

15-319 / 15-619

Cloud Computing

Recitation 5

February 14th, 2017

Overview

- **Administrative issues**
- Office Hours, Piazza guidelines
- **Last week's reflection**
- Project 2.1, OLI Unit 2 modules 5 and 6
- **This week's schedule**
 - Quiz 4 - February 17, 2017 (Modules 7, 8, 9)
 - Project 2.2 - February 19, 2017
 - Finalize 3 person teams for the Team Project - February 20, 2017

Announcements



- Monitor your expenses regularly
 - Check your bill frequently on TheProject.Zone
 - Check on AWS, use Cost Explorer & filter by tags
 - Check on the Azure portal since only \$100/sem
- **Terminate** your resources when you are done
 - Stopping a VM still costs EBS money (\$0.1/GB/month)
 - Amazon EC2 and Amazon Cloudwatch fees for monitoring, ELB
 - AutoScaling Group - no additional fees

Announcements



- Use spot instances as much as possible
- Complete the Project Survey for P1.1, P1.2, P2.1 Piazza @1216
- **Protect your credentials**
 - Crawlers are looking for AWS credentials on public repos!
- Primer for 3.1 is out
 - Storage I/O Benchmarking

Last Week's Reflection



- OLI: Conceptual Content
 - Unit 2 - Modules 5 and 6:
 - Cloud Management & Software Deployment Considerations
 - Quiz 3 completed
- P2.1: Azure, GCP, and AWS EC2 APIs
 - CLI, Java, Python
- P2.1: Load Balancing and AutoScaling
 - Experience horizontal scaling
 - Programmatically manage cloud resources and deal with failure
 - Initial experience with load balancing

Project 2.1



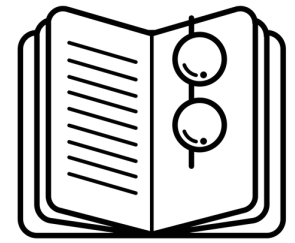
- To evaluate how well other people can read your code, we will be manually grading your submitted code
 - Azure
 - GCP
 - AWS (Horizontal and Autoscaling)
 - To enhance readability
 - Use the [Google Code Style](#) guidelines
 - Always add comments especially for complex parts

Project 2.1



- You gained experience working with Cloud Service Provider's APIs
 - Documentation may be unclear or may omit certain details.
 - Have to ensure you're referencing the correct documentation version.
- Considerations of cost vs performance

This Week: Content

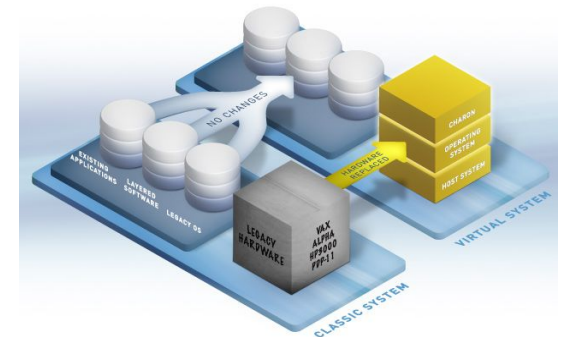


- UNIT 3: Virtualizing Resources for the Cloud
 - Module 7: Introduction and Motivation
 - Module 8: Virtualization
 - Module 9: Resource Virtualization - CPU
 - Module 10: Resource Virtualization - Memory
 - Module 11: Resource Virtualization – I/O
 - Module 12: Case Study
 - Module 13: Network and Storage Virtualization

OLI Module 7 - Virtualization

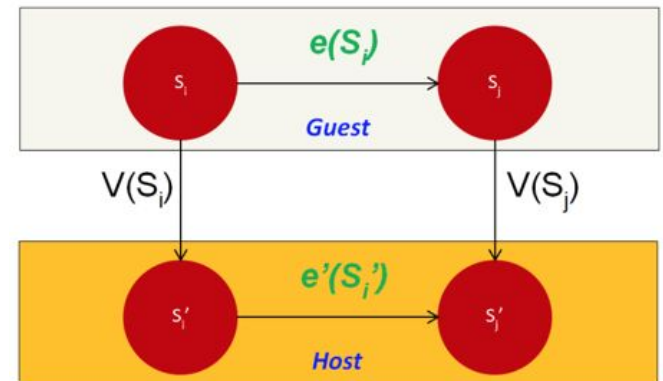
Introduction and Motivation

- Why virtualization
 - Enabling the cloud computing system model
 - Elasticity
 - Resource sandboxing
 - Limitation of General-Purpose OS
 - Mixed OS environment
 - Resource sharing
 - Time
 - Space
 - Improved system utilization and reduce costs from a cloud provider perspective



OLI Module 8 - Virtualization

- What is Virtualization
 - Involves the construction of an isomorphism that maps a virtual guest system to a real (or physical) host system
 - Sequence of operations e modify guest state
 - Mapping function $V(S_i)$
- Virtual Machine Types
 - Process Virtual Machines
 - System Virtual Machines

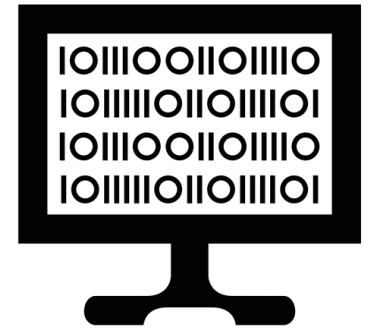


OLI Module 9

Resource Virtualization - CPU

- Steps of CPU Virtualization
 - Multiplexing a physical CPU among virtual CPUs
 - Virtualizing the ISA (Instruction Set Architecture) of a CPU
- Code Patch, Full Virtualization and Paravirtualization
- Emulation (Interpretation & Binary Translation)
- Virtual CPU

