With Big Data Comes Big Compute: Scaling Machine Learning onto Public and Commercial Clouds with Kubernetes – Part 1

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023

University of Missouri

Part 1 – Learning Objectives

▶ Containerization

▶ Kubernetes Architecture and Concepts

▶ Hands-on with Nautilus HyperCluster

▶ Deploying Pods and Jobs in Kubernetes using Nautilus

University of Missouri

Workshop Outline

▶ Software Containerization with Docker

  ▶ Containerization basics

  ▶ Docker concepts: Pods, Jobs, etc.

  ▶ Hands on with Docker

  ▶ Advanced Concepts with Docker

▶ Container management with Kubernetes

  ▶ Introduction to Kubernetes

  ▶ Kubernetes concepts

  ▶ Kubernetes usage

▶ NRP Nautilus HyperCluster

  ▶ Cluster introduction, background and resources

  ▶ Case Study: How MU utilizes Nautilus

  ▶ Hands on with Kubernetes in Nautilus

University of Missouri

High-Performance Data-Intensive
Computing Systems Laboratory

Part 1-A
Software Containerization
with Docker

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023

University of Missouri

High-Performance Data-Intensive
Computing Systems Laboratory

Introduction

What is Docker and what problem is Docker trying to solve?
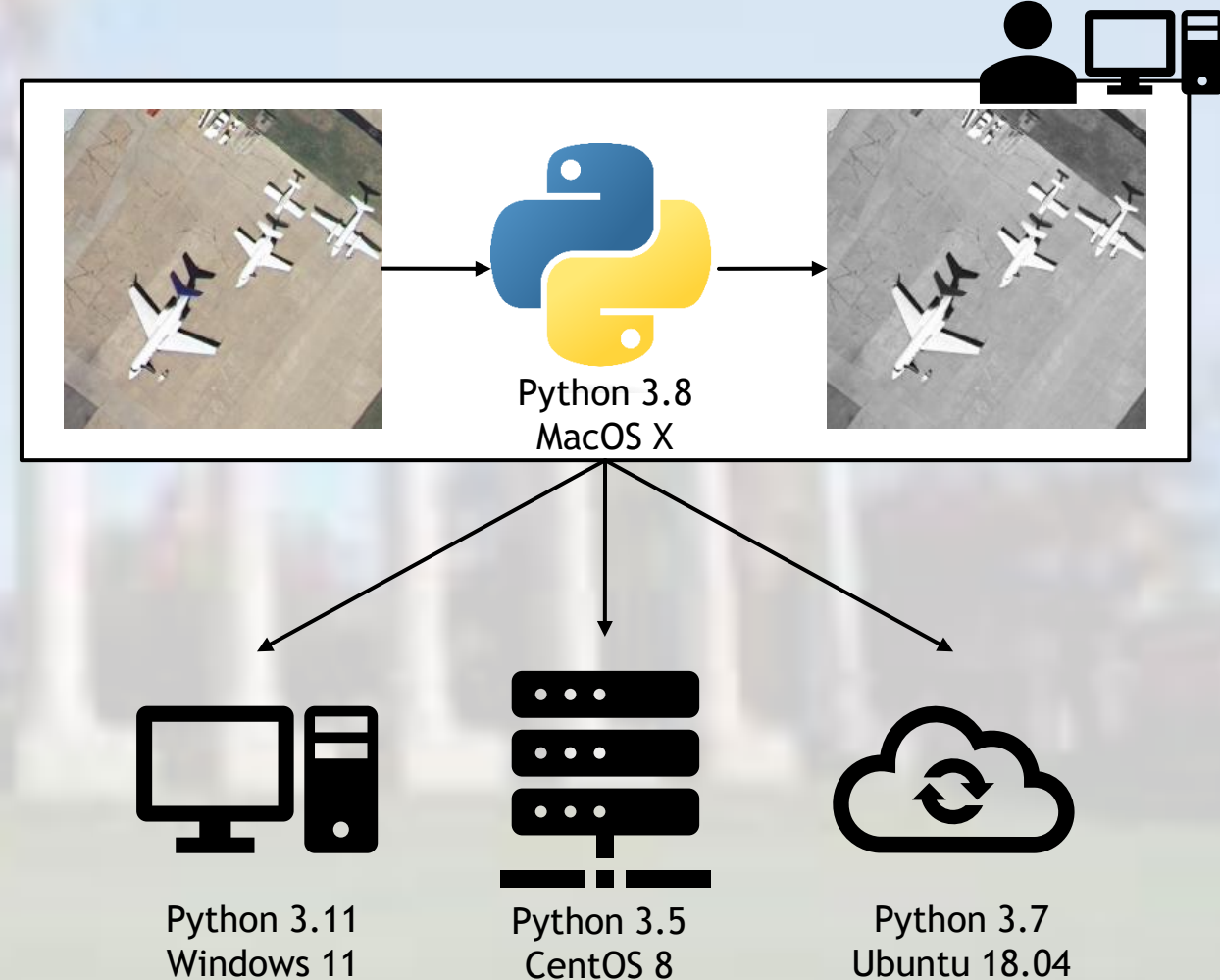
University of Missouri

