15-319 / 15-619
Cloud Computing

Recitation 5

February 12th, 2019

Administrative - OH & Piazza

- Make use of office hours
  - Make sure that you are able to describe the problem and what you have tried so far
  - Piazza Course Staff
  - Google calendar in ET
  - Google calendar in PT

- Suggestions for using Piazza
  - Contribute questions and answers
  - Read the Piazza Post Guidelines (@6) before asking questions
  - Read Piazza questions & answers carefully to avoid duplicates
  - Don’t ask a public question about a quiz question
  - Try to ask a **public** question if possible

Administrative - Cloud spending

- Suggestion on cloud service usage
  - Monitor AWS expenses regularly
  - Always do the cost calculation before launching services
  - Terminate your instances when not in use
  - Stopped instances have EBS costs ($0.1/GB-Month)
  - Make sure spot instances are tagged right after launch

Important Notice

- **DON’T EVER EXPOSE YOUR AWS CREDENTIALS!**
  - Github
  - Bitbucket
  - Anywhere public…
- **DON’T EVER EXPOSE YOUR GCP CREDENTIALS!**
- **DON’T EVER EXPOSE YOUR Azure CREDENTIALS!**
  - ApplicationId, ApplicationKey
  - StorageAccountKey, EndpointUrl

Reflection of Last Week

- Conceptual content on OLI
  - Modules 5, 6 and Quiz 3
- Project theme - **Horizontal Scaling and Advanced Resource Scaling**
  - **AWS Horizontal Scaling**
    - Launch cloud resources via the AWS APIs (EC2)
    - Horizontally scale instances to reach a target RPS
  - **AWS Autoscaling**
    - Launch cloud resources via the AWS APIs (ALB / ASG…)
    - Design autoscaling policies to achieve RPS targets within instance hour limits
    - Handle instance failures
  - **AWS Autoscaling with Terraform**
    - Develop a Terraform template to launch cloud resources
    - Contrast infrastructure as code (IaC) and cloud APIs

This Week

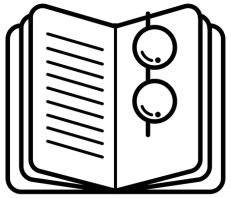
- **Code Review - Project 1.2**
  - Due on **Wednesday**, Feb 13th, 2019, 11:59PM ET
- **Online Programming Exercises**
  - Attend your scheduled session!
- **Quiz 4 (OLI Modules 7, 8 & 9)**
  - Due on **Friday**, Feb 15th, 2019, 11:59PM ET
- **Project 2.2**
  - Due on **Sunday**, Feb 17th, 2019, 11:59PM ET

- **Primers released this week**
  - P2.3 - Introduction to Cloud Functions

Guideline for Project Reflection

- Describe your approach in solving each task in this project
  - Please share your:
    - Approach to solving each task in the project
    - What you would do differently if you could do the project over again
    - Interesting problems that your overcame
  - However, please:
    - Do not share your code or pseudocode
    - Do not share details about your solution

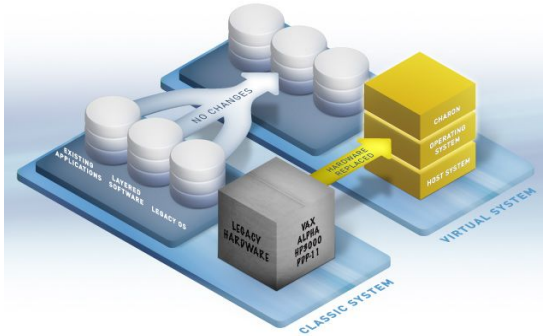
This Week: Conceptual Content



- OLI, UNIT 3: Cloud Infrastructure
  - 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: Storage and Network Virtualization

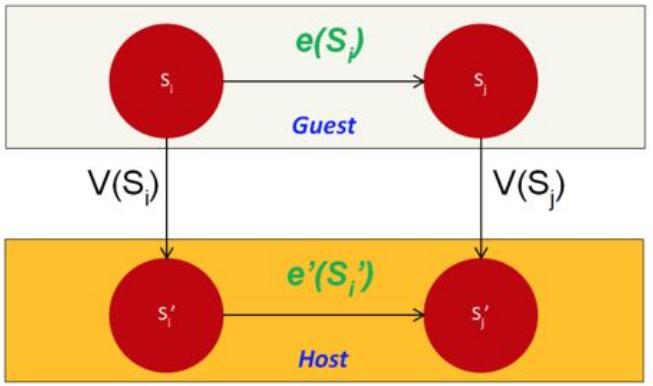
OLI Module 7 - Virtualization Introduction and Motivation

- Why virtualization?
  - Elasticity
  - Resource sandboxing
  - Mixed OS environment
  - Resource sharing
  - Improved system utilization and reduced costs



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(Si)*

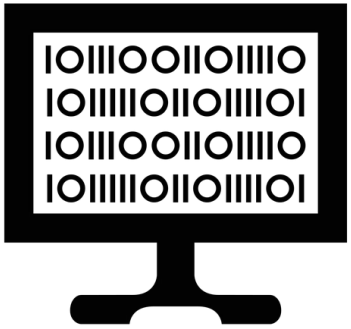


- Virtual Machine Types
  - Process Virtual Machines
  - System Virtual Machines

OLI Module 9
Resource Virtualization - CPU

- Steps for 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

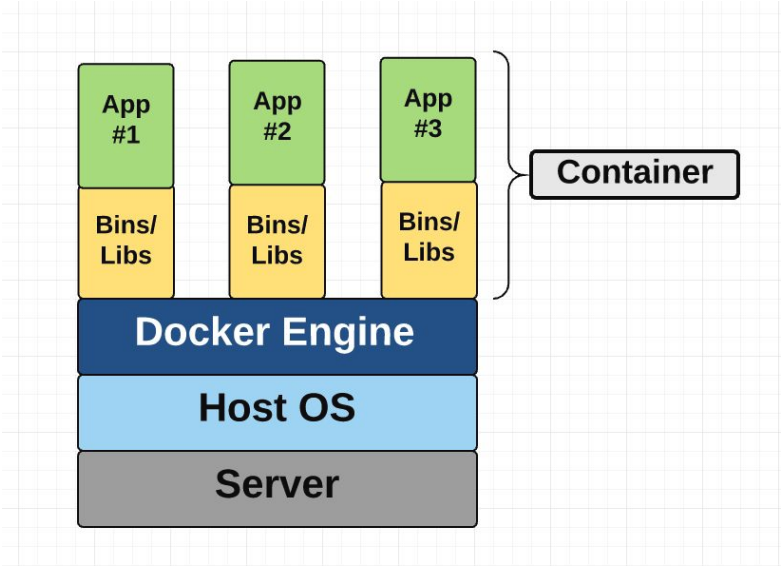
This Week's Project



- P2.1: Horizontal Scaling and Autoscaling
  - Horizontal scaling in / out using AWS APIs
  - Load balancing, failure detection, and cost management on AWS
  - Infrastructure as Code (Terraform)
- P2.2: Docker Containers and Kubernetes
  - Building your own container-based microservices
  - Docker containers
  - Manage multiple Kubernetes Cluster
  - Multi Cloud deployments
- P2.3: Functions as a Service
  - Develop event driven cloud functions
  - Deploy multiple functions to build a video processing pipeline

Containers

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

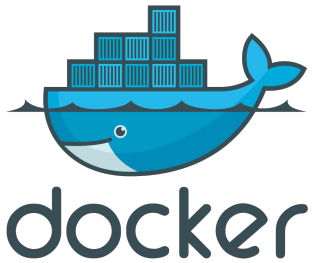


Why Containers?

