# E6156 – Topics in SW Engineering

# Cloud Computing

## Lecture 1: Introduction

*We will start in a couple of minutes*

# *Contents and Agenda*

# Contents and Agenda

- Introduction
  - Your instructor
  - Course logistics, objectives, grading and overview
  - Cloud computing – some core concepts
  - Motivating sample application
- Technical topics
  - Full-stack web application
  - Microservices
  - REST (part I)
  - Putting the pieces together
- Project – getting started

# *Introduction*

*About Your Instructor*
*Course Logistics*

# About your Instructor

- 35 years in computer science industry:
  - IBM Fellow.
  - Microsoft Technical Fellow.
  - Chief Technology Officer, CA technologies.
  - Dell Senior Technical Fellow.
  - CTO, Co-Founder, Seeka.tv.
  - Ansys (current):
    - Ansys Fellow, Chief SW Architect;
    - Lead development on cloud, platform, solutions and Autonomous Vehicles/Advanced Driver Assistance Systems
- Academic experience:
  - BA, MS, Ph.D., Computer Science, Columbia University.
  - Approx. 15 semesters as an Adjunct Professor.
  - Professor of Professional Practice in CS (2018)
  - Courses:
    - E1006: Intro. to Computing
    - W4111: Intro. to Databases
    - E6998, E6156: Advanced Topics in SW Engineering (Cloud Computing)

I have taught some version of this class 8 times.

- Approx. 65 technical publications; Approx. 12 patents.

## Personal:

- Two children:
  - College Sophomore.
  - 2019 Barnard Graduate.
- Hobbies:
  - Krav Maga, Black Belt in Kenpo Karate.
  - 1LT, New York Guard.
  - Bicycling.
  - Astronomy.
  - Languages:
    - Proficient in Spanish.
    - Learning Arabic.

# Course Logistics

- Sessions:
  - Lectures:
    - Session is Fridays, 1:10 – 3:40 PM
    - I can cover material much, more quickly than you can implement in projects. Typical lecture cadence will be
      - Lecture on new material for two lectures.
      - Followed by one lecture period of project discussions and reviews.
  - Instructor Office Hours:
    - Fridays, 8:30-10:00 AM, 4:00-5:00 (CLIC Lab, CS 486)
    - I frequently have extra office hours, usually around due dates for assignments.
- Grading and assignments:
  - This is a team project class. Students form 5 person teams.
  - Final project's completion of objectives determines the grade.
  - There are mandatory weekly project status reports and periodic reviews.

# Some Details: Assignments Structure and Objectives

- No exams.
- Projects:
  - We will form small teams and build a simple, realistic microservice/cloud application.
  - The teams will use a simplified version of Agile SW Development.
  - There will be a common core set of functionality to help you get started, e.g. user registration.
  - The teams can choose their project functionality -- what application do you want to build?
  - Teams have some flexibility on which cloud technology they use based on their interests.
- Course objectives:
  - Practical experience with modern technology for building solutions in an agile team.
  - Be able to say that you have built a microservice/cloud application delivering APIs, and cite a list of technology used, e.g. IaaS, PaaS, SaaS, API Management, DBaaS, federated security, ... ...
  - Become a better programmer through experience with patterns and best practices.
    My experience has been that most students write "crappy code."
  - Prepare you for internships and jobs.
  - Have seriously cool stuff to put on your resume and discuss on interviews.

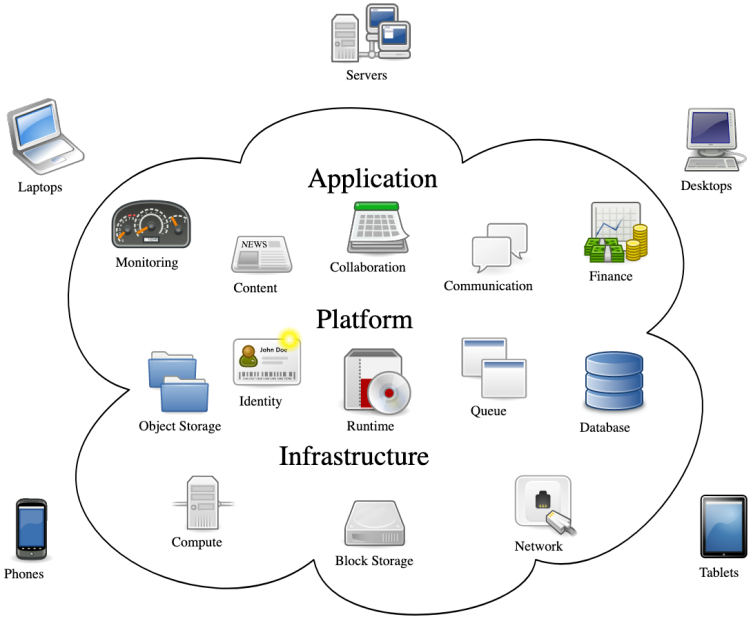
# Course Material

- Course Material:
  - No textbook. Textbooks become out of date in this rapidly changing area. Material would span several books.
  - Lecture notes and code samples on GitHub. Sample code is mostly in Flask/Python and (maybe) JavaScript/Node.js.
    - Course website: https://donald-f-ferguson.github.io/E6156F21/
    - Course repo: https://github.com/donald-f-ferguson/E6156F21
  - References to web documents and tutorials.
  - There will be some programming and example for UI in HTML/CSS/Angular, but this is not the focus of this course. UI/User Experience is a multi-semester course by itself.
- Environment:
  - Required:
    - AWS (team) free tier account
    - We will try to use Azure and but do not sign up yet.
    - Trello –– project planning, light weight agile development.
  - Recommended:
    - PyCharm (free student licenses https://www.jetbrains.com/student/) and/or PyCharm
    - Ananconda to read Jupyter Notebooks for lectures (https://www.anaconda.com/download/#macos)
    - WebStorm for JavaScript (same link as PyCharm)
- Collaboration environments:
  - Slack: Please join the channel #e6156f21 – https://join.slack.com/t/dff-columbia/shared_invite/zt-3mnfb5it-MaZbLZRd9vGg06M5ueC8Zw
  - We will use the Ed Discussion features, accessible through CourseWorks, for discussions.