This Week: Project

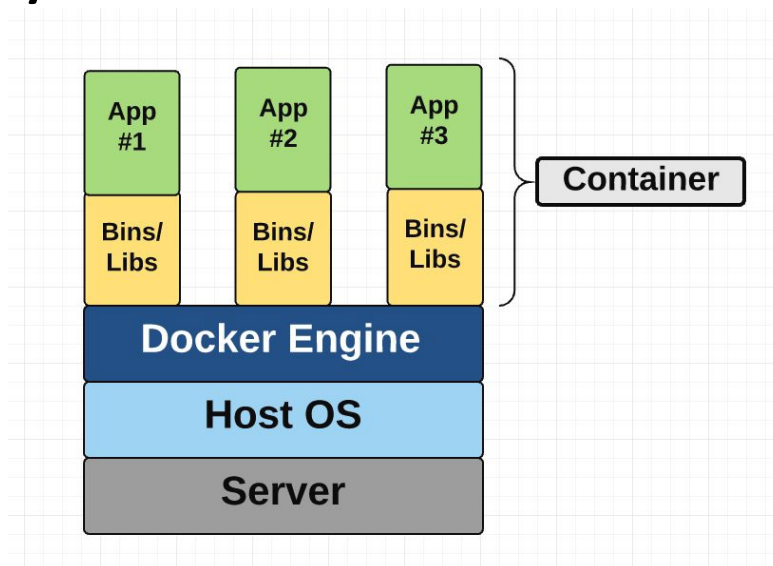


- P2.1: Horizontal Scaling and Autoscaling
 - MSB Interview
- P2.2: Containers and Kubernetes
 - Building a Coding Interview Playground
 - Working with Docker and Kubernetes

Containers



- Provides OS-level virtualization.
- Provides private namespace, network interface and IP address, etc.
- Big difference with VMs is that containers share the host system's kernel with other containers.



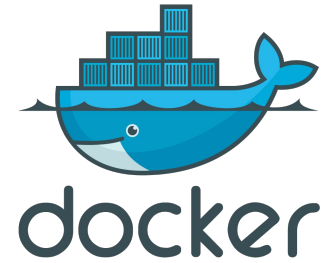
Why Containers?



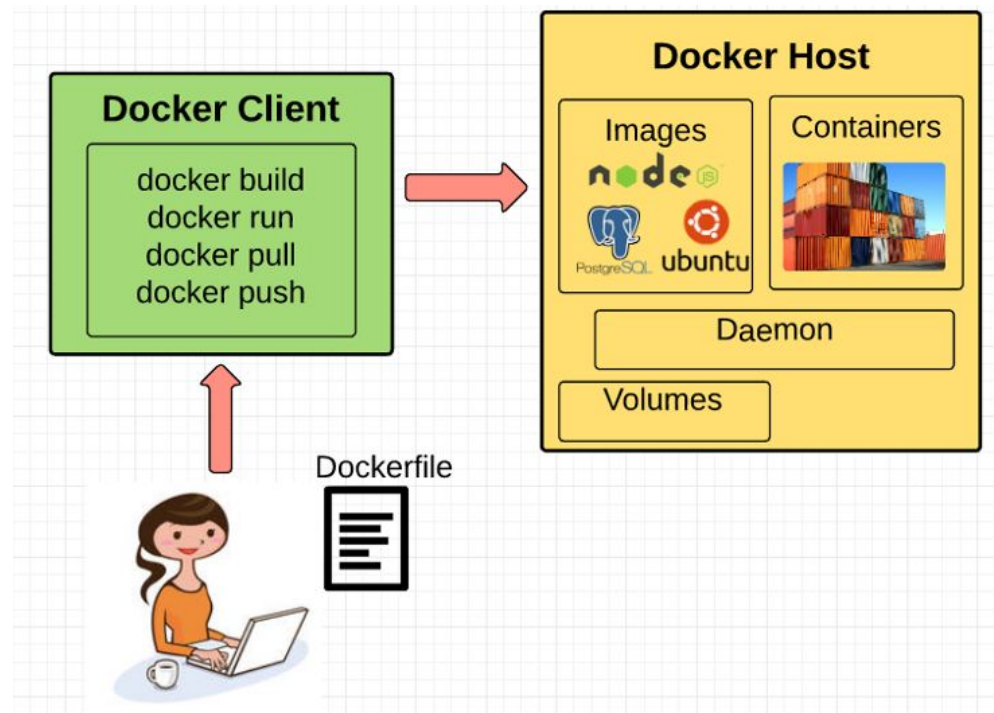
- Faster deployment
- Portability across machines
- Version control
- Simplified dependency management

Build once, run anywhere

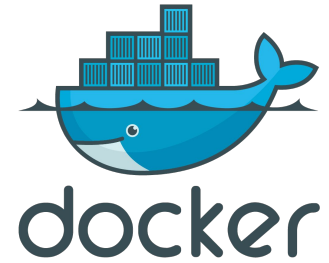
Docker Engine



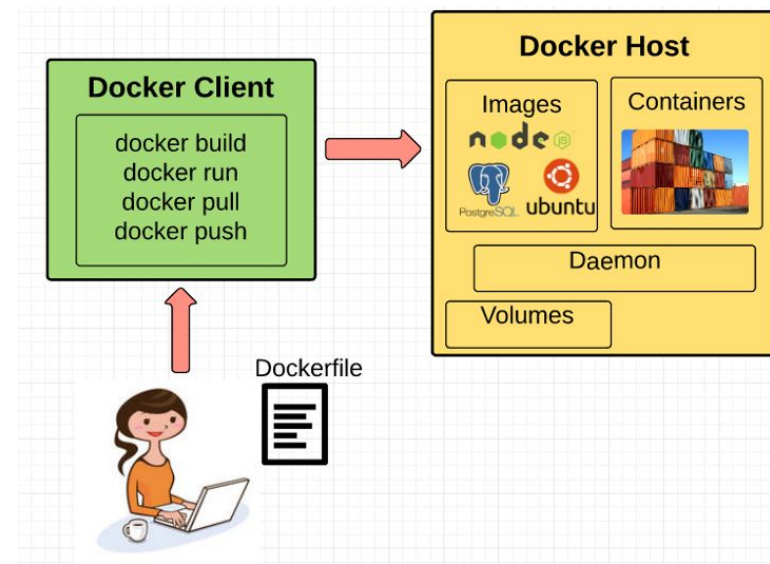
- An orchestrator that comprises:
 - Docker Daemon
 - Docker Client
 - REST API