The Problem:
Scalability & Reproducibility

▶ Development often occurs on local development machines

▶ How do we ensure reliable portability of the software developed on local development machines to other computational environments?

▶ How do we move code from local development to running on large servers on large swaths of data?

Local Machine

?

Compute Server

Cloud

University of Missouri
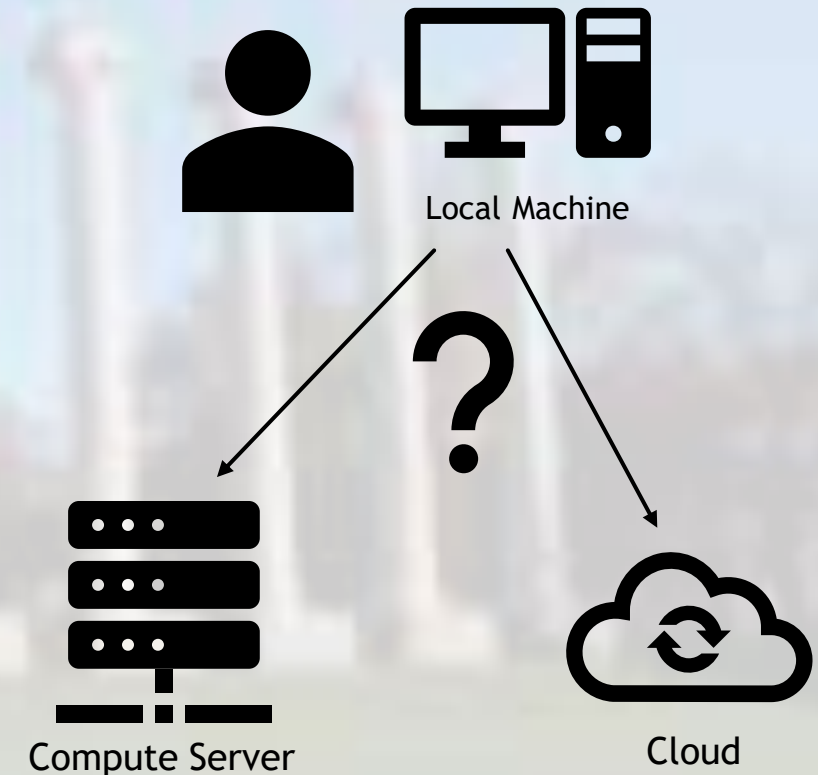
Example: Python Application for Image Processing

▶ I've built an application on my local machine to enable efficient image processing using Python 3.8

▶ My application requires certain Python packages **AND** system packages

▶ The specific versions of Python and the system packages are required

▶ How do I move my application to co-worker's computer? What about to a compute server? To the cloud?

▶ Each of these locations will have their own installed system libraries, python installations, and python packages