# *Cloud Computing – Core Concepts*

# Cloud Computing

- "Cloud computing is an information technology (IT) paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility." (https://en.wikipedia.org/wiki/Cloud_computing)




NIST Reference Architecture

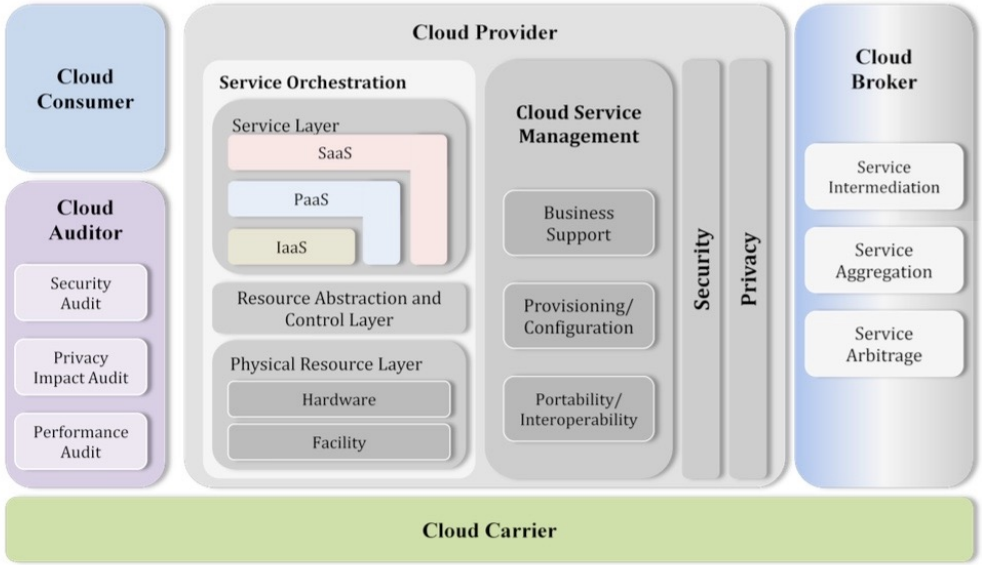



**Figure 1: The Conceptual Reference Model**

# Core Layers

## Cloud Computing Layers

Our first cloud elements will be IaaS (VMs).

Resources Managed at each layer

Software as a service (SaaS)

Business Applications, Web Services, Multimedia

Application

- Google Apps
- Salesforce, Zendesk, ServiceNow, Concur, … …
- Office 365, Trello
- … …

Platform as a service (PaaS)

Software Framework (Java, .NET)
Storage (DB, File)

Platform

- Google App Engine, AWS Elastic Beanstalk, …
- AWS SQS, Azure Queuing Service, Google PubSub
- DynamoDB, Cosmos DB, … …
- … …

Infrastructure as a service (IaaS)

Computation (VM) Storage (block)

Infrastructure

- AWS EC2, Google Compute Engine, Azure VMs, … …
- AWS Elastic Container Service, Azure Kubernetes, … …

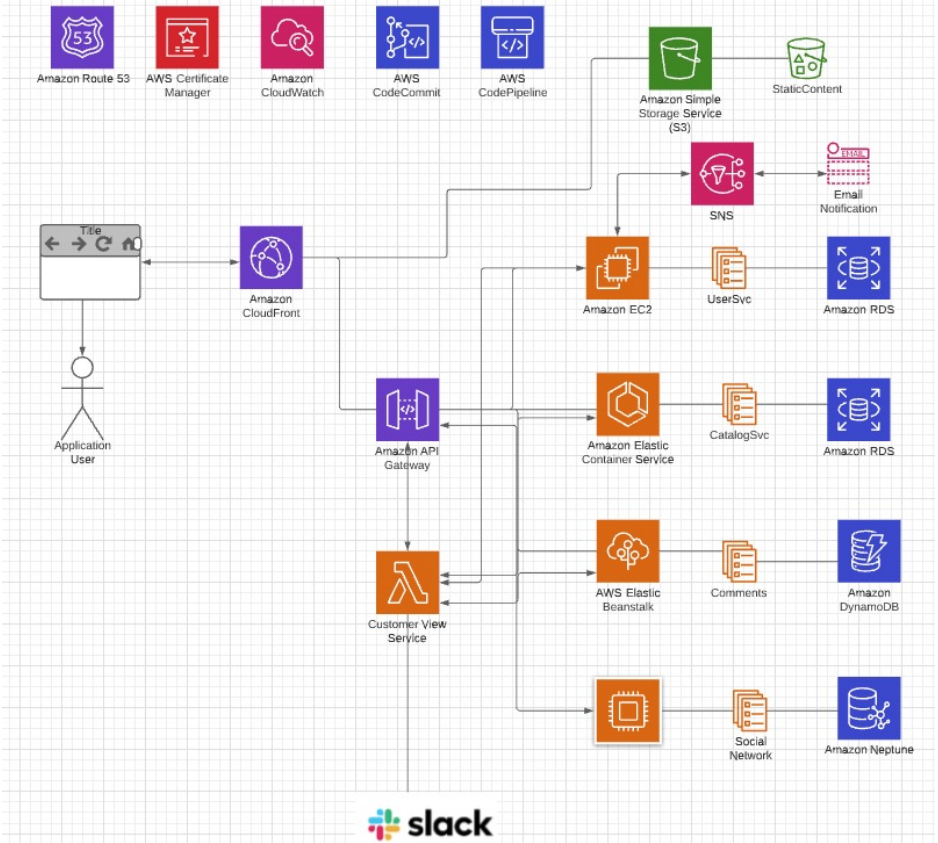
CPU, Memory ,Disk, Bandwidth

Hardware

- IaaS is compute, storage, networking.
- With an OS.
- Delivered as a set of VMs and/or containers.