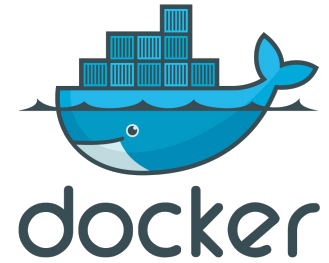
Docker Daemon



- Main brains behind Docker Engine
- The Docker Client is used to communicate with the Docker Daemon
- The Daemon does not have to be on the same machine as the Client



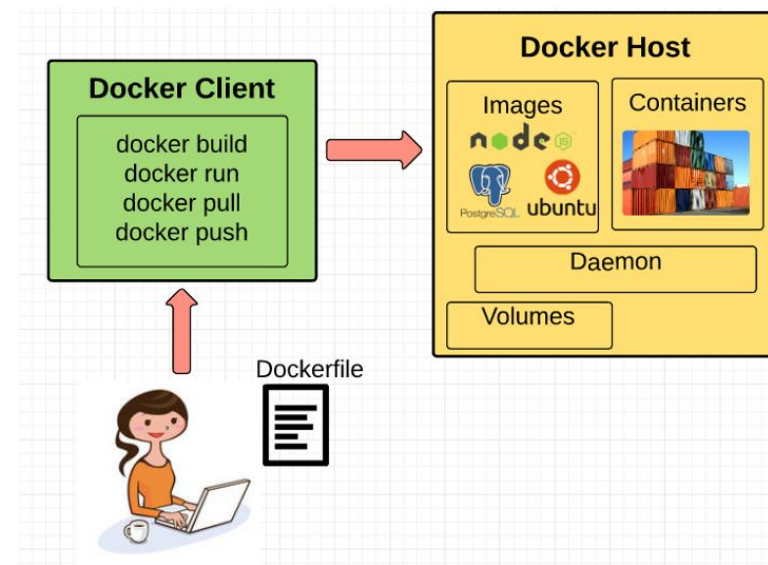
Docker Client



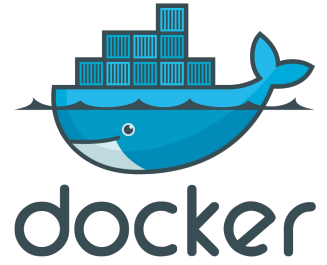
- Also known as Docker CLI
- When you type:

```
docker build nginx
```

You are telling the Docker client to forward the
build nginx
instruction to the Daemon



Dockerfile



- We can use a Dockerfile to build container images
- Dockerfile tells Docker:
 - What base image to work from
 - What commands to run on base image
 - What files to copy to the base image
 - What ports should the container listen on?
 - What binaries to execute when the container launches?
- In short, Dockerfile is a recipe for Docker images

Let's go through a sample Dockerfile!

Example Dockerfile

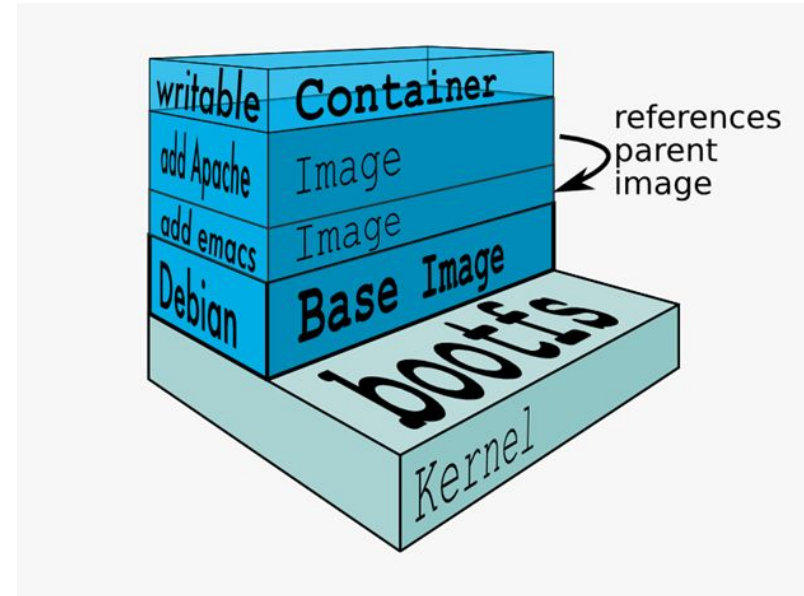
```
# Debian as the base image  
FROM debian:latest
```

```
# Install additional packages  
RUN apk add --update emacs  
RUN apk add --update apache
```

```
# index.html must be in the current directory  
ADD index.html /home/demo/
```

```
# Define the command which runs when the container starts  
CMD ["cat /home/demo/index.html"]
```

```
# Use bash as the container's entry point. CMD is the argument to this entry  
point  
ENTRYPOINT ["/bin/bash", "-c"]
```



Example Dockerfile

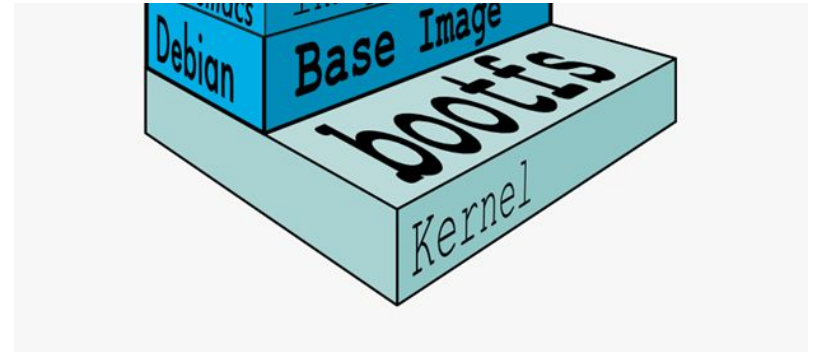
```
# Debian Linux as the base image  
FROM debian:latest
```

```
# Install additional packages  
RUN apk add --update emacs  
RUN apk add --update apache
```

```
# index.html must be in the current directory  
ADD index.html /home/demo/
```

```
# Define the command which runs when the container starts  
CMD ["cat /home/demo/index.html"]
```

```
# Use bash as the container's entry point. CMD is the argument to this entry  
point  
ENTRYPOINT ["/bin/bash", "-c"]
```



