Containerization and Container Orchestration
Assignment 2 Out (Due 2/13)
AWS Credits Released (see Ed)
History of Web Hosting
(and enterprise compute)
Early 1990s: Physical Hosting

- Each individual physical server hosted one website on port 80

- **Scale challenge**: Adding capacity to accommodate traffic influx is subject to hardware lead time

- **Provisioning challenge**: Even if hardware available, still need to reinstall server software manually!
Late 1990s – Mid 2010s: Virtual Hosting

- Key innovation (1999): virtual machine hypervisors allow running of multiple guest OSes on a single physical host
  - Emulate the entire guest OS, both kernel and userspace

- Addresses many inefficiencies with physical hosting:
  - Can buy fewer, high-capacity servers, dividing capacity across VMs for different websites
  - Provisioning is faster: just create new VMs from stored images (or script provisioning)

- Enables cloud IaaS: someone else buys the physical servers and rents you the VM abstraction
Challenges of Physical Hosting

- Continued **scale challenges:**
  - VM provisioning can be multiple minutes → **under capacity** while dealing with traffic influxes
  - VM images can be several GB → **storage cost**

- **Management overhead:** still responsible for patching OS and software
  - Need for DevOps/Infrastructure Engineers to handle this responsibility
**Guiding principle**: Make running web (or other) applications as independent as possible from the underlying infrastructure.
Containers
Containers

- **Definition**: a *container* is a portable software package containing all resources needed to run it, providing:
  - **Isolation**: processes of container A don’t interfere with those of container B
  - **Replicability**: same process from same container image should execute the same on any host machine/OS/configuration

- Introduced in its current form in 2013 by Docker
  - Alternatives include Podman and LXD, but Docker is by far the most commonly used platform
More about containers

- A container behaves somewhat like its own isolated OS, **but shares its kernel with the host OS**
  - Key distinction compared to virtual machines

- Usually run on Linux hosts, deeply reliant on Linux kernel features
  - **Cgroups** (for process isolation and resource limiting)
  - **iptables** (for networking)
  - **OverlayFS** (layered, contained filesystems)
Building a Docker Container

Dockerfile
---------

# Inherit from parent container definition
FROM python:3.12-slim

# Run all future commands in /app directory
WORKDIR /app

# Make port 8080 accessible from host
EXPOSE 8080

# Run a simple HTTP server
# Use ENTRYPOINT to avoid overwriting parent setup steps
CMD ["python3", "-u", "-m", "http.server", "8080"]
Demo: Interacting with a Web Container
Container Networking

- By default: Docker containers are given own isolated network interface

- NAT behind host machine allows outbound network access, but need to explicitly expose ports for inbound

- Containers cannot talk to each other unless they are attached to the same software-defined network
  - Alternatively: use host networking mode to remove NAT abstraction (i.e., process that listens on a container port is accessible at the host scope too)
Container Storage

- Anything saved within the filesystem of a running container is *ephemeral*
  - Destroyed after the invocation ends

- Storage persistence through *volumes*: directories on the host system mapped to directories within the container

  ```
  docker run --rm -v ~/postgres:/var/lib/postgresql/data -p 5432:5432 postgres:latest
  ```

- Volumes are also how we can share host data with the container!

  ```
  ```
Container Storage on the Cloud

- In practice: we want to **separate compute and storage concerns**
  - *Treat your compute like cattle, not pets*

- Generally, store long-term persistent data elsewhere
  - Databases
  - Object storage (e.g., S3)
Container Registries

- Use pre-made container definitions as is, or extend them
  - Analogy: **package management for containers**
  - Inheriting from a parent container in a Dockerfile appends your Dockerfile to the parent’s

- Public registries: DockerHub, Quay.io
  - Freely download and host public images, many base images e.g. Go, Python, Nginx, Postgres

- Private registries: AWS Elastic Container Registry, Google Artifact Registry
  - Often used for organization-internal images, since download requires authentication

- Pulling containers: through Fully-Qualified Image Identification (FQID)
  - Format components include: `registry_name`, `username`, `image_name:tag`
  - e.g. `docker pull docker.io/library/ubuntu:22.04`
  - e.g. `FROM 123456789012.dkr.ecr.us-west-2.amazonaws.com/mywebapp:latest`
How to Containerize a Web Application

1. Pick a base image that simplifies some work for you
   ○ e.g. something with a language package manager/common dependencies installed

2. Copy app files

3. Install dependencies

4. Expose inbound port

5. Run application

Note: multistage builds can reduce final container size when working with compiled languages (e.g. Go, Rust)
Demo: Multistage Go Webapp Container Build
Container Orchestration
Motivation

- Containers, on their own, help us deal with software challenges of running web apps: dependencies and isolation.

- But infrastructure challenges still persist: scale, provisioning, storage, ...

- **Container Orchestration**: tooling that automates provisioning, scheduling, scaling, resource allocation, monitoring, and networking configuration across container task lifecycles.
Basic container orchestration: docker-compose

- Container orchestration for development environments
  - Run an application and all its dependencies together in an isolated networked environment

- **Limitation:** can only run containers on a single host
  - Hampers scaling and redundancy
Demo: docker-compose
Kubernetes

- Conceptually: Kubernetes is an **operating system for distributed container clusters**
  - Each process is a container
  - Kubernetes takes care of scheduling the container and managing its lifecycle given some configuration parameters

- Kubernetes features are oriented towards building scalably deployable and portable application patterns
  - Container scheduling, autoscaling, versioning, health checks
  - Networking, DNS service discovery, load balancing, ACLs, MTLS
  - Secrets and config management, observability
  - Further extensible using third party plugins!
Kubernetes Terminology

- **Cluster**: a set of *nodes* (hosts) that run containers
  - Made up of a single *control plane* and multiple *worker nodes*

- **Namespace**: an isolated group of resources within a cluster

- **Pod**: a group of one or more containers used for a single purpose
  - Share a *network namespace*, just like docker-compose

- **Deployment**: a way of maintaining a set of pods for scaling and redundancy
  - Ensures that the right number of pods are always running regardless of failures

- **Service**: a way to expose pods/deployments for external network access
  - Assigns a pod/deployment a virtual IP and/or DNS address
Example Architecture with Kubernetes
Demo: Kubernetes
Q: Kubernetes has a lot of features! Why don’t we just use the open-source framework instead of closed-source cloud provider-managed solutions?

A: Management overhead.
AWS Elastic Container Service (ECS)

- Fully-featured container orchestration service
  - Proprietary AWS platform
  - Just like Kubernetes: can handle container management across multiple nodes
  - Some different terminology: tasks vs pods

- A lot simpler to manage than Kubernetes (and EKS)
  - Can simply bring the container(s) and tell AWS how to run it
  - Set and forget

- Some limitations, such as persistent storage and task count

Takeaway: ECS is a great way to start doing simple container deployments (e.g. Assignment 2) without dealing with the complexity of Kubernetes
Zooming out: Cloud-Managed Container Orchestration

How much control do you want to retain vs how much the cloud provider manages for you?

More control
DIY all config and maintenance

Self-managed k8s on a fleet of compute instances

AWS EKS on EC2

AWS ECS on EC2
AWS EKS on Fargate
Azure AKS

More managed
Bring the container, everything else done for you

Google GKE
Azure Container Apps
AWS ECS on Fargate
Next Lecture: Infrastructure-as-Code (1/31)