Python 3.8
MacOS X

Python 3.11
Windows 11

Python 3.5
CentOS 8

Python 3.7
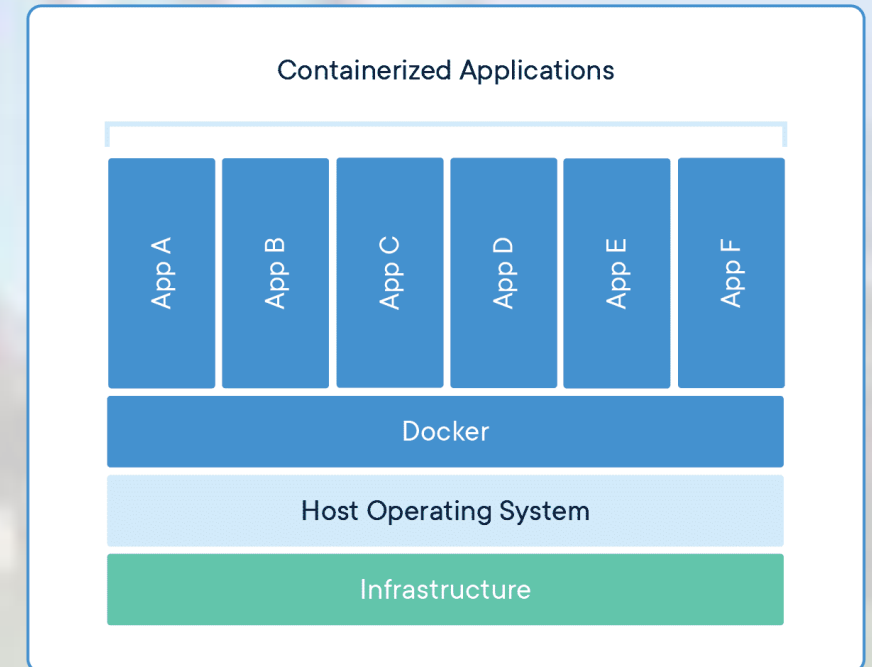Ubuntu 18.04

University of Missouri

Containerization

► Solution: Package all of our requirements, at system level and at the language-specific level, into a software package, called a **container image,** that can be run **anywhere**

► Each location now only needs a **container runtime** and the software package

University of Missouri
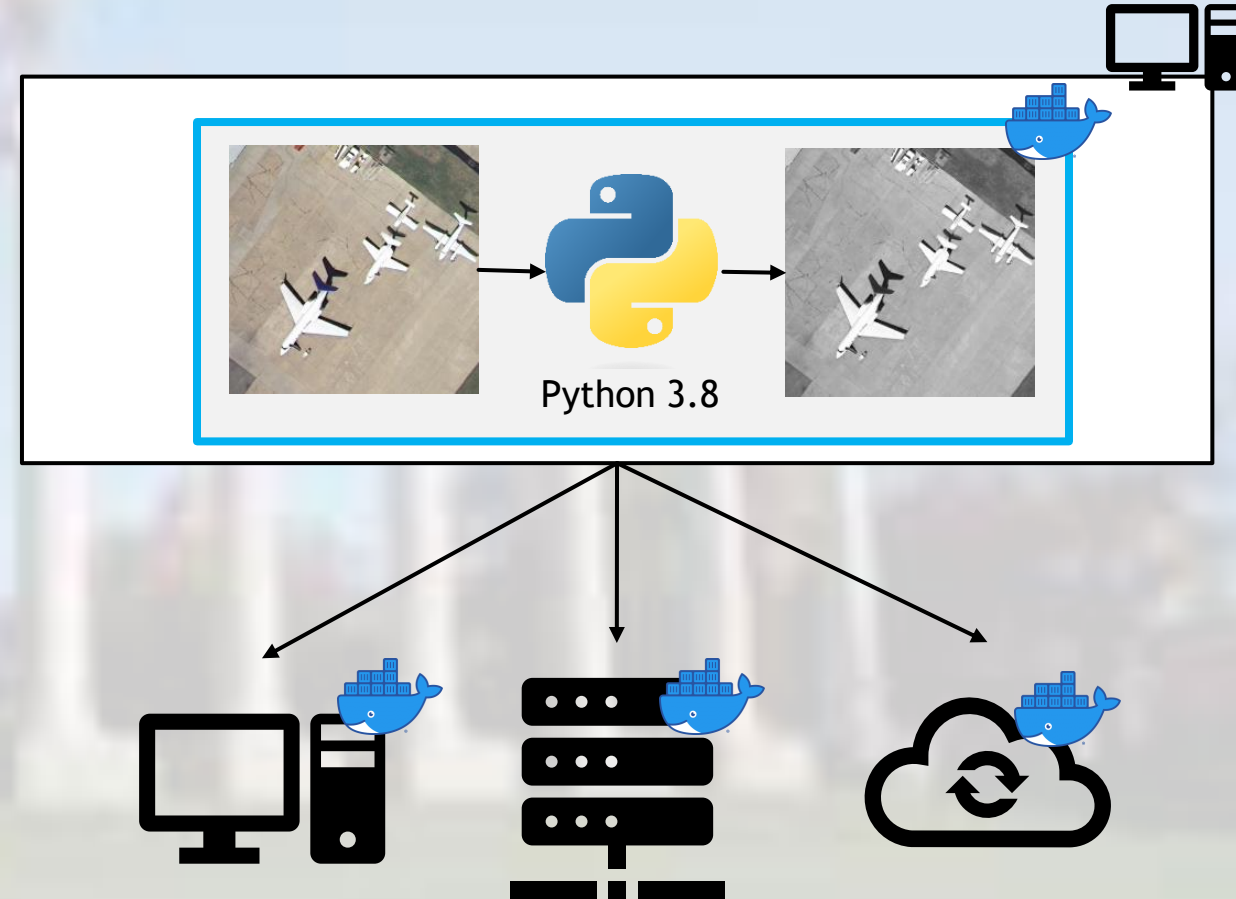
Container Runtimes