# Motivating Sample Application (Topology)

- Note: Cannot show all of the technology and connections with creating spaghetti.
- Databases/Storage:
  - RDS
  - DynamoDB
  - MongoDB (or Document DB)
  - Neptune (or Neo4j)
  - S3
- Networking/Communication:
  - Route 53 (DNS)
  - Certificate Management
  - CloudFront
  - VPC
  - API Gateway
- Compute:
  - EC2
  - Elastic Beanstalk
  - Elastic Container Service (or Docker)
  - Lambda Functions
- Integration/collaboration:
  - SNS, SQS
  - Email
  - Slack
- Continuous Integration/Continuous Deployment:
  - Code Commit, Code Pipeline
  - Or GitHub and actions

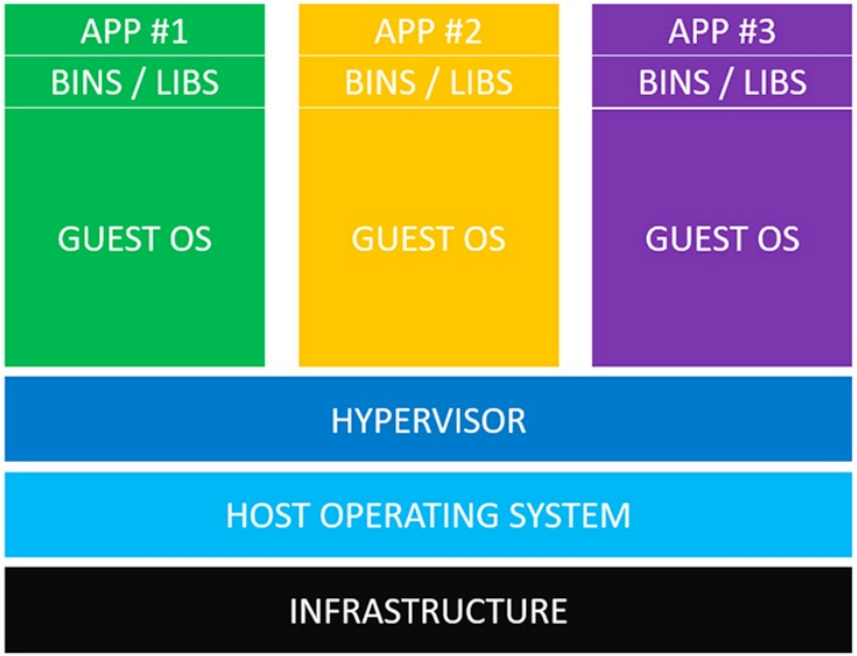


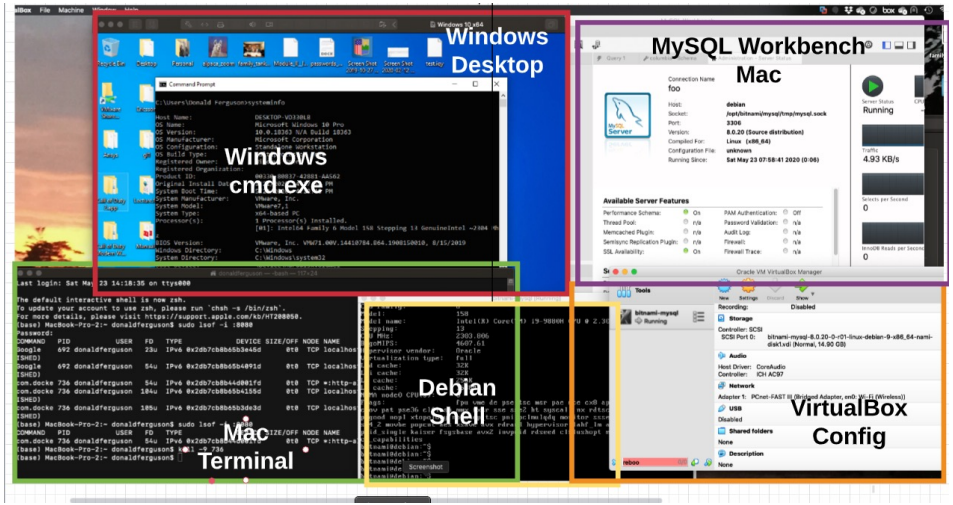
# *Technical Topics*

# *VMs, IaaS*

# Virtualization, Virtual Machines

- "In computing, a virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination." (https://en.wikipedia.org/wiki/Virtual_machine)