- Faster deployment
- Portable
- Modularity
- Consistent Environment

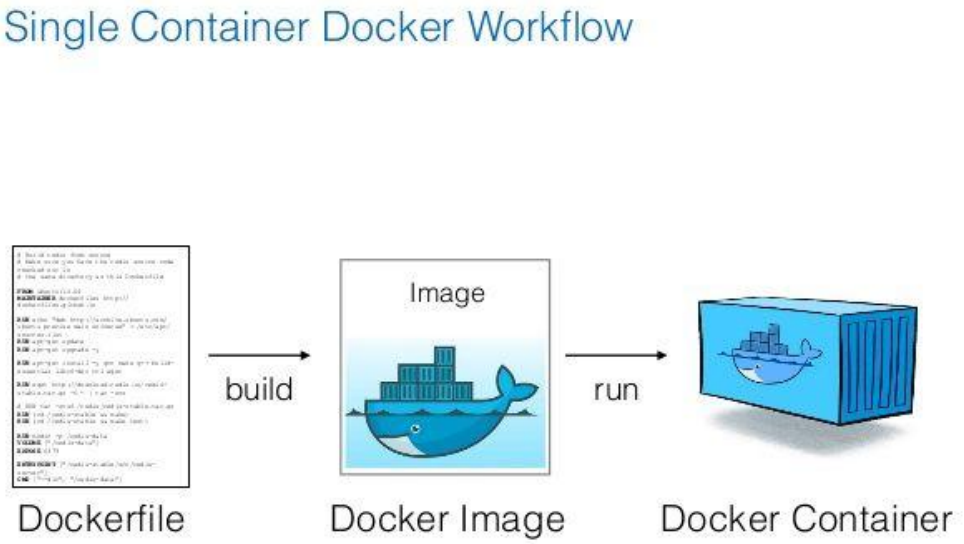
Build once, run anywhere

Docker

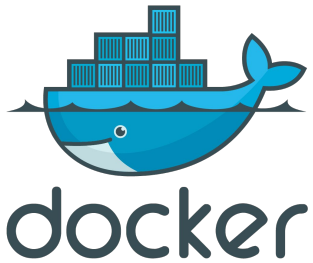


- Docker is an open platform for developing, shipping, and running applications.

- Dockerfile
- Docker Image
- Docker Container



Dockerfile



- Dockerfile tells Docker how to build an **image**:
  - Base Image
  - Commands
  - Files
  - Ports
  - Startup Command
- In short, a Dockerfile is a recipe for Docker images

Let's go through a sample Dockerfile!

Example Dockerfile

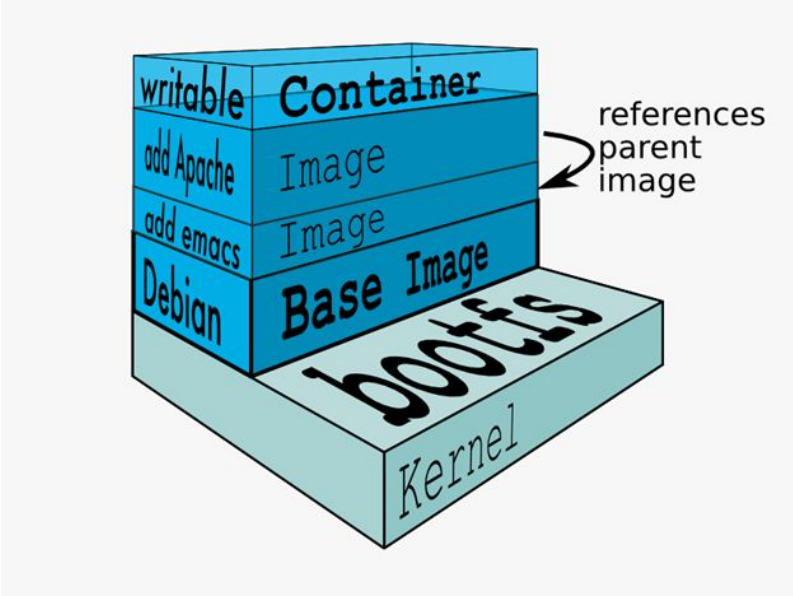
```
# Debian as the base image
FROM debian:latest

# 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

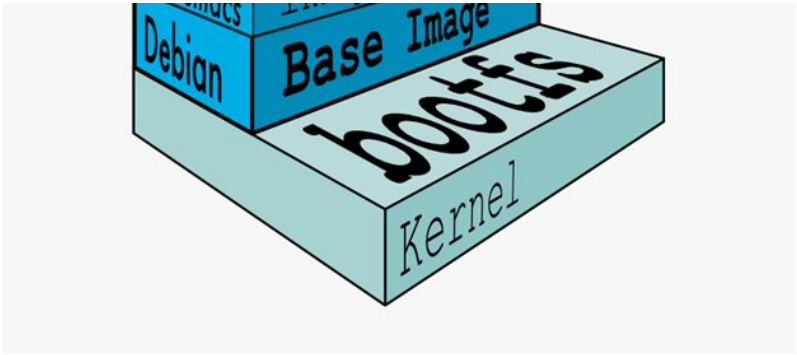
```
# Debian Linux as the base image
FROM debian:latest

# 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

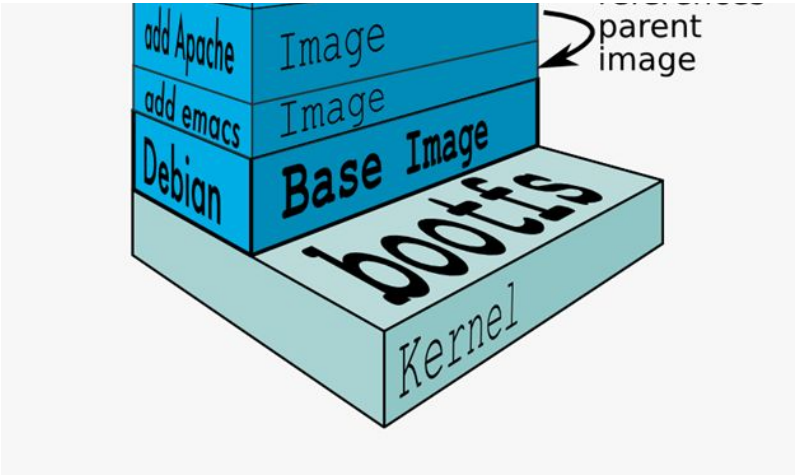
```
# Alpine Linux as the base image
FROM debian:latest

# 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

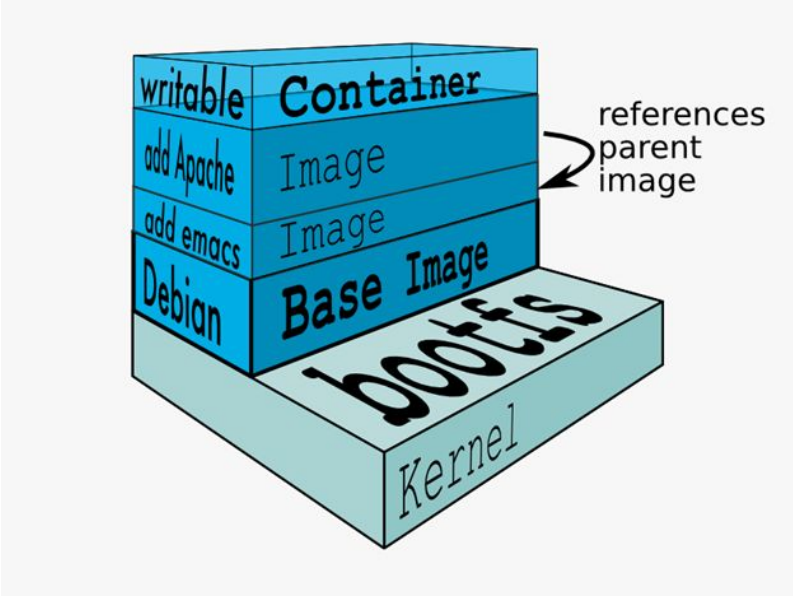
```
# Alpine Linux as the base image
FROM debian:latest

# 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"]
```

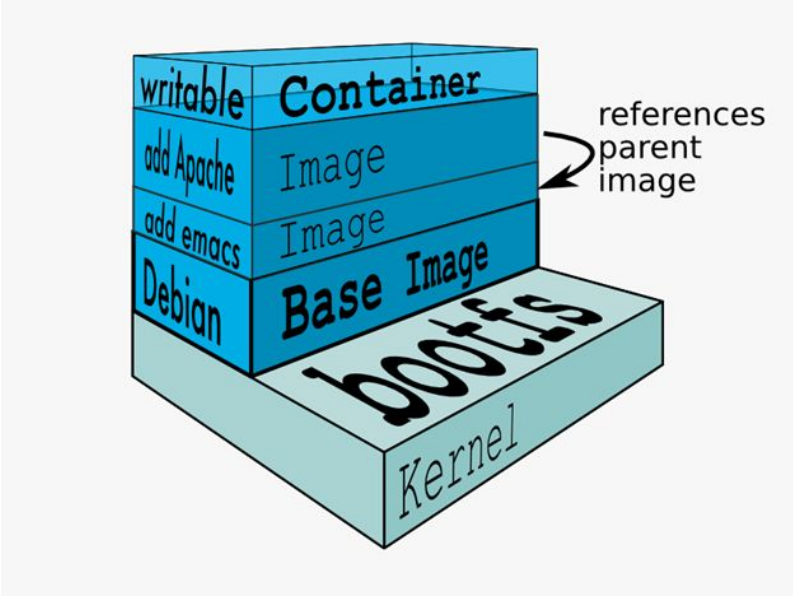


Example Dockerfile