Example Dockerfile

```
# Alpine Linux as the base image
```

```
FROM debian:latest
```

```
# Install additional packages
```

```
RUN apk add --update emacs
```

```
RUN apk add --update apache
```

```
# index.html must be in the current directory
```

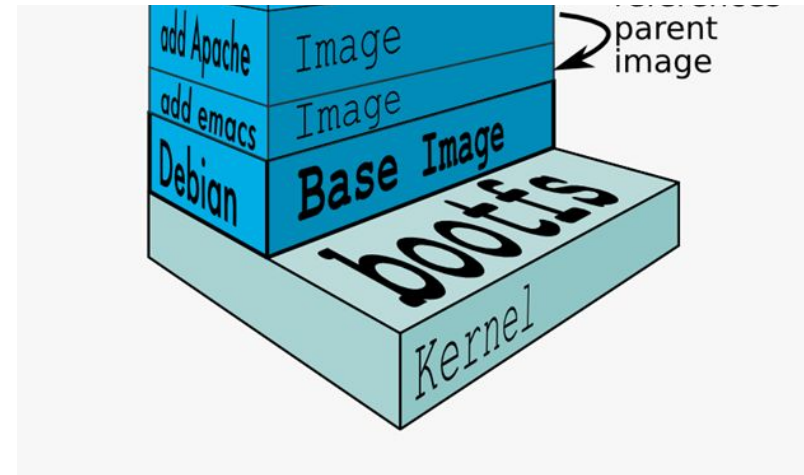
```
ADD index.html /home/demo/
```

```
# Define the command which runs when the container starts
```

```
CMD ["cat /home/demo/index.html"]
```

```
# Use bash as the container's entry point. CMD is the argument to this entry point
```

```
ENTRYPOINT ["/bin/bash", "-c"]
```



Example Dockerfile

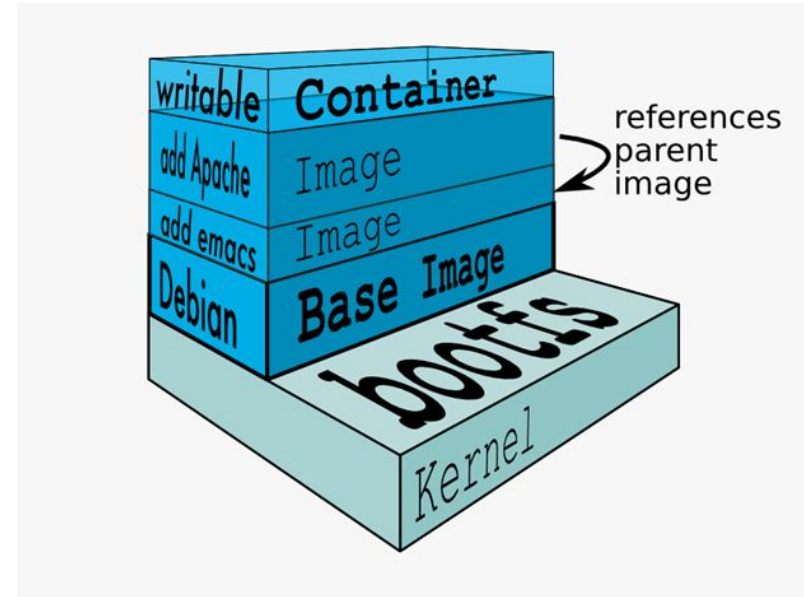
```
# Alpine Linux as the base image
FROM debian:latest

# Install additional packages
RUN apk add --update emacs
RUN apk add --update apache

# index.html must be in the current directory
ADD index.html /home/demo/

# Define the command which runs when the container starts
CMD ["cat /home/demo/index.html"]

# Use bash as the container's entry point. CMD is the argument to this entry
point
ENTRYPOINT ["/bin/bash", "-c"]
```



Example Dockerfile

```
# Alpine Linux as the base image  
FROM debian:latest
```

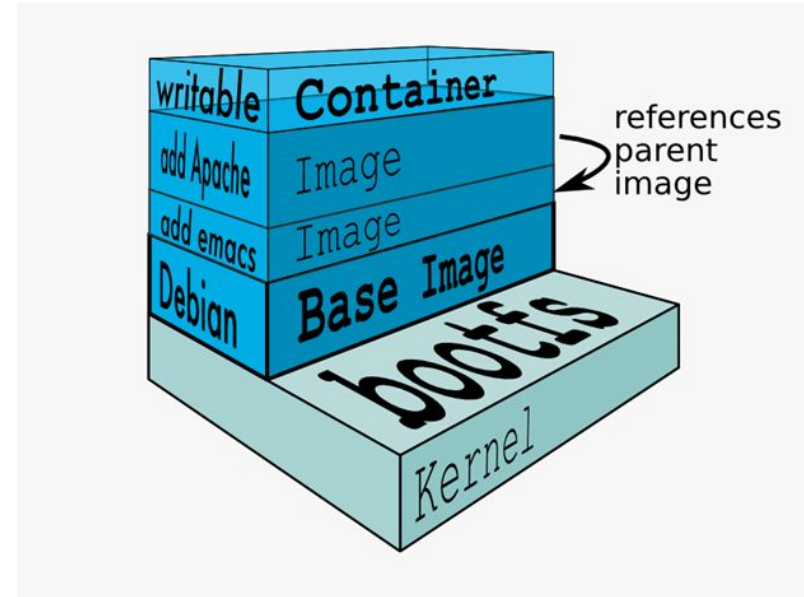
```
# Install additional packages  
RUN apk add --update emacs  
RUN apk add --update apache
```

```
# index.html must be in the current directory  
ADD index.html /home/demo/
```

```
# Define the command which runs when the container starts  
CMD ["cat /home/demo/index.html"]
```

```
# Use bash as the container's entry point. CMD is the argument to this entry  
point
```

```
ENTRYPOINT ["/bin/bash", "-c"]
```