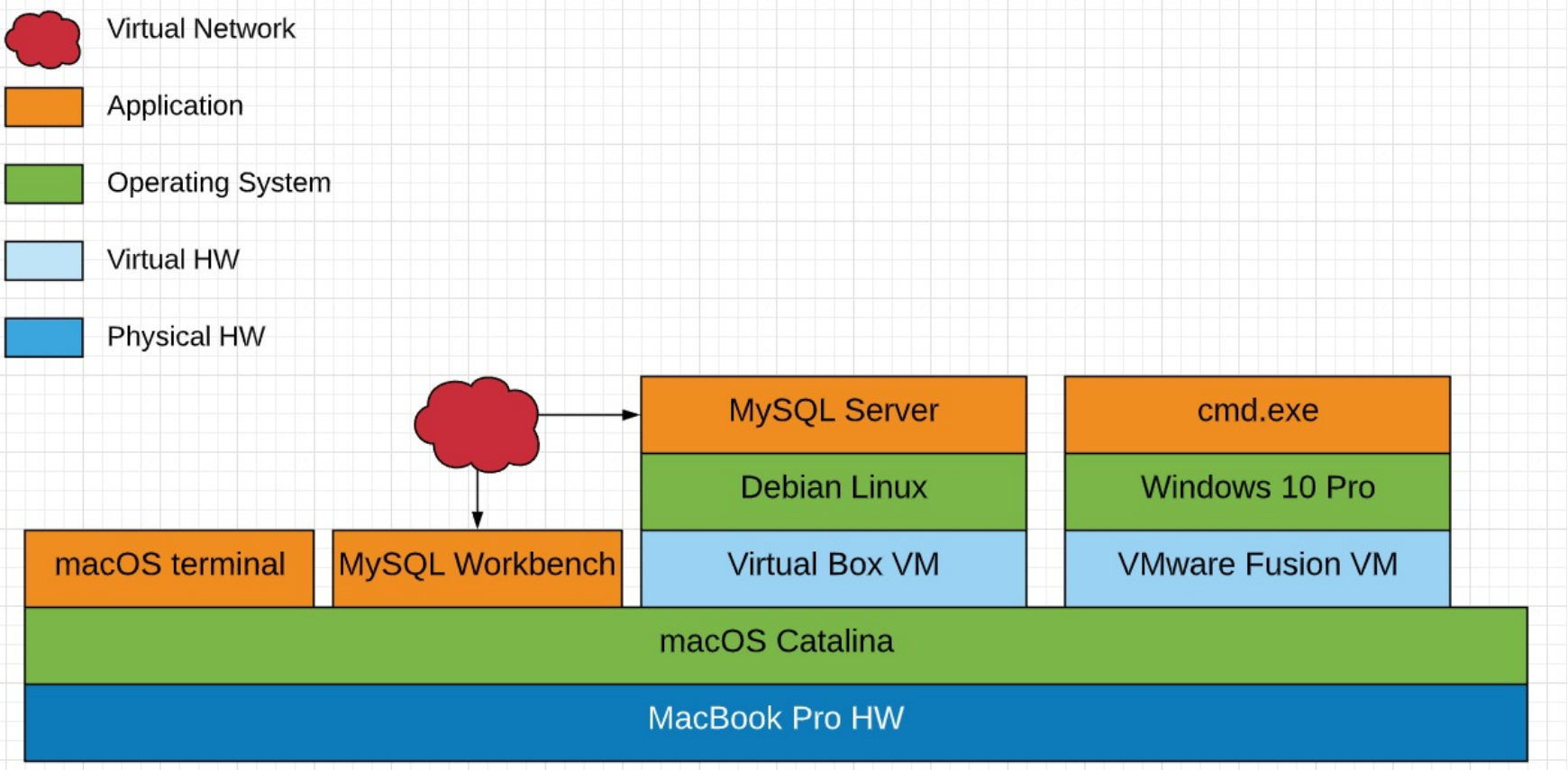


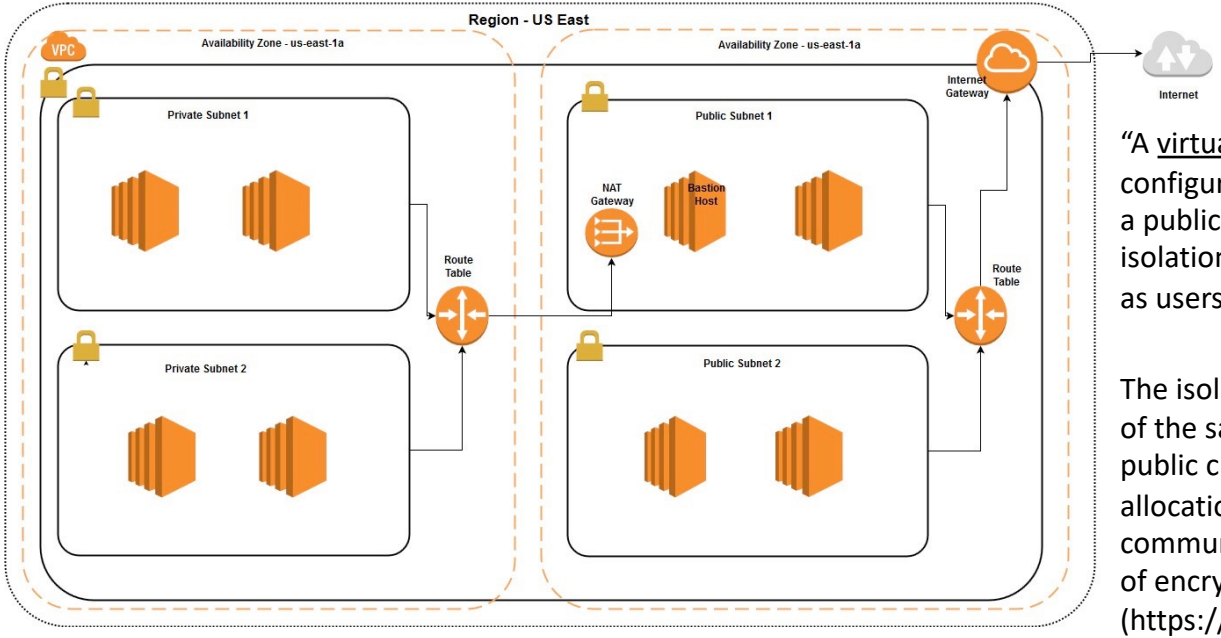

On mac laptop, I have:

- One host operating system and HW (Mac, MacOS)
- Two hypervisors (Virtual Box, VMWare Fusion)
- Two guest OS: Windows 10, Debian Linux

# Virtual Machines on Physical Machines



# Virtual Private Cloud




"A virtual private cloud (VPC) is an on-demand configurable pool of shared resources allocated within a public cloud environment, providing a certain level of isolation between the different organizations (denoted as users hereafter) using the resources. ... ...

The isolation between one VPC user and all other users of the same cloud (other VPC users as well as other public cloud users) is achieved normally through allocation of a private IP subnet and a virtual communication construct (such as a VLAN or a set of encrypted communication channels) per user." (https://en.wikipedia.org/wiki/Virtual_private_cloud)

- A virtual machine on my laptop has a virtual adaptor connected to a real network.
- A virtual private cloud is a set of virtual machines connected to a virtual network.
- All cloud providers support virtual private clouds with the same concepts, but with slightly different realizations and terms.

# Some Terminology (AWS)

## There are several online tutorials. Understanding some terms is useful.

- AWS has the concept of a Region, which is a physical location around the world where we cluster data centers. We call each group of logical data centers an Availability Zone. Each AWS Region consists of multiple, isolated, and physically separate AZ's within a geographic area.
- An Availability Zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. AZs give customers the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center.
- A VPC is a virtual private cloud, which works like a private network to isolate the resources within it.
- A route table contains a set of rules, called routes, that are used to determine where network traffic is directed.
- A subnet is a defined set of network IP addresses that are used to increase the security and efficiency of network communications. You can think of them like postal codes, used for routing packages from one location to another.
- A network interface represents a virtual network card. The network interface displays its network interface ID, subnet ID, VPC ID, security group, and the Availability Zone that it exists in.
- A security group is a set of rules that controls the network access to the resources it is associated with. Access is permitted only to and from the components defined in the security group's inbound and outbound rules.
- An internet gateway is a VPC component that allows communication between instances in your VPC and the internet.
- A VPC endpoint enables you to privately connect your VPC to supported AWS services without using public IP addresses.