▶ Container runtimes are software components that run containers on a host operating system

▶ Every machine that we want to run containers on needs a container runtime

▶ There are multiple container runtimes:

  ▶ Docker: most common

  ▶ Podman: RHEL/CentOS replacement for Docker. Does not require root access

  ▶ Singularity: Common in HPC applications

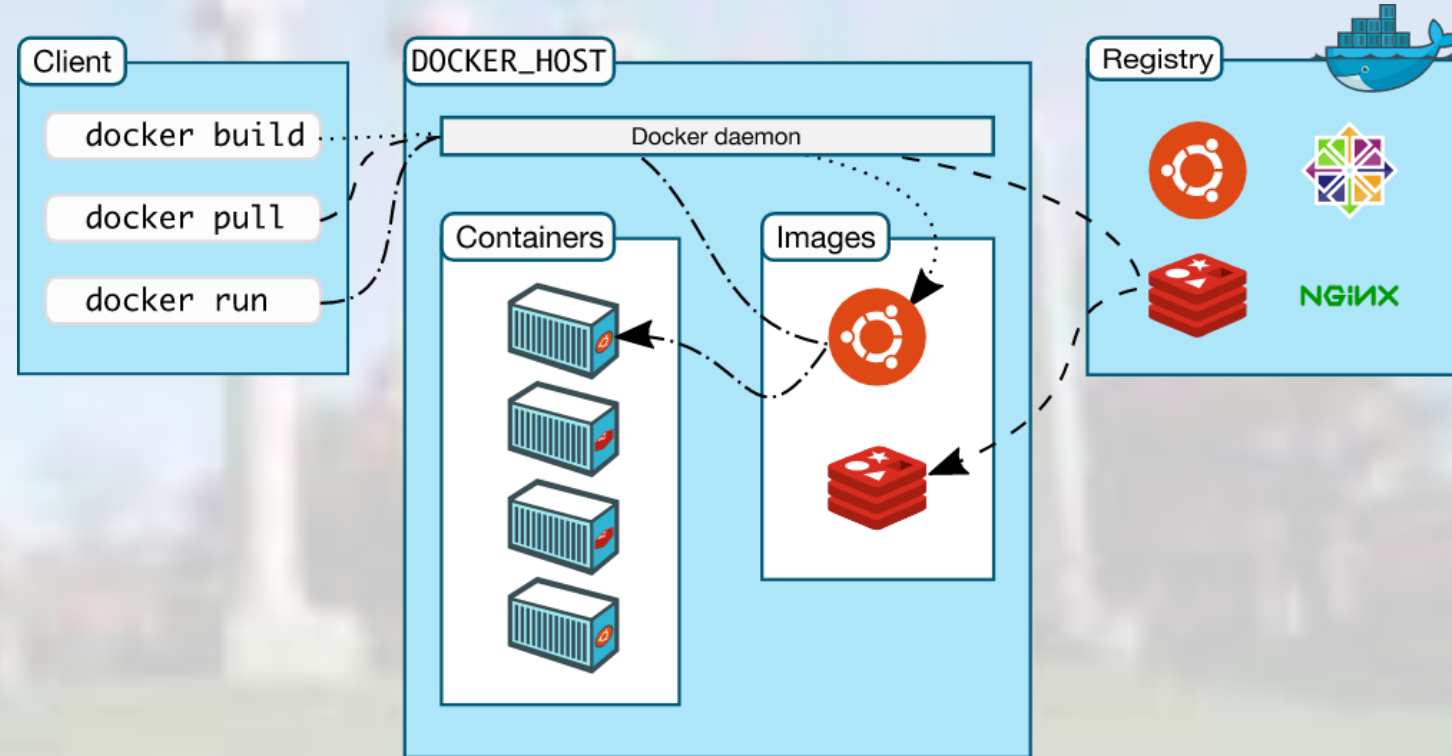▶ We will discuss Docker here, at its concepts are generalizable to other runtimes