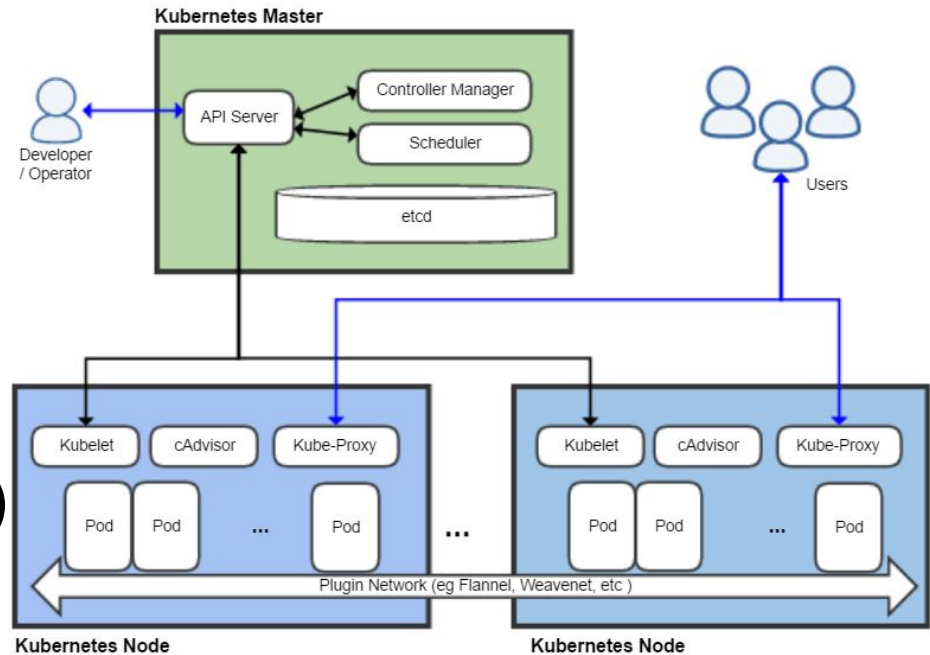


Images & Containers

- An image: is a static file; never changes
- A container: a live instance of an image
- Think of it this way – you have a DVD that installs Windows OS (image). After you install it, you can write files to it (container).
- `docker build`
 - builds an image
- `docker run`
 - runs a container based off of an image

Kubernetes

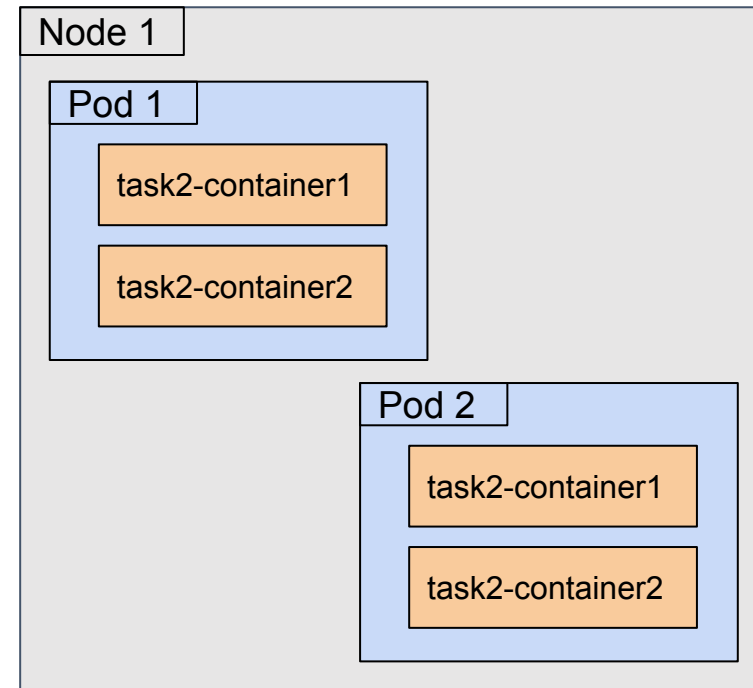
- [Kubernetes](#) is an open-source platform for automating deployment, scaling, and operations of application containers.
- Containers built using Docker
- Images stored in a private registry
- Application defined using a YAML (or JSON) template