## All clouds have similar concepts but with different names.

# Let's Build our First VPC

- Well, actually, let's not do that for now.
- We are just going to create our first virtual machine.
- But,
  - The virtual machine is in my "default" VPC.
  - Setting up and using the VM requires understanding some of the concepts.
- We can mostly just use the EC2 wizard for now.
- I am going to set up a VM to host our first cloud application
  - I will deploy the application and supporting SW.
  - Basic tasks:
    1. Configure and deploy VM, SSH credentials, security group rules.
    2. Install supporting software: Python, MySQL.
    3. Set up the database and deploy the application.
    4. Run "Hello World" test.
- <u>But first, let's walk through the application structure and implementation.</u>

# *Full Stack Application Introduction*

# Full Stack Application

## Full Stack Developer Meaning & Definition

In technology development, full stack refers to an entire computer system or application from the front end to the back end and the code that connects the two. The back end of a computer system encompasses "behind-the-scenes" technologies such as the database and operating system. The front end is the user interface (UI). This end-to-end system requires many ancillary technologies such as the network, hardware, load balancers, and firewalls.

## FULL STACK WEB DEVELOPERS

Full stack is most commonly used when referring to web developers. A full stack web developer works with both the front and back end of a website or application. They are proficient in both front-end and back-end languages and frameworks, as well as server, network, and hosting environments.

Full-stack developers need to be proficient in languages used for front-end development such as HTML, CSS, JavaScript, and third-party libraries and extensions for Web development such as JQuery, SASS, and REACT. Mastery of these front-end programming languages will need to be combined with knowledge of UI design as well as customer experience design for creating optimal front-facing websites and applications.

https://www.webopedia.com/definitions/full-stack/

## Full Stack Web Developer

A full stack web developer is a person who can develop both **client** and **server** software.

In addition to mastering HTML and CSS, he/she also knows how to:

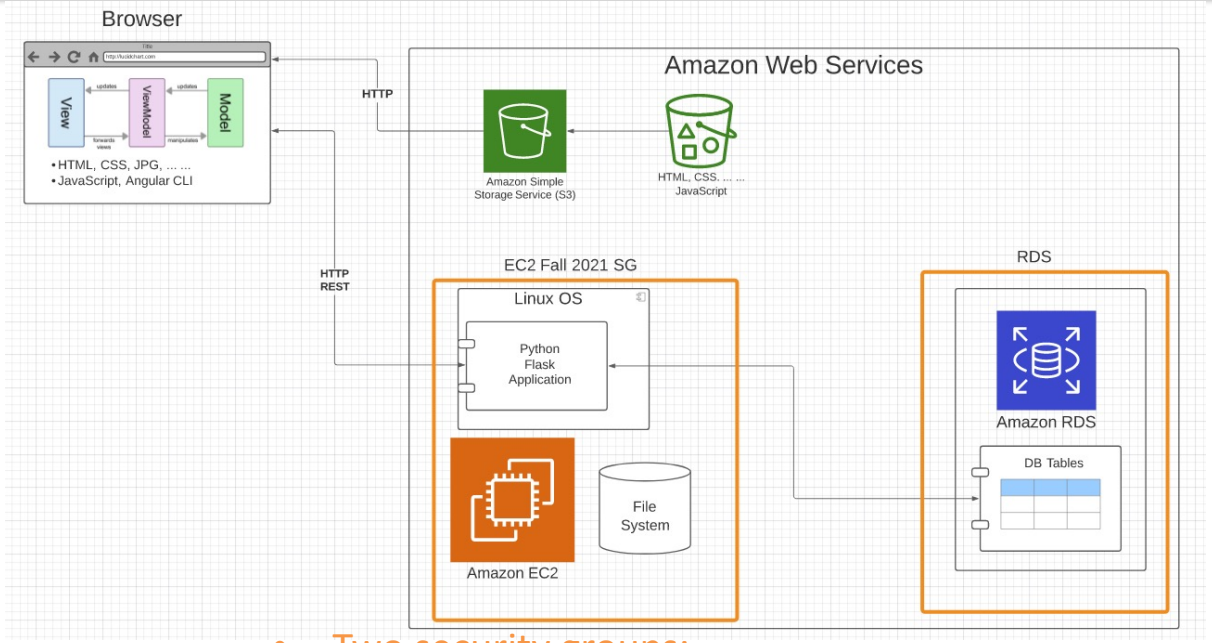
- Program a **browser** (like using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (like using PHP, ASP, Python, or Node)
- Program a **database** (like using SQL, SQLite, or MongoDB)

https://www.w3schools.com/whatis/whatis_fullstack.asp

- There are courses that cover topics:
  - COMS W4156: Advanced Software Engineering
  - COMS W4111: Introduction to Databases
  - COMS W4170 - User Interface Design
- This course will focus on cloud realization, microservices and application patterns, … …
- Also, I am not great at UIs … … We will not emphasize or require a lot of UI work.