Containerized Applications

App A | App B | App C | App D | App E | App F

Docker

Host Operating System

Infrastructure

University of Missouri
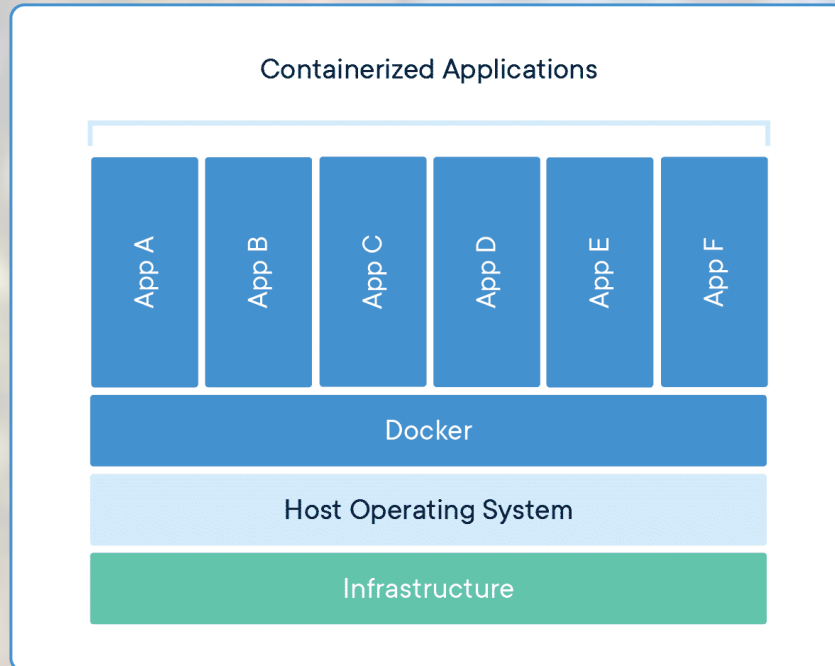
Docker

▶ **What is Docker?**

  ▶ Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.[1]

  ▶ You can think of Docker containers as mini-VMs that contain all the packages, both at the OS and language-specific level, necessary to run your software.

▶ **Why Docker?**

  ▶ Docker enables predictable and reliable deployment of software.

  ▶ Docker containers are portable!

    ▶ local development computers, compute clusters, internal compute servers, cloud infrastructure, and more!

▶ *Docker containers enable reliable portability of software to nearly any compute environment*

University of Missouri

1. https://en.wikipedia.org/wiki/Docker_(software)
2. Image: https://docs.docker.com/get-started/overview/