Deploying with Kubernetes

- [kubectl](#) is the command that interacts with the Kubernetes Master (API Server)
- Pod Configuration [Reference](#)
 - `kubectl create -f demo_pod.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: test
  labels:
    app: test
spec:
  containers:
    - name: backend1
      image: us.gcr.io/cc-p22/task2-container1:v0
      ports:
        - name: http
          containerPort: 8080
  ...
```



Project 2.2 - Containers

- Build a service to compile and run user code submitted through a front end.
- Four tasks:
 - Task 1: Containerize an Nginx server and run container locally.
 - Task 2: Build a service to call two web services and return the sum. Deploy application to a Kubernetes cluster.
 - Task 3: Build a Python web service to evaluate Python code submitted from the UI and return the result.
 - Task 4: Multi-cloud deployment. Python and SQL code evaluation services. Multiple application + DB in the same pod.

Task 1 Objectives

- Work with Dockerfiles
- Master the Docker CLI, including useful commands like:
 - `docker build`
 - `docker images`
 - `docker run`
 - `docker ps`
- Think about integration between the host and the container

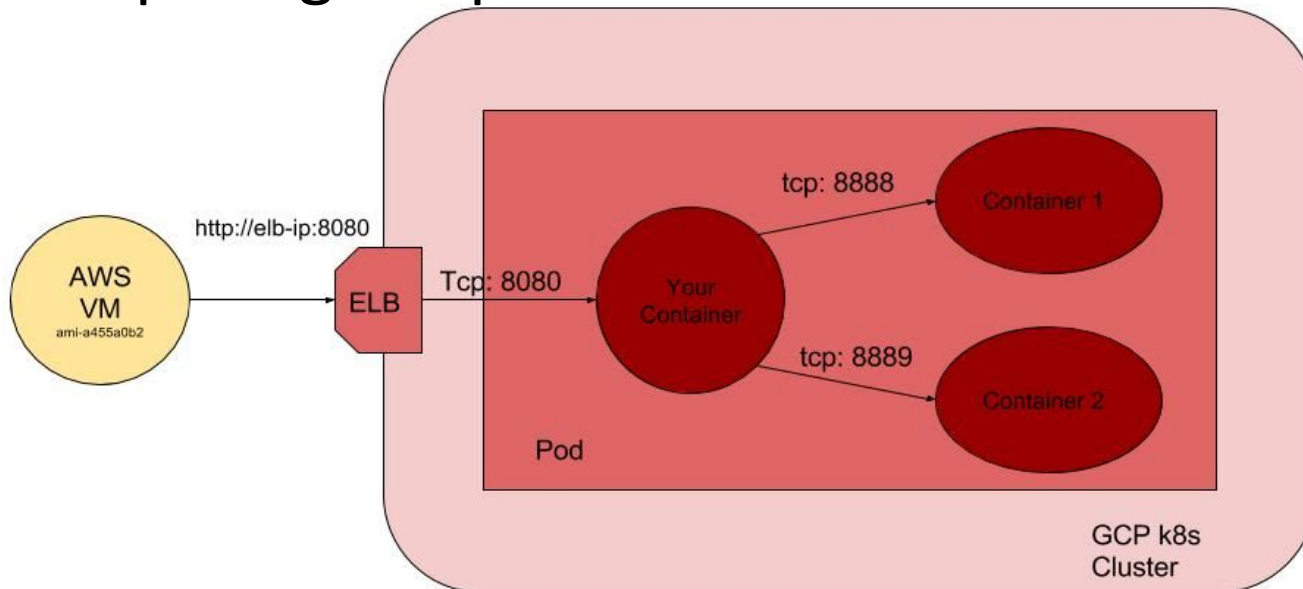
Task 1 Overview

- Configure a Docker container with an Nginx web server
- Nginx server listening on port 15619
- Port 15619 of host VM mapped to the container port



Task 2 Objectives

- Work with more complex Dockerfiles
- Define a Kubernetes YAML file for your application
- Deploying your application to the cluster
 - Multiple pods per container
 - Exposing the pod as a Service via ELB

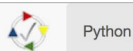
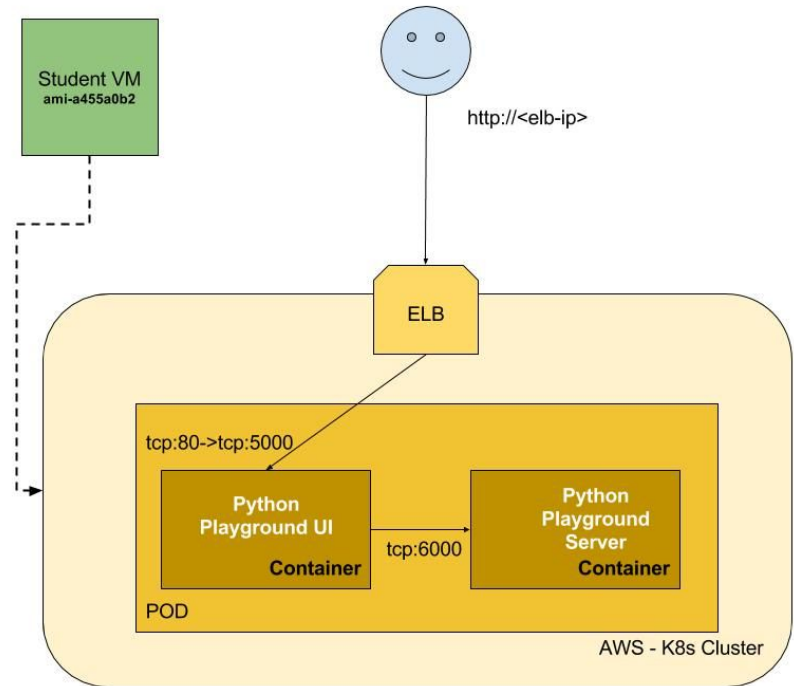


Task 3 Objectives

- Developing a code execution service
 - Accepts Python code from the UI (over HTTP)
 - Use the provided API specification to develop your application.
 - Service that consumes and produces JSON.
- Develop Kubernetes YAML definition
 - Application for UI, exposed as Service via LB
 - Backend service that is not exposed to the Internet, only accessible to the UI container

Task 3 Overview

- Python code evaluation service architecture.
- The UI is exposed on the internet, accepts POST requests.



Python

Python Code

```
1 # Welcome to the Python Explorer
2 import random;
3
4 firstRandom = random.randint(1, 100);
5 secondRandom = random.randint(1, 100);
6
7 sum = firstRandom + secondRandom;
8
9 print ("The first random number is: " + str(firstRandom));
10 print ("The second random number is: " + str(secondRandom));
11
12 print ("The sum of the random numbers is: " + str(sum));
13
```

Output