# Sample Application Structure

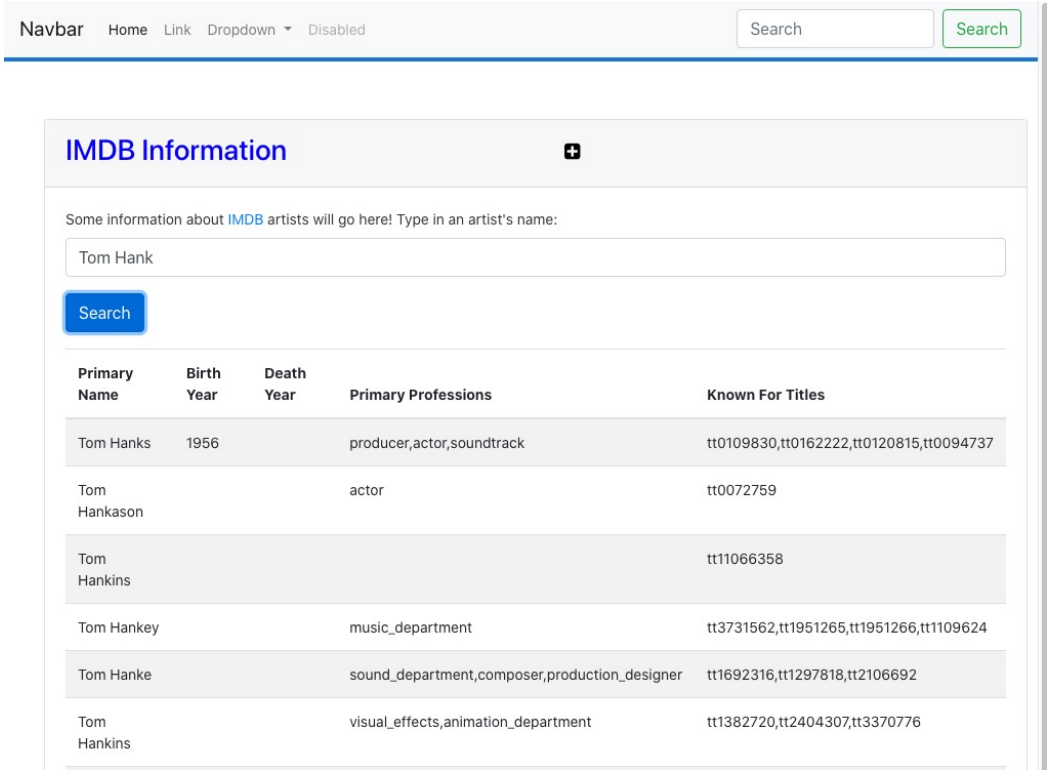
- Browser Application:
  - Content: HTML, CSS, … …
  - MVVM with Angular CLI
- Static Content:
  - S3 Bucket
  - Web site hosting
- Server: Amazon EC2 instance (IaaS)
  - Linux
  - Web Application Server
    - Flask
    - Python
- Database: Amazon RDS (DBaaS)
  - Simple tables
  - MySQL Database Server
- Lifecycle:
  - Develop and demo locally
  - Deploy and configure onto AWS
  - Will work on CI/CD later.




- Two security groups:
  - One for EC2 instance
  - One for RDS
- Slightly more secure than 1 SG.
- Will cover security later.

# *User Interface*

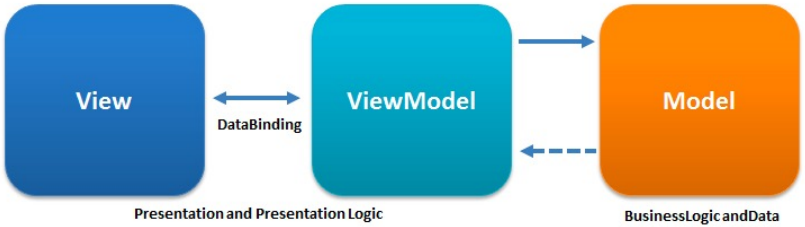
# Browser Based User Interface




- HTML define the elements, e.g.
  - Sections
  - Headers
  - Paragraphs
  - Input fieds
- CSS defines the layout and style, e.g.
  - Font sizes and colors
  - Rows and columns, and widths
  - Spacing, line breaks, … …
- Angular (or React, AngularJS) provides a framework for building interactive applications and MVVM, e.g.
  - Application, module, …
  - Components
  - Services
  - Routes/paths/routing
  - … …

# Web UI – Model-View-View Model

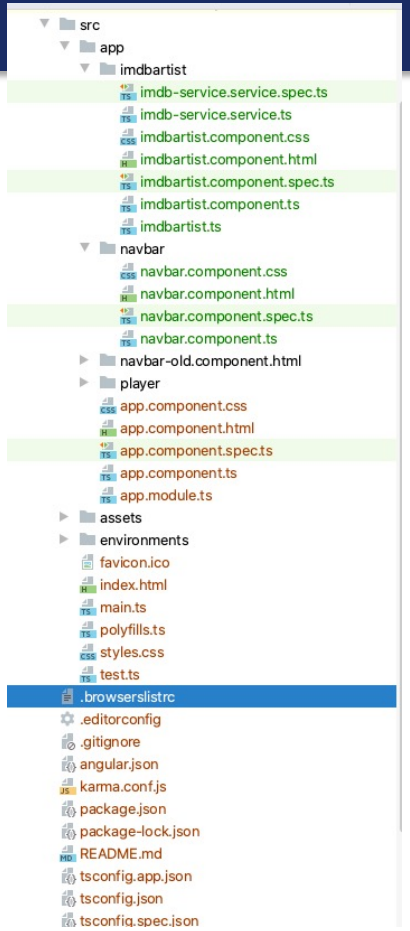
- “Model–view–viewmodel (MVVM) is a software architectural pattern that facilitates the separation of the development of the graphical user interface (the view) – be it via a markup language or GUI code – from the development of the business logic or back-end logic (the model) so that the view is not dependent on any specific model platform.
  ...
  The view model of MVVM is a value converter,[1] meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. In this respect, the view model is more model than view, and handles most if not all of the view's display logic”
  (https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel)



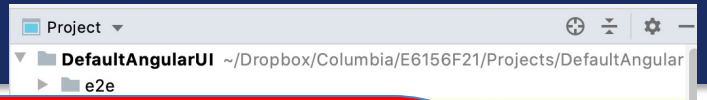

- **Model** – It simply holds the data and has nothing to do with any of the business logic.
- **ViewModel** – It acts as the link/connection between the Model and View and makes stuff look pretty.
- **View** – It simply holds the formatted data and essentially delegates everything to the Model.

# Sample UI Structure

- /src/app is the directory for the application
  - app.module.ts is the “main program”
  - app.component.* is the home page component
  - app-routing.module.ts maps URL paths to components implementing the path
- The elements of a component are:
  - .html: Page content, elements, forms, ... ...
  - .css: CSS style elements to apply to the page
  - .component.ts: Implements MVVM, and attributes/properties are the model instances.
- Services:
  - .service.ts: Interacts with business logic, gets and updates model.
  - xxx.ts defines the classes (types) for model objects.



# Sample UI Structure




- /src/
  - –
  - –
  - –
- Th
  - –
  - –
  - –
- Se
  - –
  - – xx

Disclaimer:

- I really, really dislike do UIs.
- Angular, HTML, CSS, ... give me terrible headaches.
- I get really, really grumpy.

