
University of Colorado Anschutz Medical Campus
Center for Health AI (DBMI)
Software Engineering Open Hours - August 17, 2022
University of Colorado
University of Colorado

The Department and Center Mission

DBMI and the Center for Health AI fosters a thriving community of researchers on the Anschutz Medical Campus who are inventing and deploying advanced analytical approaches. The goal of building this community is to make the Anschutz Medical Campus a leader in translating data into advances in research practice, health care delivery, and population health and in scaling these to provide nationwide benefit through innovative technologies.

Agenda

- Schedule: 1st and 3rd weeks of every month
- Walkthrough(s)
  - Kubernetes (k8s) Overview
  - Deploying a Sample App to k8s
- Ideas for next topics
- How to reach us

Talk Overview

- Images and Containers
- Why Orchestration Matters
- Why Kubernetes Matters
- Kubernetes Overview
- Tools
- Running a Sample App on Kubernetes

Images and Containers, a Brief Background

- An **image** is a blueprint for a small system that contains an application, i.e.
  - a **filesystem** which contains the executables, resources, and dependencies
  - an **entrypoint** that specifies how the application starts
  - optional mappings to **host resources**, e.g.
    - **ports** on which internal services are listening for requests
    - **volumes**, which specify paths within the container that should, e.g., persist between container runs, or be shared between containers, or be shared with the host

Images and Containers, a Brief Background

- An **image** is a blueprint for a small system that contains an application, i.e.
  - a **filesystem** which contains the executables, resources, and dependencies
  - an **entrypoint** that specifies how the application starts
  - optional mappings to **host resources**, e.g.
    - **ports** on which internal services are listening for requests
    - **volumes**, which specify paths within the container that should, e.g., persist between container runs, or be shared between containers, or be shared with the host
- To actually start the app, the image is loaded into a runtime (e.g., the Docker daemon), its host resources are allocated and mapped, and its entrypoint is launched, at which point it becomes a running **container**

Images and Containers, a Brief Background

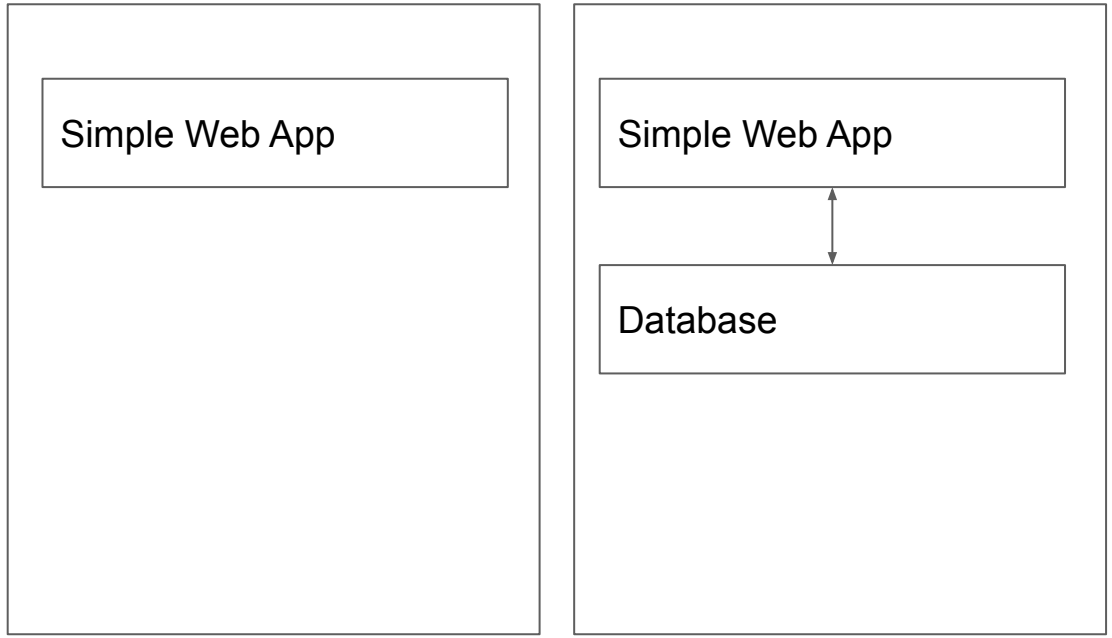
- An **image** is a blueprint for a small system that contains an application, i.e.
  - a **filesystem** which contains the executables, resources, and dependencies
  - an **entrypoint** that specifies how the application starts
  - optional mappings to **host resources**, e.g.
    - **ports** on which internal services are listening for requests
    - **volumes**, which specify paths within the container that should, e.g., persist between container runs, or be shared between containers, or be shared with the host
- To actually start the app, the image is loaded into a runtime (e.g., the Docker daemon), its host resources are allocated and mapped, and its entrypoint is launched, at which point it becomes a running **container**
- This separation between the app and host has **lots of advantages**, e.g. it:
  - makes it easy to **launch others' applications** with confidence that it will run the same as it did on their machine
  - **prevents polluting the host** with the application's dependencies
  - eases running **multiple independent copies of an application** mapped to different ports and volumes on the host