```
1 The first random number is: 34
2 The second random number is: 74
3 The sum of the random numbers is: 108
4
```

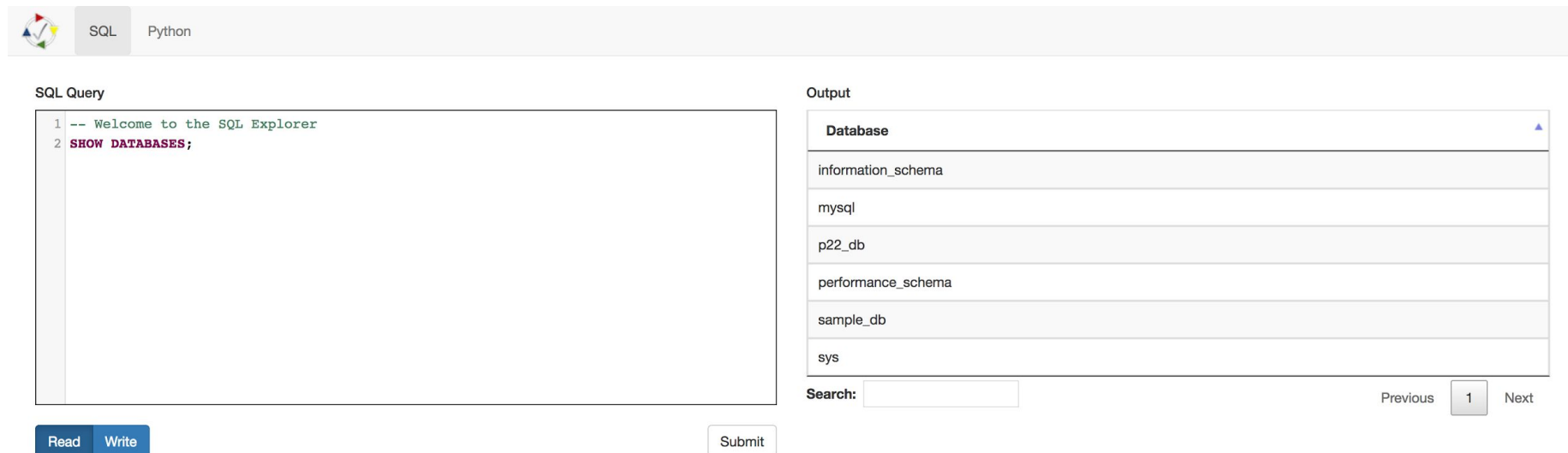
Submit

Task 4 Objectives

- Task 4 will build on task 3 by introducing a SQL evaluation service.
 - Deploy a Java REST service and MySQL database to your Kubernetes cluster.
- Achieve high availability!
 - Multi cloud deployment!
 - The Python evaluation service will be replicated across multiple clouds.

Task 4 Overview

- Multi-cloud application deployments.
 - High availability! Python evaluation service in GCP and Azure.
- UI will handle SQL and Python code samples.



The screenshot displays a web-based SQL Explorer interface. At the top, there is a navigation bar with a logo on the left and two tabs: 'SQL' (which is active) and 'Python'. Below the navigation bar, the interface is split into two main sections. The left section, titled 'SQL Query', contains a text area with two lines of code: `1 -- Welcome to the SQL Explorer` and `2 SHOW DATABASES;`. Below the text area are two buttons: 'Read' and 'Write'. The right section, titled 'Output', displays the results of the query in a table format. The table has a single column labeled 'Database' and lists the following databases: 'information_schema', 'mysql', 'p22_db', 'performance_schema', 'sample_db', and 'sys'. Below the table is a search bar with the label 'Search:'. At the bottom right of the interface, there are navigation controls: 'Previous', a page number '1' inside a box, and 'Next'. A 'Submit' button is also located at the bottom center of the interface.

SQL Explorer

SQL Python

SQL Query

