinocc.pdf>
- [124] D. C. Hyde, *Introduction to the programming language occam*. Bucknell University, 1995 [Online]. Available: <http://www.cs.otago.ac.nz/cosc441/occam.pdf>
- [125] Tiobe, “TIOBE index.” Web Page, Sep-2018 [Online]. Available: <https://www.tiobe.com/tiobe-index/>
- [126] G. von Laszewski, *Linux for cloud computing*, Fall 2019. Bloomington, Indiana: Indiana University, 2019 [Online]. Available: <https://laszewski.github.io/book/linux/>
- [127] A. Mashraki, “Go cheat sheet.” Web Page, Sep-2015 [Online]. Available: [https://github.com/a8m/go-lang-cheat-sheet/blob/master/golang\\_refcard.pdf](https://github.com/a8m/go-lang-cheat-sheet/blob/master/golang_refcard.pdf)
- [128] K. Seguin, “The little go book.” Web Page, Jan-2018 [Online]. Available: <https://github.com/karlseguin/the-little-go-book>
- [129] M. McGranaghan, “Go by example.” Web Page, Oct-2018 [Online]. Available: <https://gobyexample.com>
- [130] S. Keys, “Learn go in y minutes.” Web Page, Sep-2013 [Online]. Available: <https://learnxinyminutes.com/docs/go/>

- [131] G. Schier, “Sendwithus go workshop.” Web Page, Oct-2015 [Online]. Available: <https://github.com/sendwithus/workshop-go>
- [132] A. N. Chomsky, “Go fragments: A collection of annotated go programs examples.” Web Page, Mar-2013 [Online]. Available: <http://www.gofragments.net/>
- [133] K. Quest, “50 shades of go: Traps, gotchas, common mistakes for new golang devs.” Web Page, Jan-2018 [Online]. Available: <http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>
- [134] S. VJ, “GoLang tutorials.” Web Page, May-2011 [Online]. Available: <http://golangtutorials.blogspot.com/2011/05/table-of-contents.html>
- [135] I. Zazueta-Hall, “Building bridges that educate and empower underrepresented communities.” Web Page, Jan-2015 [Online]. Available: <https://golangbridge.org/>
- [136] I. Zazueta-Hall, “Building bridges that educate and empower underrepresented communities.” Web Page, Jan-2013 [Online]. Available: <https://bridgefoundry.org/>
- [137] golangbot.com, “Golang tutorial series.” Web Page, Sep-2018 [Online]. Available: <https://golangbot.com/learn-golang-series/>
- [138] Stefan Nilsson, “Algorithms to go.” Web Page, Aug-2019 [Online]. Available: <https://yourbasic.org/>
- [139] Many, “Go language tutorials.” Web Page, Jan-2018 [Online]. Available: <https://www.cybrhome.com/topic/go-language-tutorials>