```
# 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"]
```

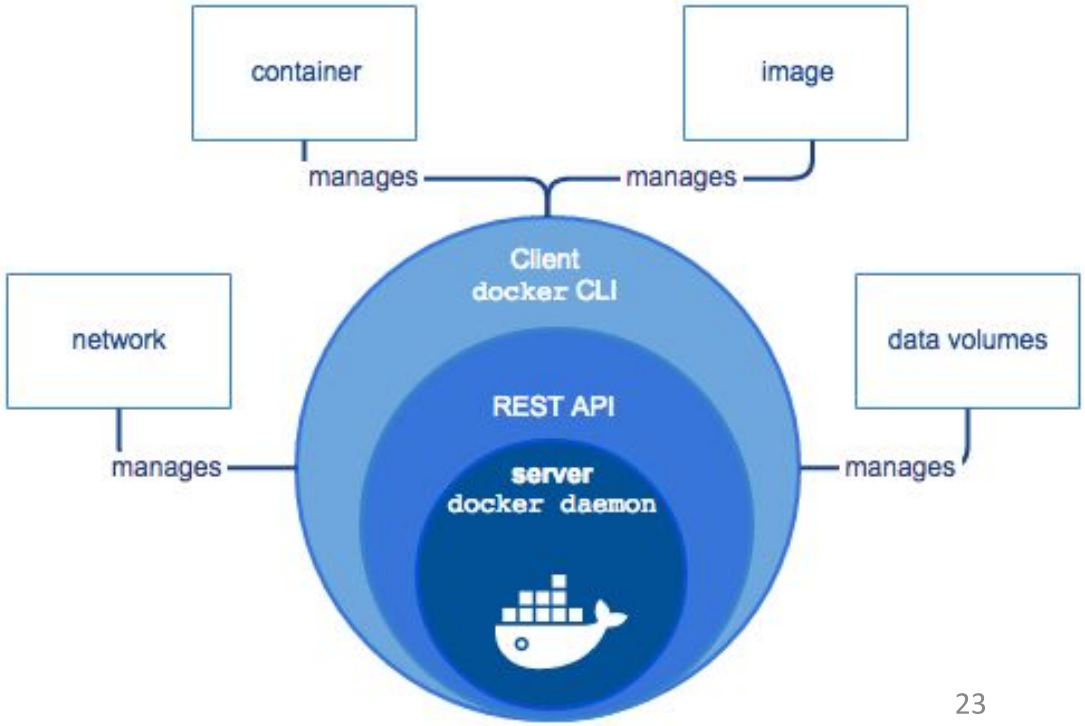
Images & Containers

- `docker build`
  - Builds an image
- `docker run`
  - Runs a container based on an image

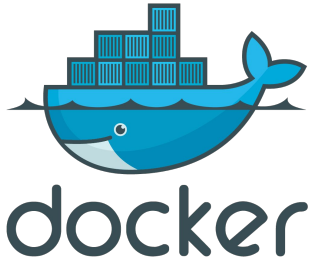
- Images are immutable (Like a Class)
  - View these with `docker images`
- Containers are a ‘running instance of an Image’ (Like an Object)
  - View these with `docker ps`

Docker Engine

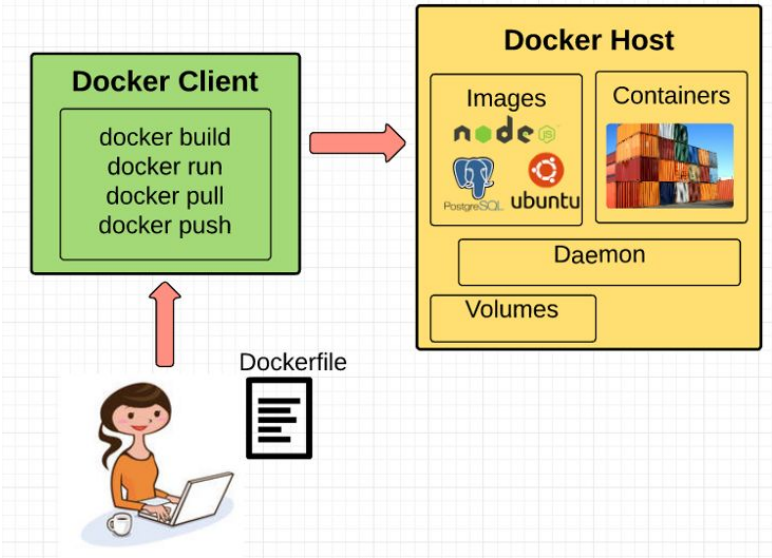
- A client-server application
  - Docker Daemon
  - Docker CLI
  - REST API



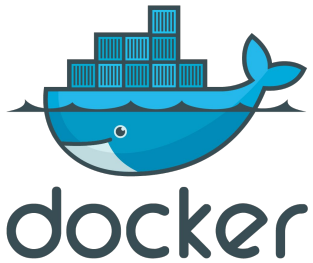
Docker Daemon



- Listens for Docker API requests
- Manages Docker objects
- The Daemon does not have to be on the same machine as the Client



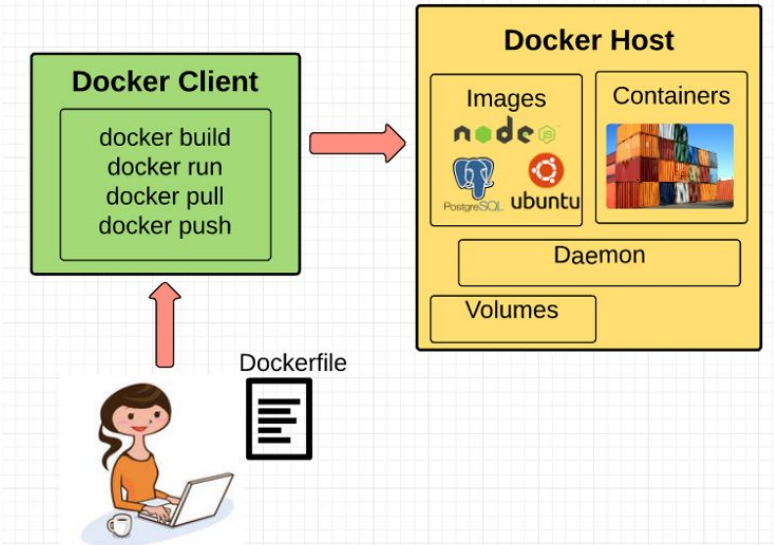
Docker CLI



- Communicates with Daemon using an API
- When you type:

```
docker build nginx
```

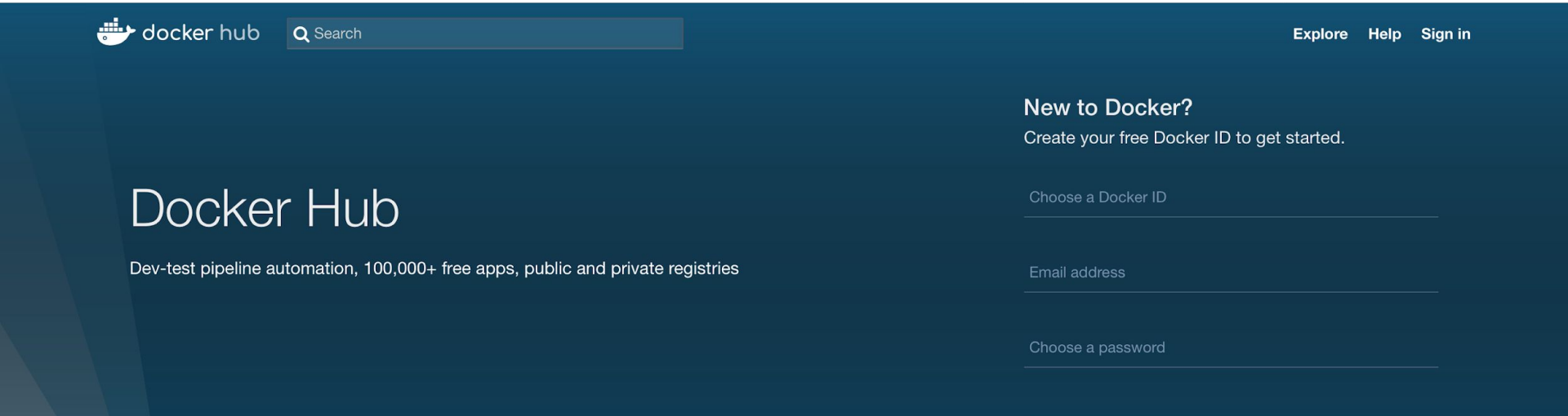
You are telling the Docker client to forward the `build nginx` instruction to the Daemon



Docker Registries

- Store Docker images
- Examples
    - Docker Hub and Docker Cloud
    - GCP Container Registry

- `docker pull`
- `docker push`



Containers are Useful, but What About Resource Management?

- Containers can do many good things for us
- However, what do you still have to do manually?
  - Load Balancing
  - Fault Tolerance
- How should we do this?

Kubernetes

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers.
  - Horizontally Scalable
  - Self-Healing
  - Service Discovery
  - Automated Rollbacks
  - Utilization

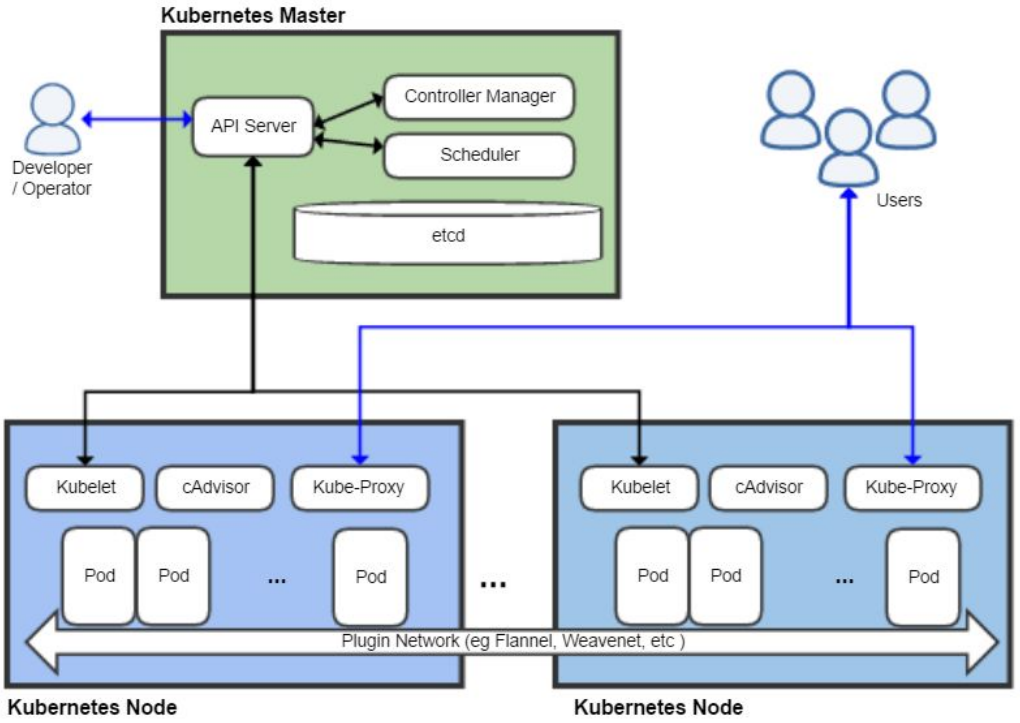
kubernetes

Kubernetes Overview

- API Objects
  - Pods - Collection of Containers
  - Deployment - Manages Pods
  - Service - Network Endpoint
- Desired State Management
  - YAML (YAML Ain’t a Markdown Language)
- Kubectl - CLI for Kubernetes
  - `kubectl create config.yaml`

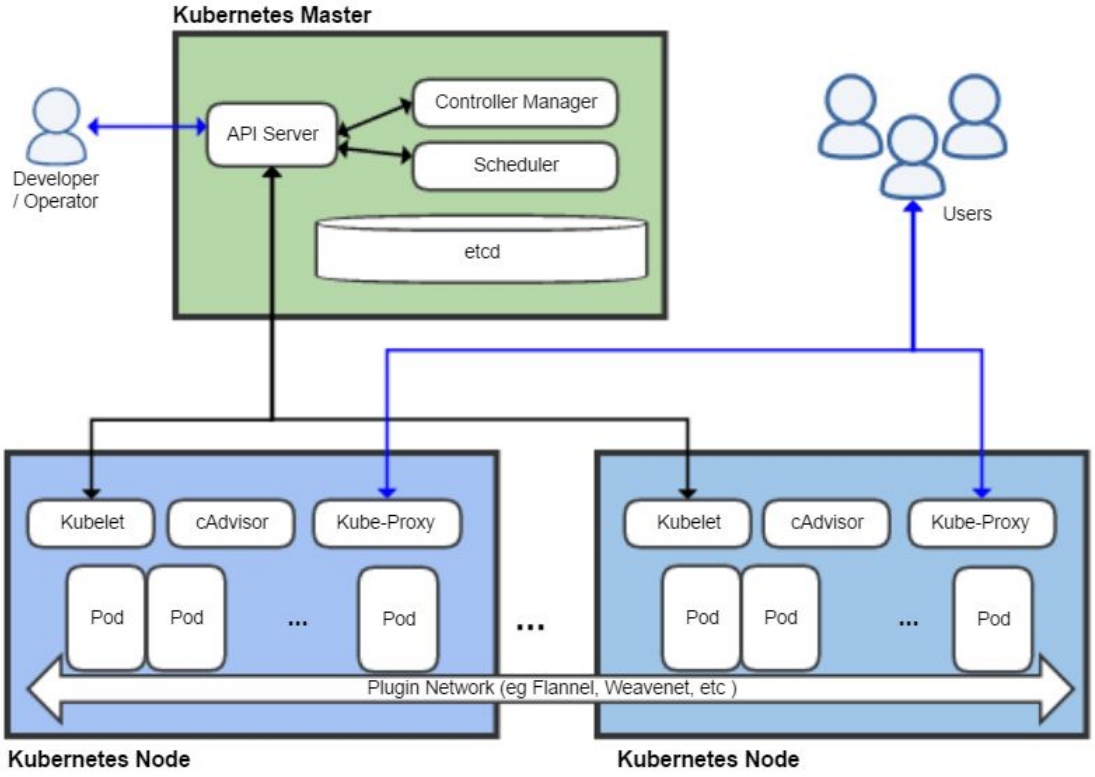
Kubernetes Cluster - Master

- Master Node
  - API Server
  - Controller Manager
  - Scheduler



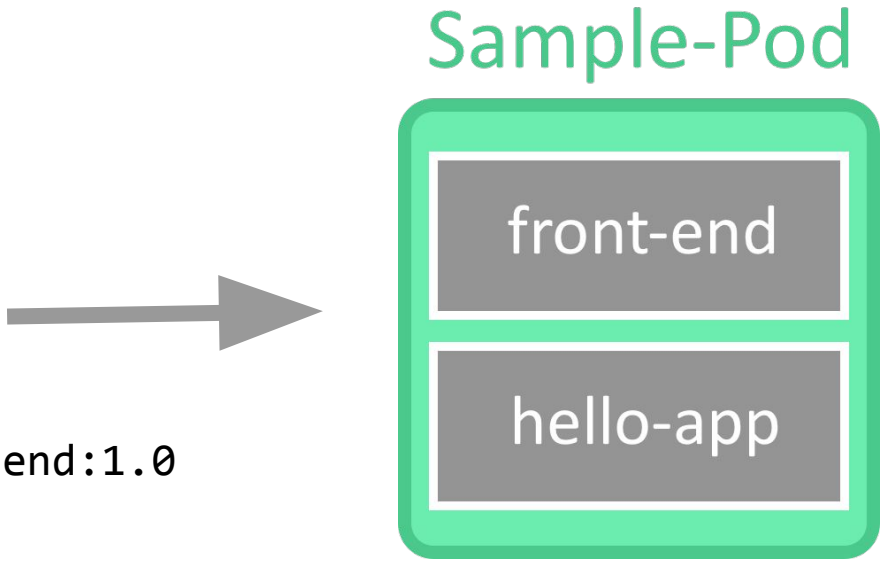
Kubernetes Cluster - Worker

- Worker Nodes
  - Kubelet Daemon
  - Kube-Proxy



Sample Kubernetes Config YAML