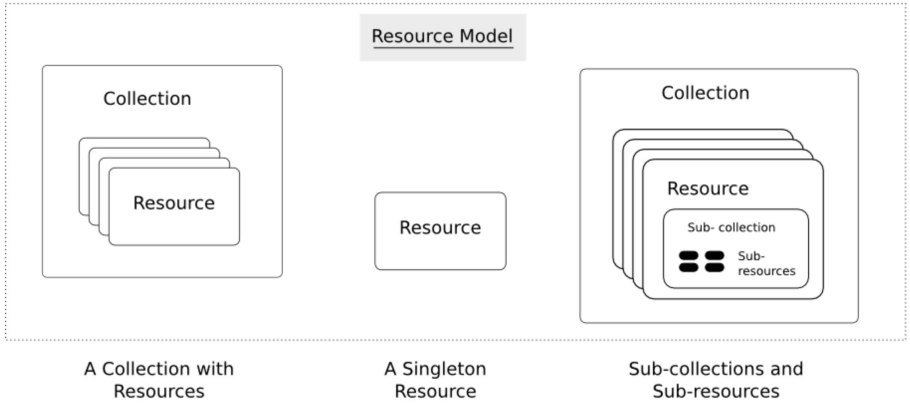
Course focus:

- Doing some UI work produces a cooler, more demoable project.
- Primary topics are on what happens behind the APIs (in the cloud).

# *REST*

# REST

- “Representational state transfer (REST) is a software architectural style which uses a subset of HTTP.[1] It is commonly used to create interactive applications that use Web services. A Web service that follows these guidelines is called RESTful. Such a Web service must provide its Web resources in a textual representation and allow them to be read and modified with a stateless protocol and a predefined set of operations. This approach allows interoperability between the computer systems on the Internet that provide these services. REST is an alternative to, for example, SOAP as way to access a Web service.” (https://en.wikipedia.org/wiki/Representational_state_transfer)




*We will learn a lot more about REST in this class.*

The only resource in example is /api/courses/<id>

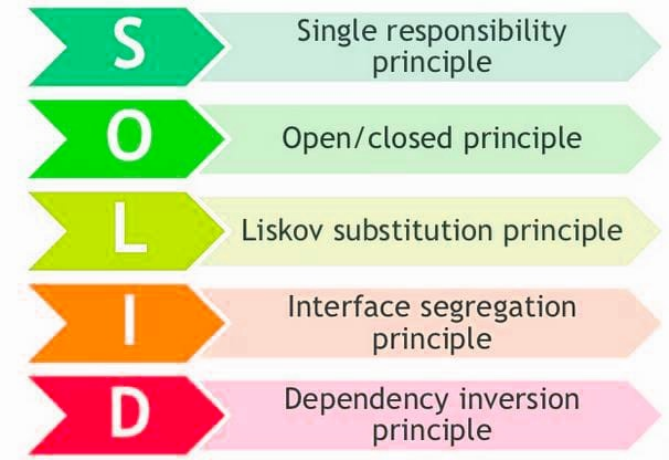
- Resource:
  - A resource is a “thing,” e.g. student, course
  - Singletons, collections, hierarchical
- URI identifies (locates) resources, e.g.
  - /api/courses: collection of all courses.
  - /api/courses/11912: a specific course
  - /api/courses?year=2003&semester=001: All courses matching a query.
- Operations:
  - No “verbs,” e.g. enrollStudent, addSection, ...
  - Create-Retrieve-Update-Delete resources using POST, GET, PUT (PATCH), DELETE.
  - Application logic implemented in functions that handle CRUD operations on URIs

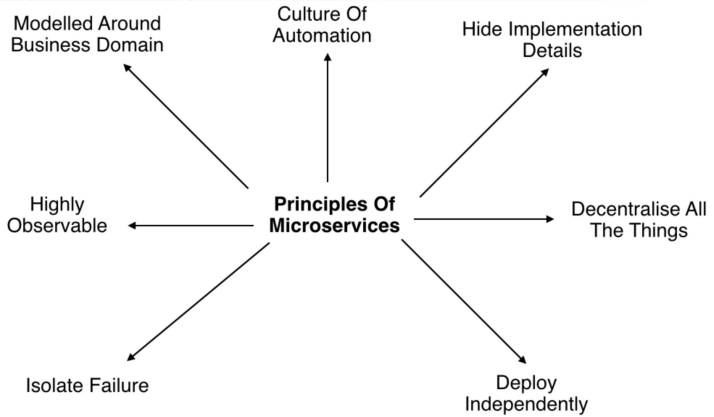
*(Web) Application*

# Some Terms (https://en.wikipedia.org/wiki/... ...)

- “A web application (or web app) is application software that runs on a web server, unlike computer-based software programs that are run locally on the operating system (OS) of the device. Web applications are accessed by the user through a web browser with an active network connection. These applications are programmed using a client–server modeled structure—the user ("client") is provided services through an off-site server that is hosted by a third-party. Examples of commonly-used web applications include: web-mail, online retail sales, online banking, and online auctions”
- “An application server is a server that hosts applications. ... ... Application server frameworks are software frameworks for building application servers. An application server framework provides both facilities to create web applications and a server environment to run them.”
- “Service-oriented architecture (SOA) is an architectural style that supports service orientation. ... ... A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.”
- “Microservice architecture – a variant of the service-oriented architecture (SOA) structural style – arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight.”