Why Orchestration Matters

- A single container: **Docker**

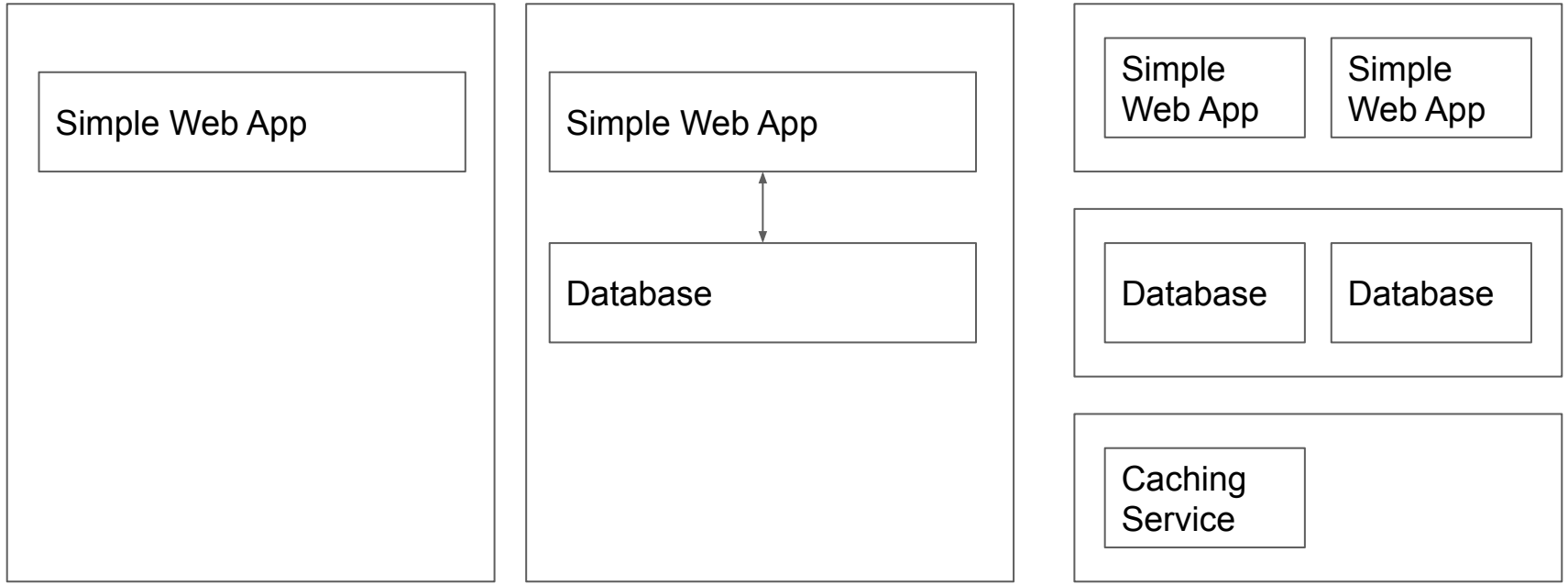
Why Orchestration Matters

- A single container: **Docker**
- Multiple containers on a single machine: **Docker Compose**



Why Orchestration Matters

- A single container: **Docker**
- Multiple containers on a single machine: **Docker Compose**
- Multiple containers on multiple machines: **Orchestration** (e.g., Docker Swarm)



Why Orchestration Matters

- A single container: **Docker**
- Multiple containers on a single machine: **Docker Compose**
- Multiple containers on multiple machines: **Orchestration** (e.g., Docker Swarm)
- Specifically, orchestration allows containers to be mapped to nodes based on **constraints**, e.g.
  - The node's current available CPU and memory vs. what the container declares it needs
  - Whether all the containers should run on the same node (e.g., if they share data on a local device), or on different nodes (for fault-tolerance)
  - Explicitly, e.g. "service A goes to node 1"

Why Orchestration Matters

- A single container: **Docker**
- Multiple containers on a single machine: **Docker Compose**
- Multiple containers on multiple machines: **Orchestration** (e.g., Docker Swarm)
- Specifically, orchestration allows containers to be mapped to nodes based on **constraints**, e.g.
  - The node's current available CPU and memory vs. what the container declares it needs
  - Whether all the containers should run on the same node (e.g., if they share data on a local device), or on different nodes (for fault-tolerance)
  - Explicitly, e.g. "service A goes to node 1"
- Orchestration also allows containers to be **monitored**, **rebooted** if the container errors, **moved to a different node** if a node should fail, etc.
  - effectively anything that one would have to do directly (aka **imperatively**) one can configure an orchestrator to do **declaratively**
  - To put it differently, rather than tell the cluster *how to do it*, you tell it *what you want*