```
apiVersion: apps/v1beta1
kind: Pod
metadata:
 name: Sample-Pod
 labels:
   app: web
spec:
 containers:
   - name: front-end
     image:
gcr.io/samples/hello-frontend:1.0
     ports:
       - containerPort: 80
   - name: hello-app
     image:
gcr.io/samples/hello-app:1.0
     ports:
       - containerPort: 8080
```

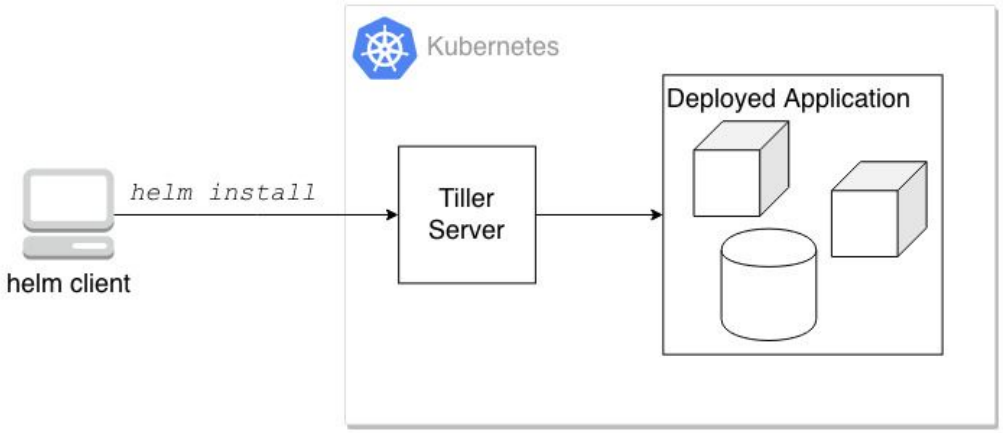


Helm

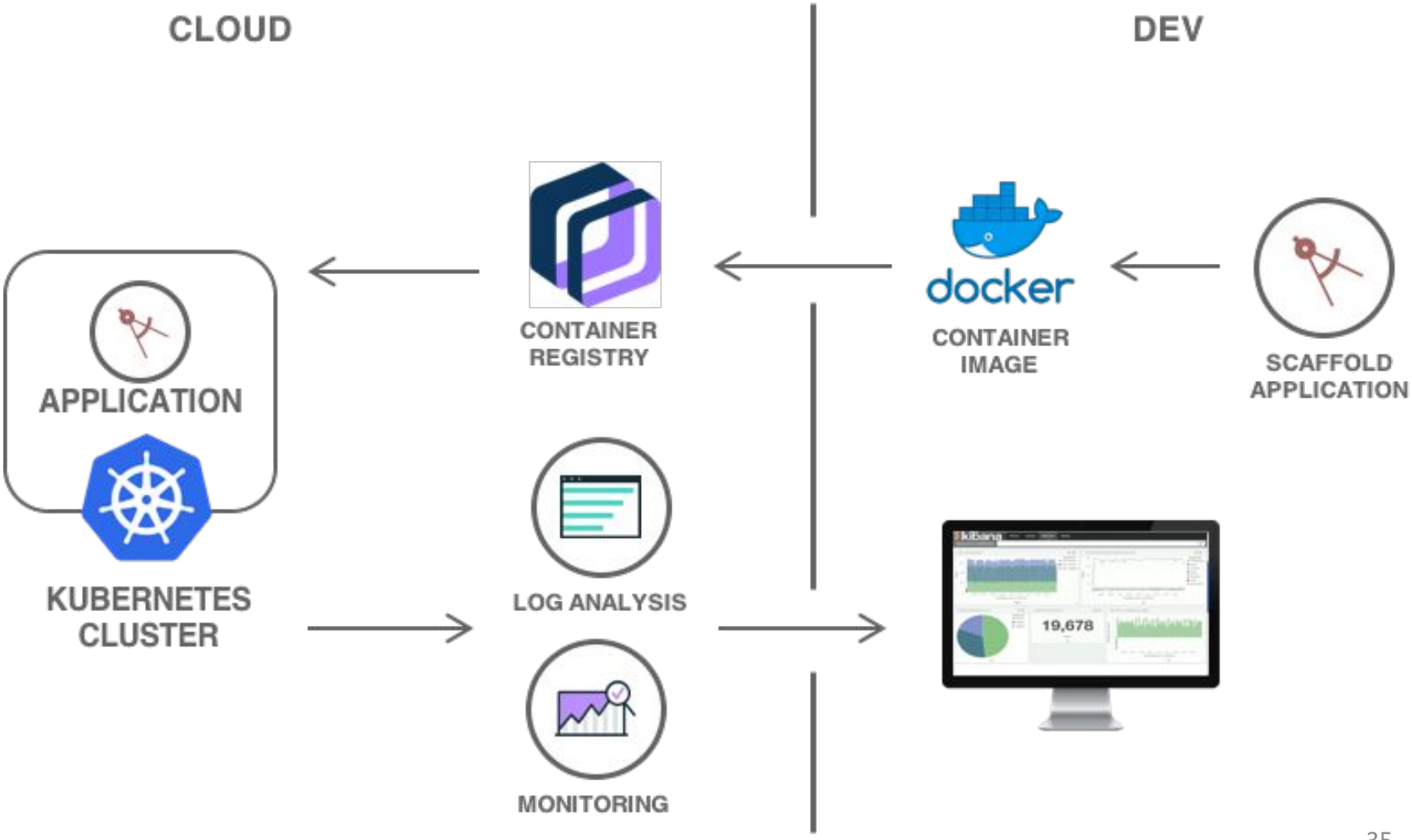
- A tool for managing Kubernetes applications
- Helm Charts help you define, install, and upgrade complex Kubernetes application
- Chart structure:
  - **Chart.yaml**
    - A YAML file that contains chart information (name, version, description, etc.)
  - **Values.yaml**
    - The default configuration of this chart. The values listed in this file will be substituted in the files under the templates/ directory.
  - **templates/**
    - A directory of template files that will be combined with the values defined in Values.yaml. The files under this directory will be used to define all of the Kubernetes objects required to deploy the application.

Helm - Architecture

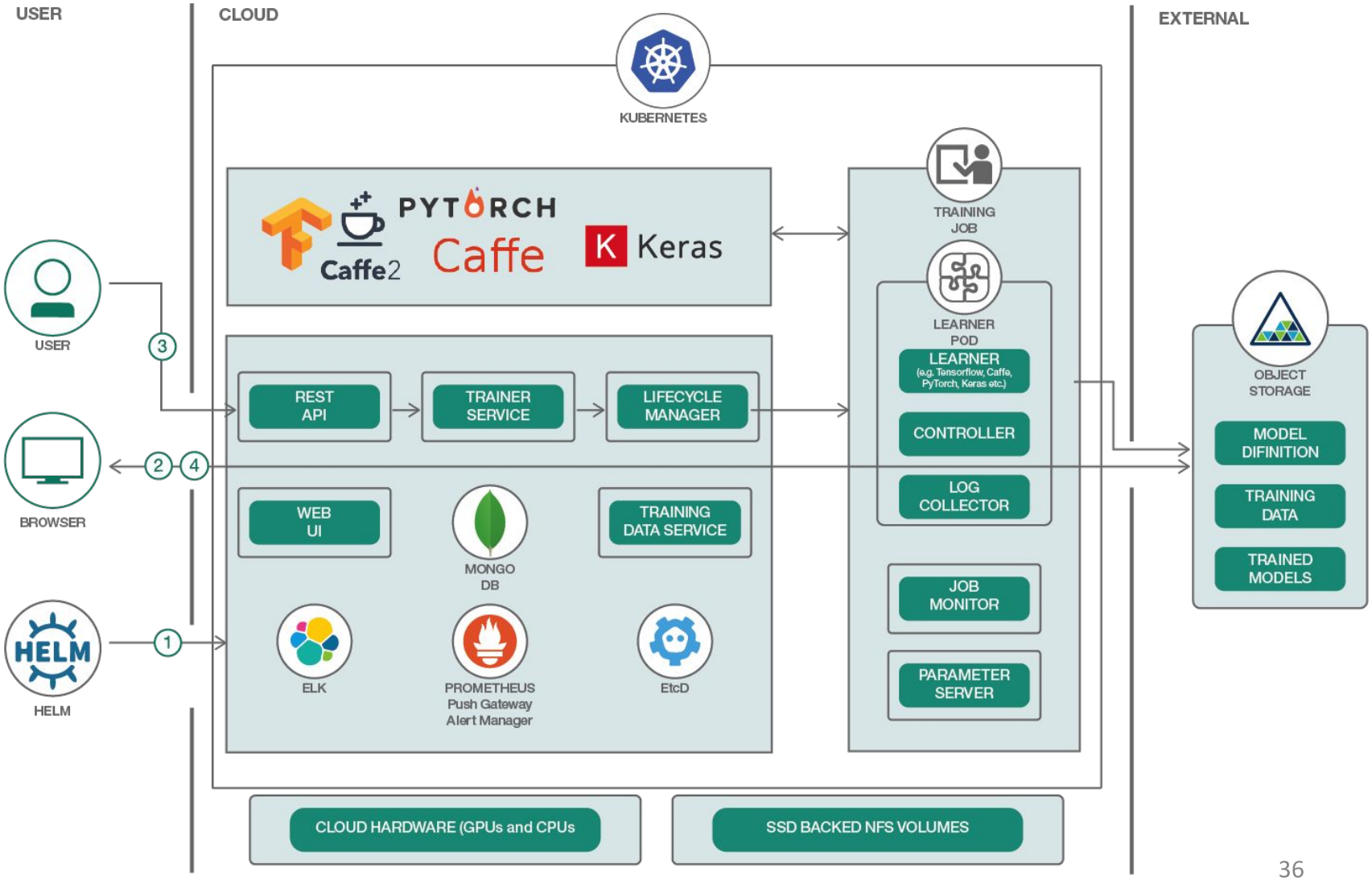
- Helm has two primary components:
  - **Helm client**
    - A command line tool that enables users to develop charts, manage repositories
    - Communicates with Tiller to install, describe or upgrade a release
  - **Tiller server**
    - Communicates with the Kubernetes API to manage the state of your application (e.g., installing, upgrading, or removing applications)



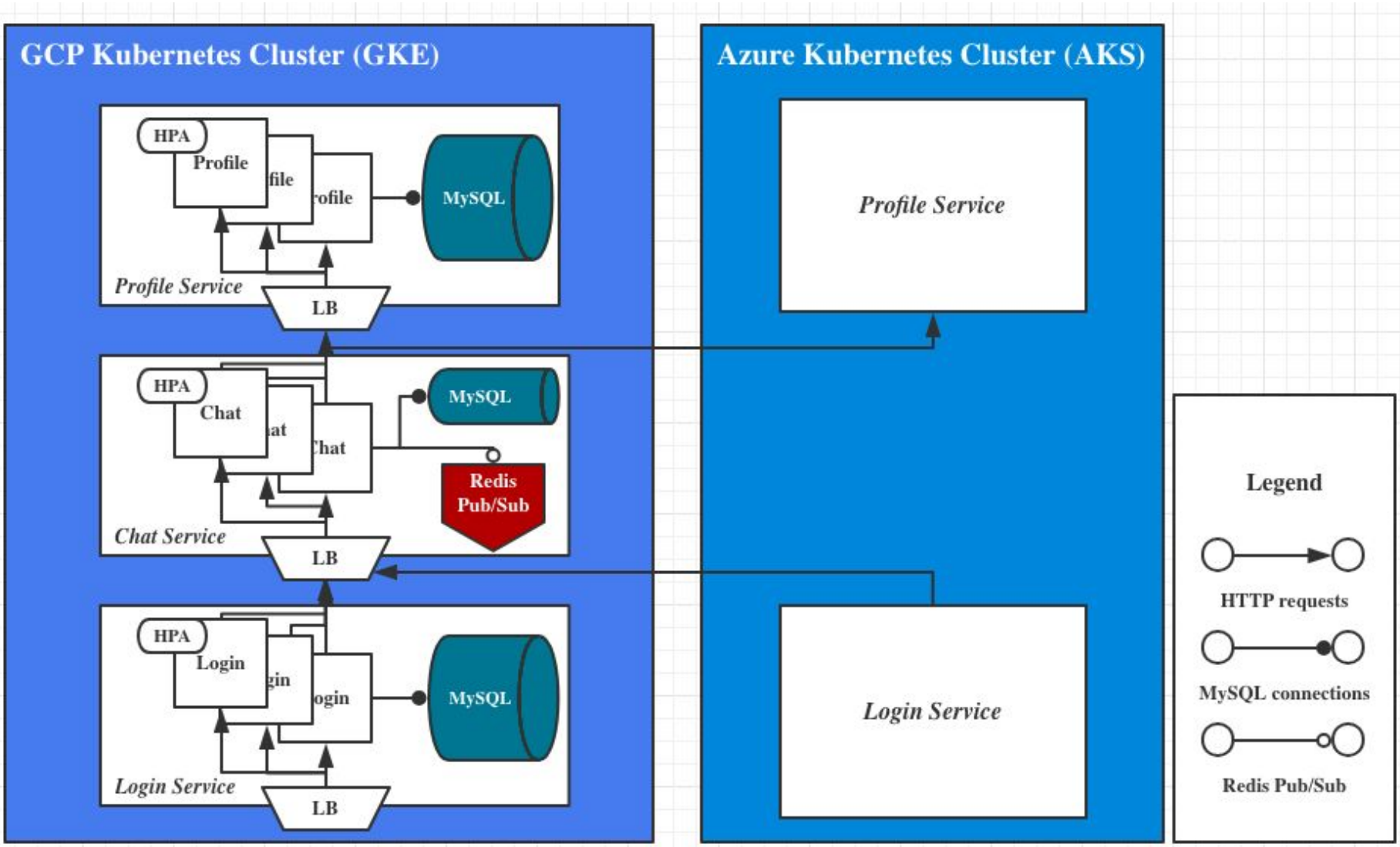
Docker, Kubernetes Workflow