# Some Principles and Concepts

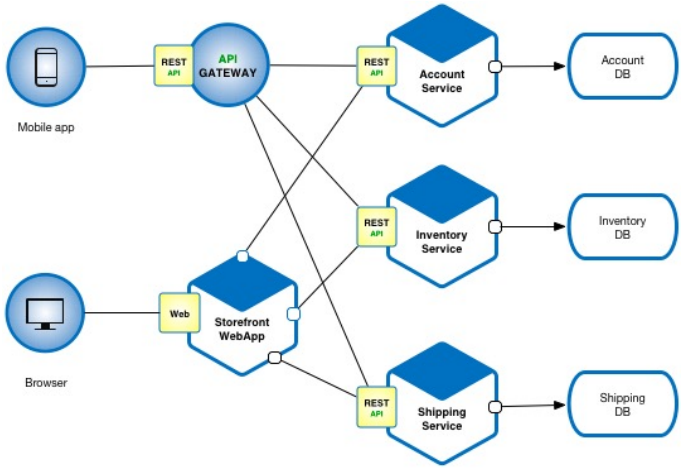



Characteristics of a Microservice Architecture

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

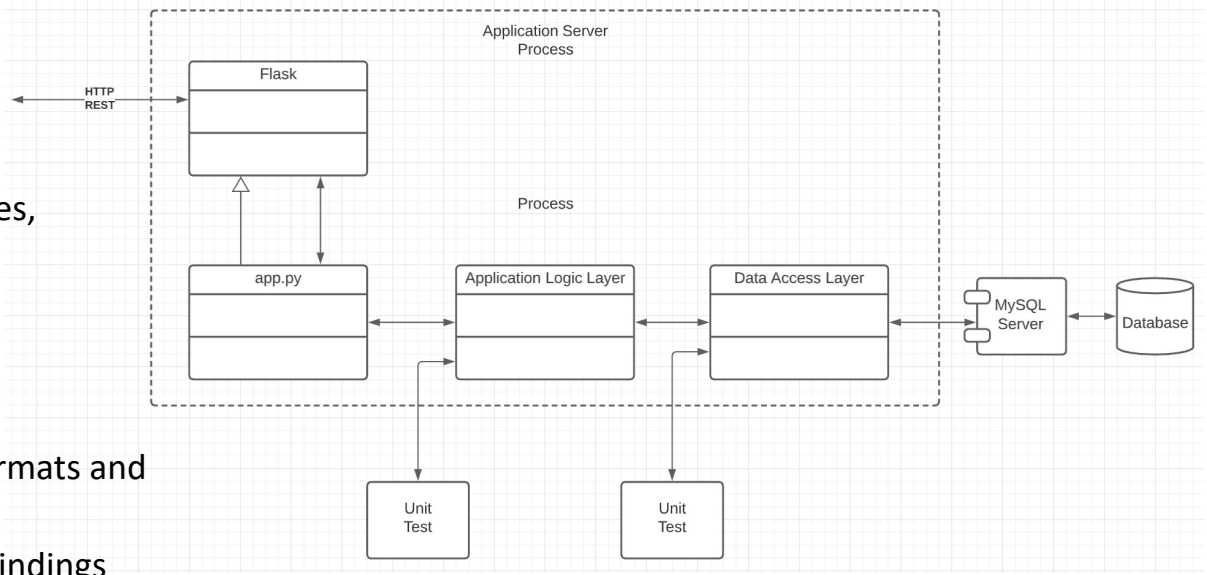
We will spend more time on all of these topics.



# Application Structure

- Single Responsibility: Each layer (class, module, ...) performs one task, and can be developed, evolved and tested in isolation.
- Data Access Layer:
  - Map between application data structure and DB.
  - Isolation application from DB changes.
- Business Logic Layer:
  - Implement business rules, policies, rules, ... ...
  - Application logic that requires calling other services.
- app.py (binding)
  - Map between HTTP and REST formats and language runtime/formats.
  - An application may have many bindings

The structure, layers, etc. are valuable for large scale, complex applications but will seem strange for our small project.



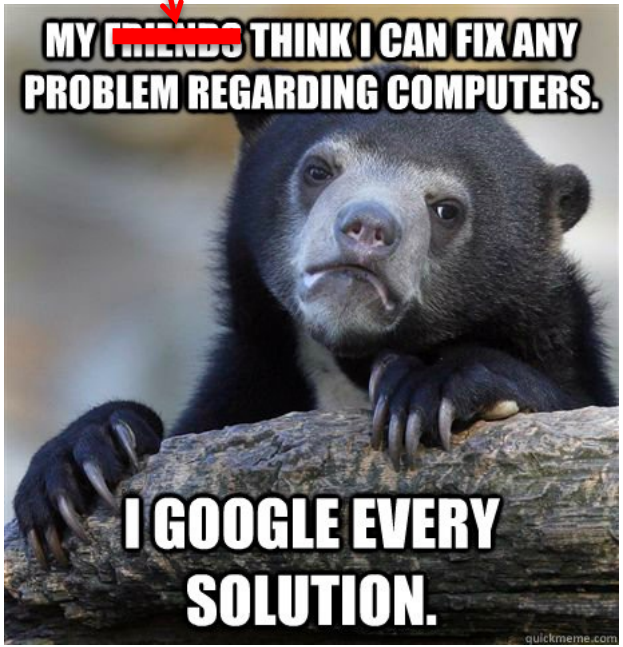
# *Sample Application Demo, Code Review and Deployment*

# Topics to Cover in Code/Config Walkthrough

- End-to-end interactions
- Database and connection
- Data Access Layer
- Business Service Layer
- app.py:
  - Routes, parameters
  - Request and Response objects
- ~~Open API example~~
- Deployment
  - Create VM
  - Create RDS
  - Install software
  - Deploy Flask Application
  - Deploy Angular application to S3
  - Install data
- Critical configuration options
  - Enable remote access
  - Security Group rules
- SSH certificate
- ssh -i ./ColumbiaFall2021.pem ec2-user@ec2-54-242-71-165.compute-1.amazonaws.com
- Certificate file protection
- Putty and other options
- requirements.txt
- Setting host to 0.0.0.0
- Environment variables
- Add rules to security groups:
  - Laptop → RDS
  - Laptop → SSH EC2 instance
  - EC2 SG → RDS SG
  - HTTP to EC2 instance on 5000

# How Do I Learn these Things? Remember Them?

STUDENTS




The Reality.

I ask myself the question, “Am I the first person who ever needed to do this?”

- If No → Google
- If Yes → Maybe this is not a good idea.


Also have to change localhost in mysql.users

- https://techviewleo.com/how-to-install-mysql-8-on-amazon-linux-2/
- https://medium.com/@rodkey/deploying-a-flask-application-on-aws-a72daba6bb80

# *Assignment/Project 1*

# Assignment 1/Project Step 1

- Individual:
  - Create an AWS account.
  - Configure laptop
    - IDE:
      - Get JetBrains student license.
      - Install PyCharm
      - Optional:
        - » Install WebStorm
        - » Install DataGrip
    - Install Anaconda (new environment)
    - Install MySQL
    - Postman
  - Demo application:
    - Clone demo-flask project from GitHub (https://github.com/donald-f-ferguson/demo-flask.git)
    - Modify application to use a simple database table for users that you create.
    - Test application with Postman
- Group: Form 5 student team.
  - Register team information on Google Sheet for class.
  - Create AWS group account for team and add members.
  - Create Trello for team. Create GitHub repo for team.
  - Complete tasks for deploying and testing the application, REST endpoint and DB.