Why Kubernetes Matters

- Non-local volumes, container replication, custom networking needs, self-healing services: **???**

Why Kubernetes Matters

- Non-local volumes, container replication, custom networking needs, self-healing services: **???**
- **Kubernetes** (aka “k8s”) has the same core capabilities as Docker Swarm, but abstracts a lot more infrastructure from built-in to configurable components
  - **storage:** splits volumes into persistent volumes + claims, abstracts their connection to containers
  - **physical-layer networking:** choice of bare-metal networking, overlay networks between nodes, ane more
  - **ingress:** choice of NodePorts, which are ports exposed on all nodes, or more fine-grained networking options)
  - **service discovery:** custom DNS allows services to be discovered by cluster-local hostnames

Why Kubernetes Matters

- Non-local volumes, container replication, custom networking needs, self-healing services: **???**
- **Kubernetes** (aka “k8s”) has the same core capabilities as Docker Swarm, but abstracts a lot more infrastructure from built-in to configurable components
  - **storage:** splits volumes into persistent volumes + claims, abstracts their connection to containers
  - **physical-layer networking:** choice of bare-metal networking, overlay networks between nodes, ane more
  - **ingress:** choice of NodePorts, which are ports exposed on all nodes, or more fine-grained networking options)
  - **service discovery:** custom DNS allows services to be discovered by cluster-local hostnames
  - **API:** all actions on the cluster are conducted through a RESTful HTTP API secured w/certificates, paving the way for automation

Kubernetes: An Overview

- At its core, Kubernetes is a platform for managing **pods** (collections of containers on a single physical machine)
  - this includes fetching the images for the containers, hooking them up to local resources (disk, network), booting the containers, cleaning them up when they exit, etc.
- Kubernetes is “eventually correct” and much of its configuration is represented as contracts with the system
  - Special containers with access to the cluster’s API called **controllers** are constantly polling the current state and trying to move it toward the desired state
  - E.g., a Deployment is a contract (and backed by a controller) that starts X number of Pods. When the Deployment is first created, there are no pods, but over time the Deployment ensures that X number of pods exist. if pods fail, new ones are started to bring the running count back to X
- Kubernetes is **highly extensible**:
  - new data types called Custom Resource Definitions (CRDs) can be defined and they can trigger actions via controllers that watch for changes to Custom Resources
  - many Kubernetes API clients exist for popular languages, e.g. Python
  - new controllers can be created to monitor the state of the cluster and steer it toward a desired state
- Kubernetes is configured via **manifests**, YAML documents that define entities and their relationships with each other. A list of all these entities and their manifests can be found in the Entity Reference.

Tools

- Cluster creation:
  - k3s: a small, opinionated kubernetes distribution
    - rke2 is even more opinionated, but good if you want k3s + an admin UI and other niceties
  - k0s: less opinionated than k3s, so a few things are missing out of the box, but lots of options for configuration and low overhead by default
  - minikube: a typically virtualized single-node kubernetes cluster for local development
  - * Docker Desktop k8s: a single-node local dev k8s cluster that runs within Docker, similar to kind
- Management:
  - * Lens: a GUI for managing k8s clusters
  - * kubectl: the standard CLI tool for managing k8s clusters

** tools that I'm using in this presentation*

Hands on Deployment of *Reformed* to K8S

- Pandoc, ”a universal document converter” is a useful tool for converting between a wide variety of document formats
- It can be difficult to create the environment to run pandoc, so let’s use a container that already contains that env
- Reformed implements an API on top of pandoc, and is provided as an image on Docker Hub, making it easy for us to deploy onto Kubernetes
- With a small amount of configuration, we can deploy Reformed onto Kubernetes and provide access to it at a stable URL

Live App Deployment Demo


University of Colorado
Anschutz Medical Campus

Next Topics

Ideas (so far!):

- Python, Pip, and Poetry
- Dask and Ray
- Bite-sized ML
- Vue3 and composition
- SVG Tutorial
- Clustered computing
- Cuelang and Dagger
- Jupyterbook
- DVC (data version control)
- Orchestrating containers (docker to docker-compose/swarm to k8s)
- Workflow: Snakemake, Prefect, NextFlow
- Html, CSS (LESS vs SASS), javascript
- GCP console crash course: Identity and Access Management (IAM), automation via gcloud CLI

Feel free to add ideas here: Office Hours Outline and Ideas.docx

How to reach us

Slack

#software-engineering

Email

cuhealthai-softwareengineering@cuanschutz.edu

GitHub

Center for Health AI (github.com) (this title may change soon to DBMI, so watch for that)


University of Colorado Anschutz Medical Campus
THANK YOU