An Industrial Example



Project 2.2 - Containers & Kubernetes



Project 2.2 - Containers & Kubernetes

- Build a chat room application using the microservice pattern
- Project overview:
  - Task 1: Containerize the profile service and run it locally
  - Task 2: Deploy the profile service to GKE
  - Task 3: Migrate the profile service's database from H2 to MySQL. Use Helm to manage the Kubernetes application.
  - Task 4: Install the chat service and login service using Helm charts. Connect the microservies to build an application.
  - Task 5: Replicate the profile and login services to AKS. Implement autoscaling rules to horizontally scale pods.

Task 1 - Containerize Profile Service

- Introduction to Dockerfiles
- Become familiar with the Docker CLI
  - `docker build`
  - `docker images`
  - `docker run`
  - `docker ps`
- Containerizing Java applications (a REST service)
- Consider interactions between the host machine and the container

Task 1 - Containerize Profile Service

- Run a Docker container to host the profile service
  - The Profile service exposes port 8080 on the container
  - Port 8000 of VM is mapped to the container port

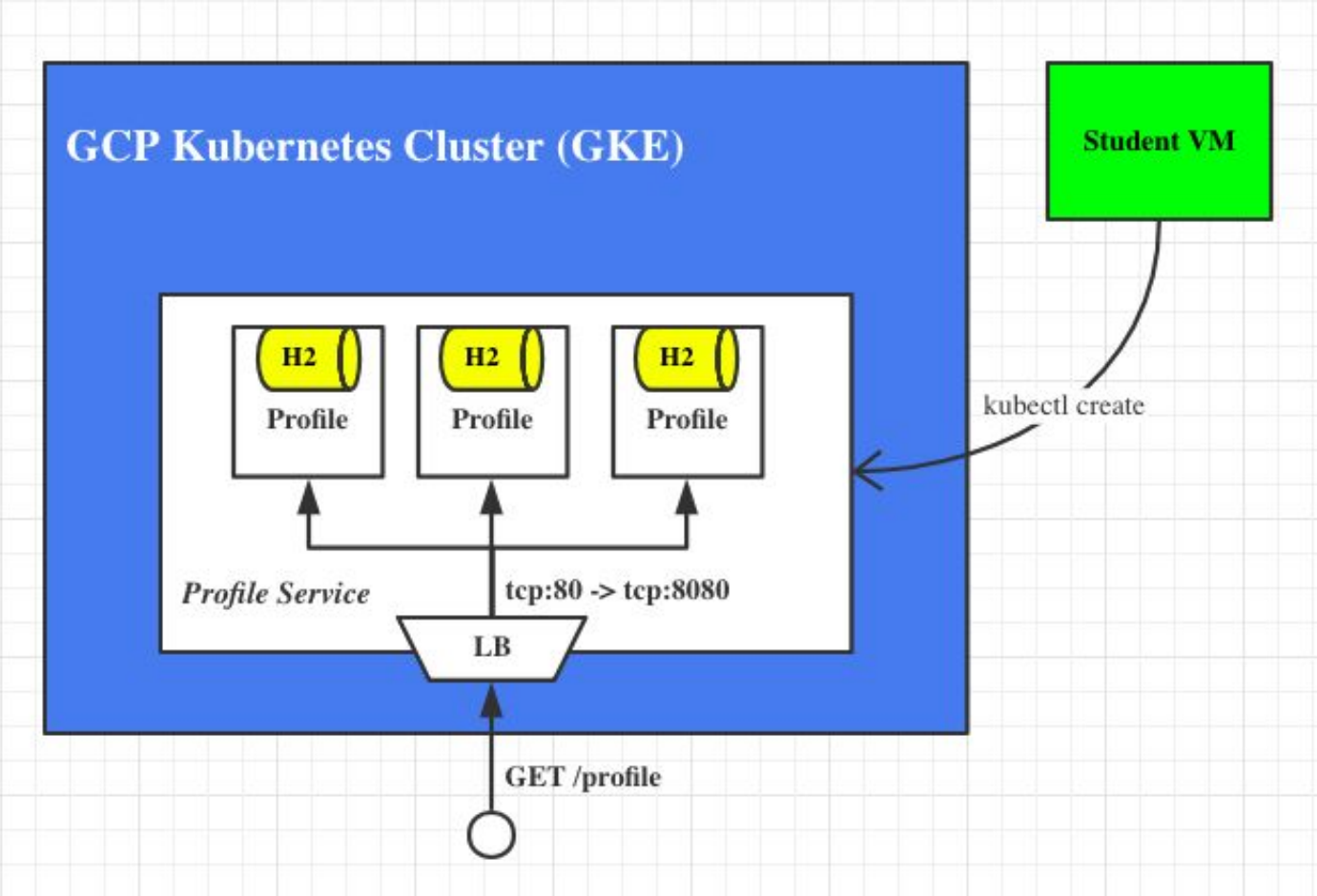
- How do we achieve this port mapping?

Task 2 - Using GCR and GKE to Deploy the Profile Service

- Push your image to a private registry
  - Push the profile service Docker image to Google Container Registry (GCR)
- Define a Kubernetes YAML configuration to
  - Create a deployment based on the image pushed to GCR
  - Expose the profile service via a (GCP) load balancer