```
1 -- Welcome to the SQL Explorer
2 SHOW DATABASES;
```

Read Write

Submit

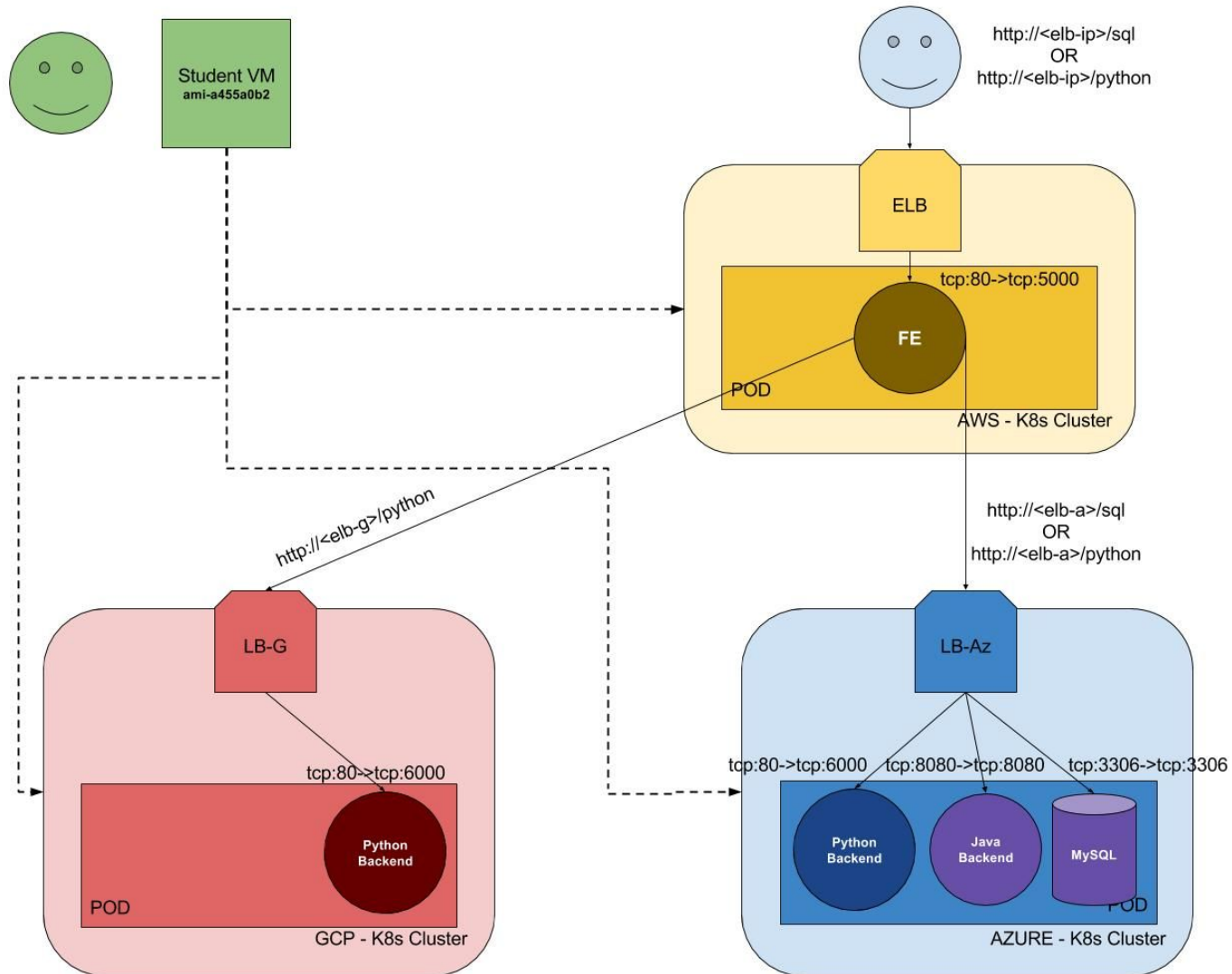
Output

Database
information_schema
mysql
p22_db
performance_schema
sample_db
sys

Search:

Previous 1 Next

Task 4 Architecture



Tips, Trips, and Tricks

- Debug, debug, debug
 - This project has many moving pieces!
 - Where is the issue occurring?
 - What is the expected behaviour of the system?
- Pods and Logs
 - Did my pod start? `(kubectl get pods , kubectl describe pods)`
 - Is my pod generating any logs? `(kubectl logs ...)`

Project 2.2 Penalties




Danger

Project Grading Penalties

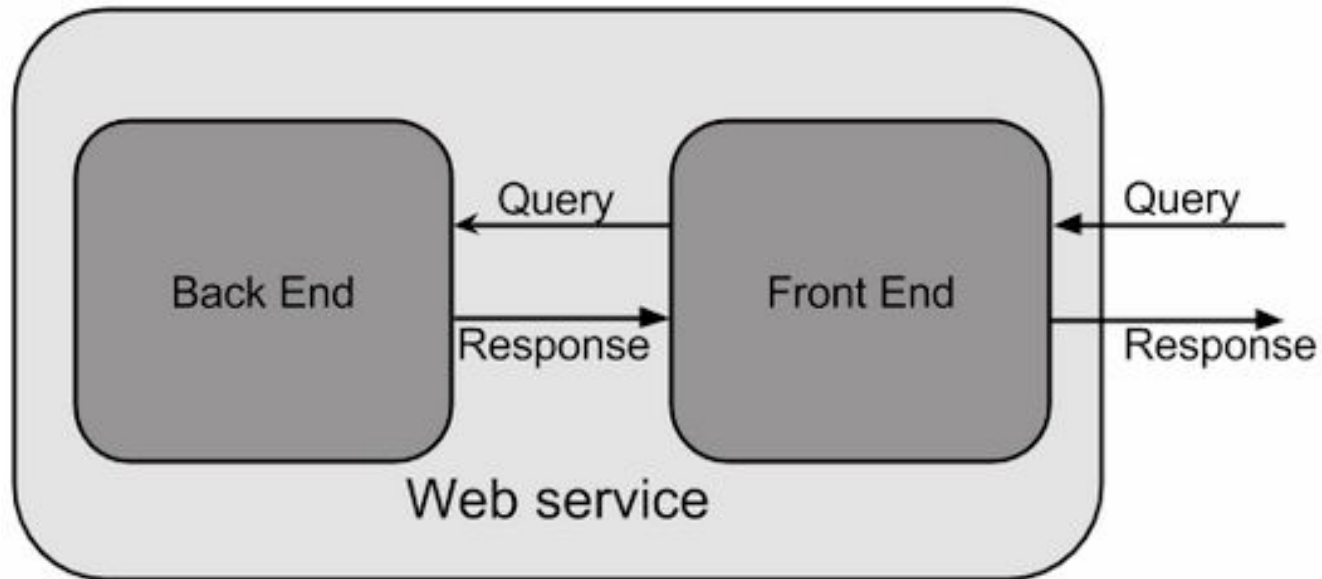
Violation	Penalty of the project grade
Spending more than \$5 for this project on AWS	-10%
Spending more than \$10 for this project on AWS	-100%
Failing to tag all your resources in any task (EC2 instances, ELB) for this project; Key: Project and Value: 2.2	-10%
Submitting your AWS, GCP, or Azure credentials, other secrets, or Andrew ID in your code for grading	-100%
Submitting executables (.jar , .pyc , etc.) instead of human-readable code (.py , .java , .sh , etc.)	-100%
Attempting to hack/tamper the autograder in any way	-100%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200% & potential dismissal

Upcoming Deadlines



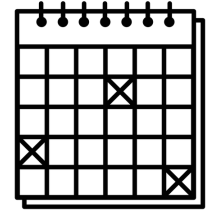
- Quiz 4: Modules 7, 8 and 9:
Due: **Friday February 17, 2017 11:59PM Pittsburgh** 
- Project 2.2: Docker and Kubernetes
Due: **Sunday February, 19 2017 11:59PM Pittsburgh** 
- Team Project: Team Formation
Due: **February, 20 2017 11:59PM Pittsburgh** 

Team Project Architecture



- Writeup and Queries will be released on Monday, **February 27th, 2017**
- We can have more discussions in subsequent recitations
- For now, ensure 3-person teams you decide have experience with web frameworks and database, storage principles and infra setup/hacking

Team Project Time Table



Phase (and query due)	Start	Deadline	Code and Report Due
Phase 1 • Q1, Q2, Q3	Monday 02/27/2017 00:00:00 ET	Sunday 03/26/2017 23:59:59 ET	Tuesday 03/28/2017 23:59:59 ET
Phase 2 • Q1, Q2, Q3, Q4	Monday 03/27/2017 00:00:00 ET	Sunday 04/16/2017 15:59:59 ET	
Phase 2 Live Test (Hbase/MySQL) • Q1, Q2, Q3, Q4	Sunday 04/16/2017 18:00:00 ET	Sunday 04/16/2017 23:59:59 ET	Tuesday 04/18/2017 23:59:59 ET
Phase 3 • Q1, Q2, Q3, Q4, Q5	Monday 04/17/2017 00:00:00 ET	Sunday 04/30/2017 15:59:59 ET	
Phase 3 Live Test • Q1, Q2, Q3, Q4, Q5	Sunday 04/30/2017 18:00:00 ET	Sunday 04/30/2017 23:59:59 ET	Tuesday 05/02/2017 23:59:59 ET

Demo

- Configure simple containerized application
- Deploy to kubernetes cluster