Task 2 - Using GCR and GKE to Deploy the Profile Service

- Profile service architecture
- The backend application accepts **GET** requests at `/profile`
- The load balancer will map port 80 to port 8080



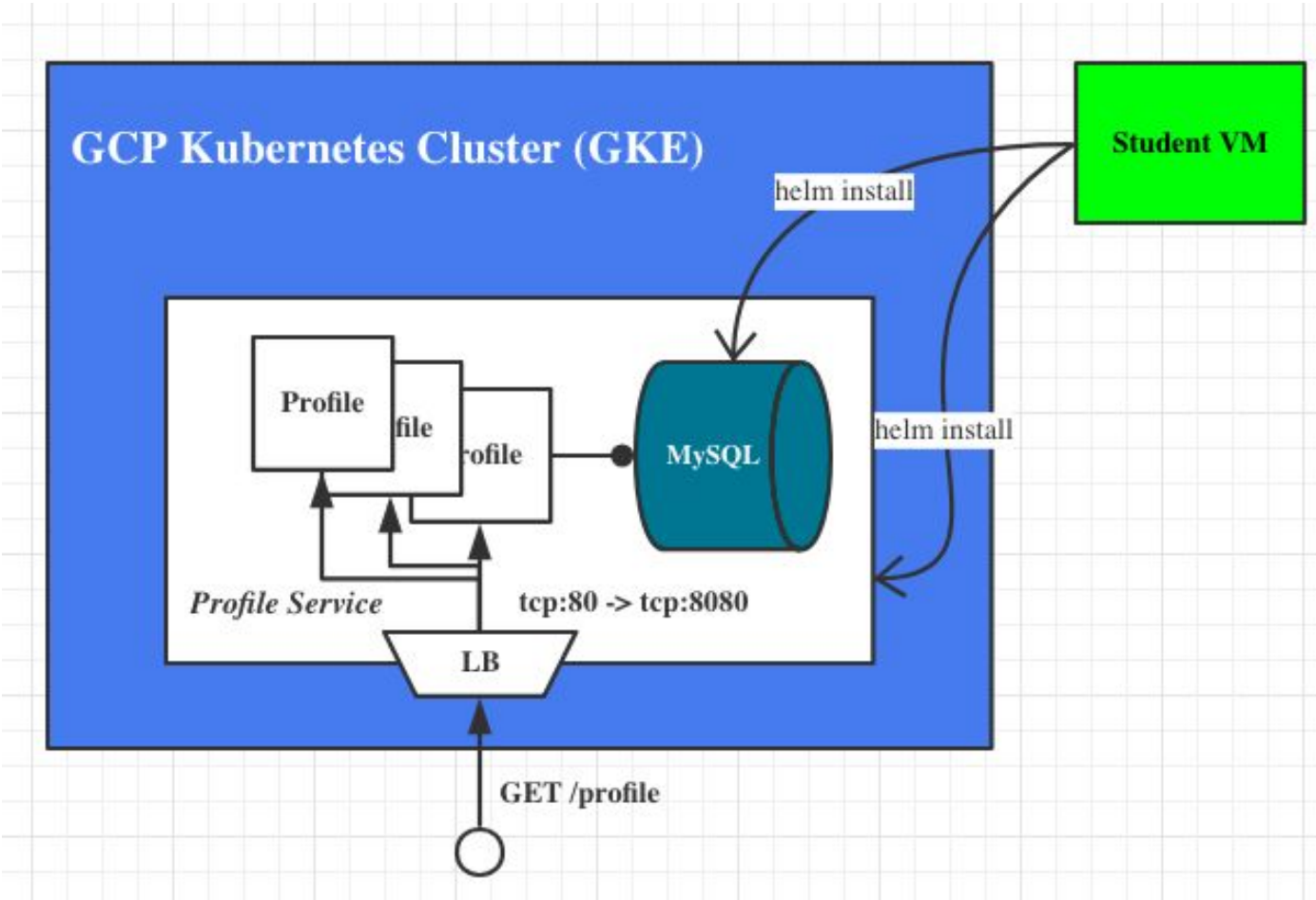
Task 3 - Introduction to Helm Charts

- Deploy a MySQL database using Helm
  - Update the profile service to use MySQL instead of the embedded H2 database
  - Remember to push your updated image to GCR!

- Develop a Helm chart for the profile service
  - Release the profile service via helm

Task 3 - Use Helm Charts and Migrating to MySQL

- Profile service architecture (MySQL)
- The backend application accepts **GET** requests at `/profile`
- The load balancer should map 80 to 8080



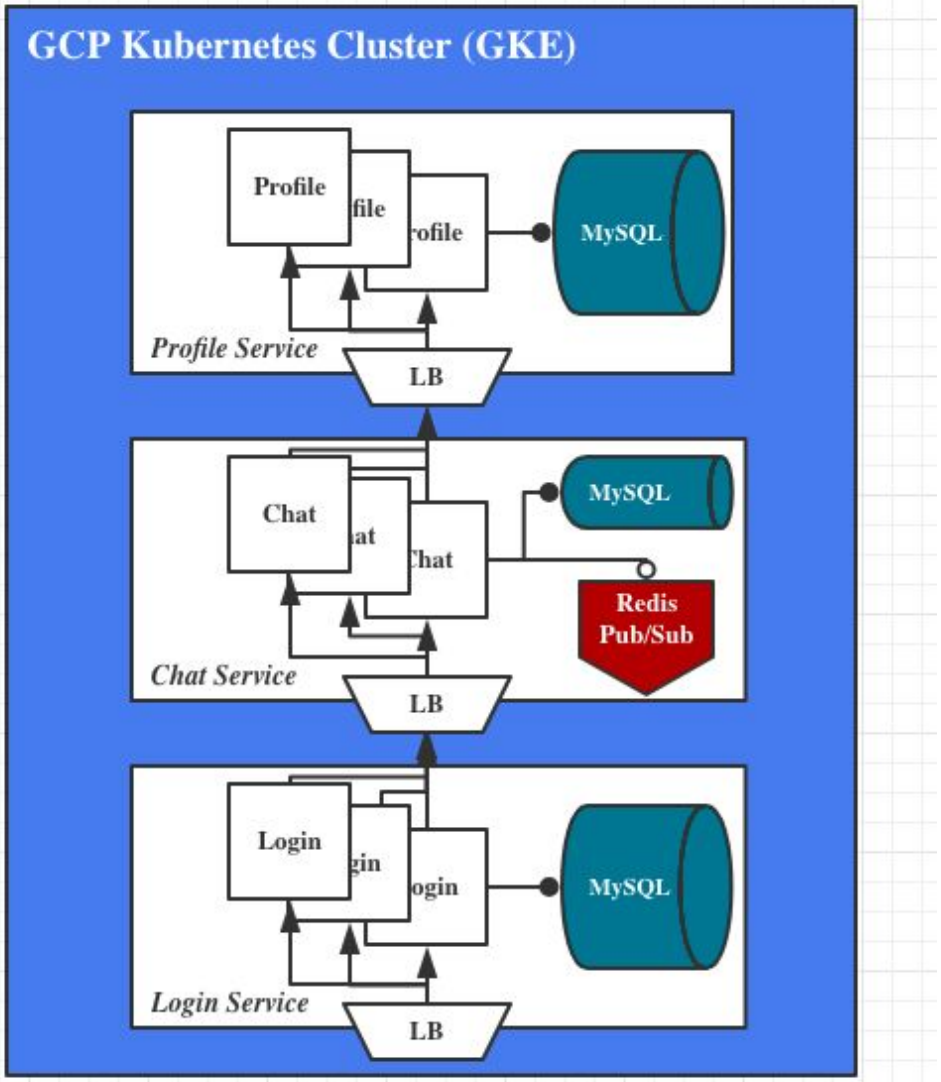
Task 4 - Cloud Chat Microservices

- Builds on Task 3
  - Additional login and group chat services

- Login service
  - Requires a separate MySQL database to store user login information

- Group chat service
  - Redis Pub/Sub messaging channel for scalability and real time communication
  - Separate MySQL to persist messages

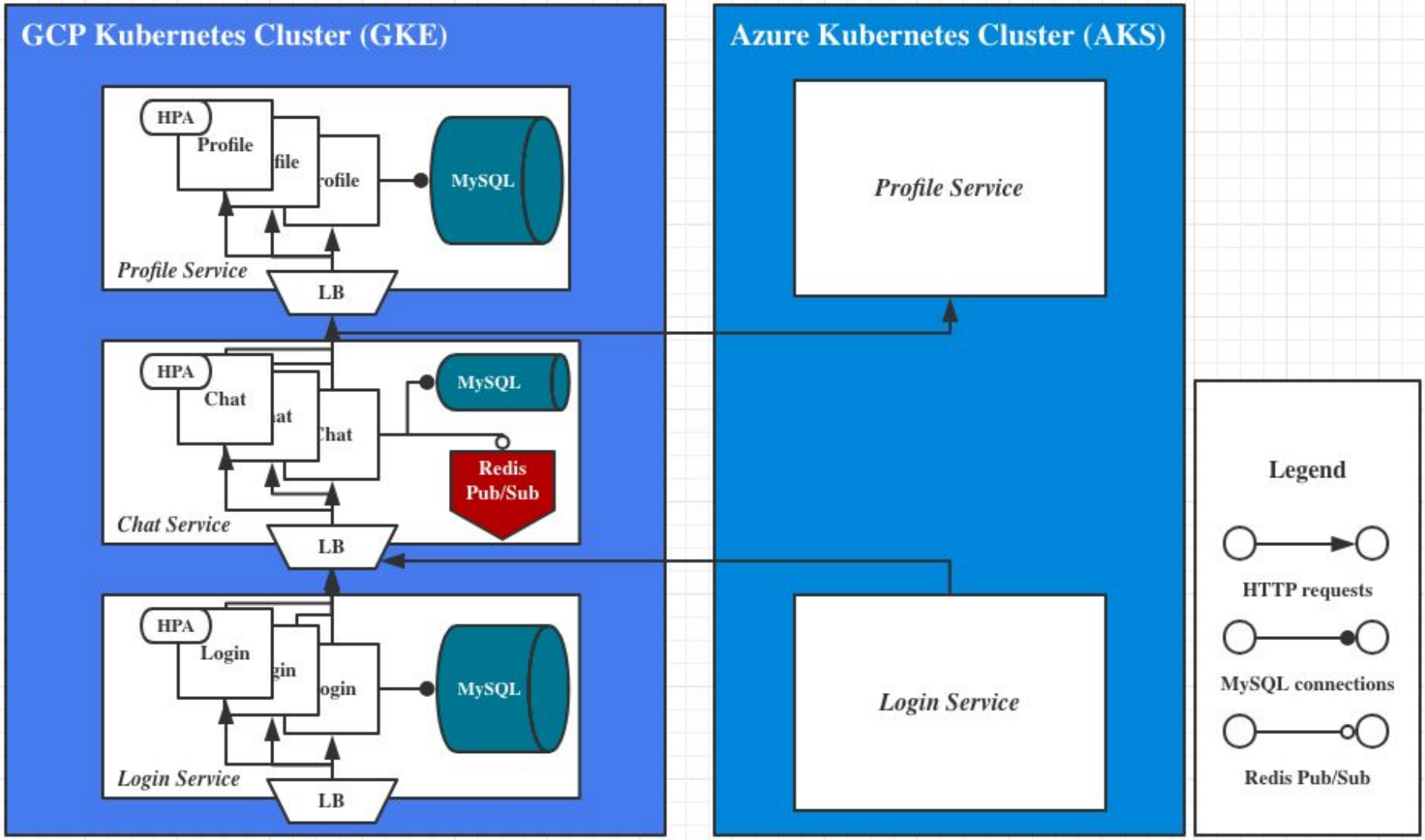
Task 4 - Cloud Chat Microservices



Task 5 - Autoscaling, Multiple Cloud Deployment and Fault Tolerance

- Builds upon Task 4
  - Consider how to handle downstream service failures
- Achieving high availability
  - Multi cloud deployments!
  - Autoscaling Kubernetes deployments to accommodated increased traffic
  - Can we use the `HorizontalPodAutoscaler` Kubernetes object?

Task 5 - Auto-scaling, Multiple Cloud Deployment and Fault-tolerance



Tips, Trips, and Tricks

- Debug, debug, debug
  - This project has many moving pieces!
  - Where is the issue occurring?
  - What is the expected behavior 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

The following table outlines the violations of the project rules and their corresponding grade penalties for this project.

Note that a penalty is the absolute value as per the table, not calculated by a percentage of your total score.

Violation	Penalty of the project grade
Incomplete submission of required files	-10%
Spending more than $7 for this project on AWS	-10%
Spending more than $10 for this project on AWS	-100%
Failing to tag all AWS resources for this project; `Key: project` and `Value: 2.2`	-10%
Provisioning resources in regions other than `us-east-1`	-10%
Using instances other than `t2.micro` in AWS as the submitter instance	-100%
Submitting your AWS credentials, other secrets, or Andrew Id in your code for grading	-100%
Submitting only executables (`.jar` , `.pyc` , etc.) without human-readable code (`.py` , `.java` , `.sh` , etc.)	-100%
Attempting to hack/tamper the grader	-100%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200% or R in the course

Upcoming Deadlines

- **Code Review - Project 1.2**
  - Due on **Wednesday**, Feb 13th, 2019, 11:59PM ET
- **Online Programming Exercises**
  - You will be notified of your schedule by email
  - Attend your scheduled session!
- **Quiz 4 (OLI Modules 7, 8 & 9)**
  - Due on **Friday**, Feb 15th, 2019, 11:59PM ET
- **Project 2.2**
  - Due on **Sunday**, Feb 17th, 2019, 11:59PM ET

Questions?