Docker: Containers vs Virtual Machines

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.
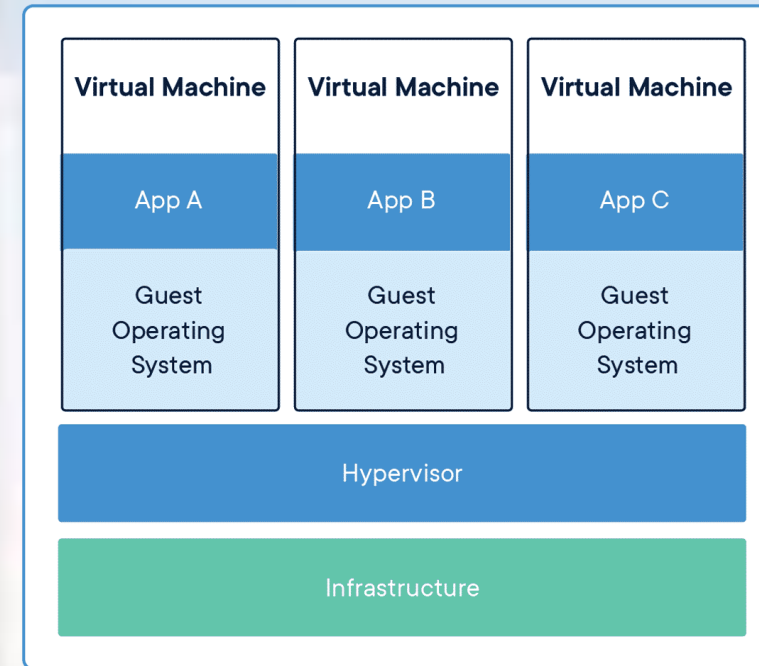
Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.
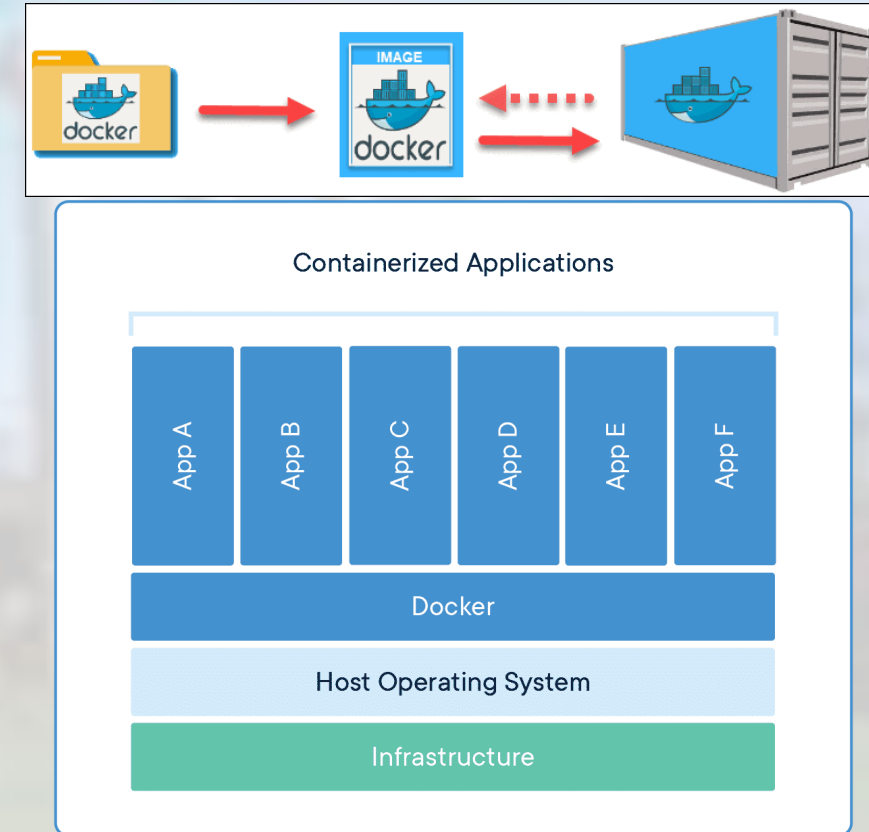
University of Missouri

https://www.docker.com/resources/what-container/

Docker Concepts

Key concepts: containers, container images, Dockerfiles, container registries

University of Missouri

Key Docker Concepts

▶ **Dockerfile** - A list of commands and instructions describing how to build an Image

▶ **Image** - Standard unit of software that packages up code and its dependencies so the application runs reliably from one computing environment to another.

  ▶ Includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

▶ **Container** - An instantiated runtime of a docker image, containing all necessary software for a given application to run, both at the OS and language-specific level

▶ **Registry** - a service for storing container images

Containerized Applications

University of Missouri

https://www.docker.com/resources/what-container/
https://phoenixnap.com/kb/docker-image-vs-container

Containers vs Container Images

▶ The terms containers and images are often used interchangeably when discussing Docker, but there are some key distinctions

▶ Container images are software packages that include all necessary software to run the application/library

▶ Containers are an instantiation of container images

    ▶ Images become containers at runtime

    ▶ Analogous to relationship between a script and the process or a class and an object
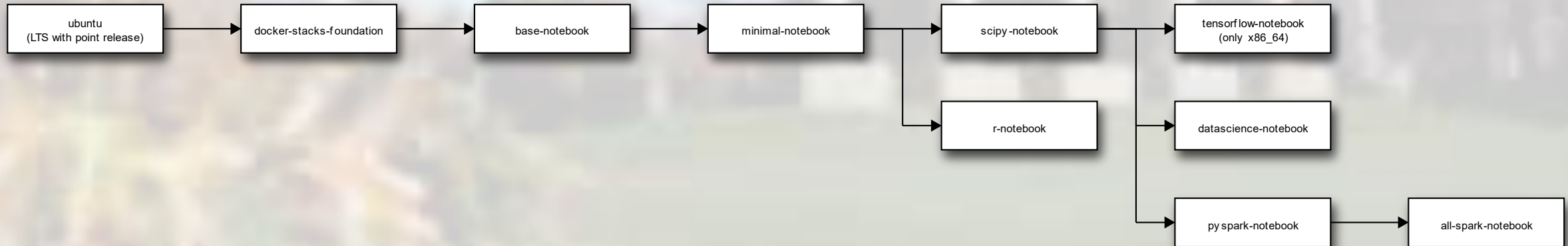
Python 3.8

Container Image

University of Missouri
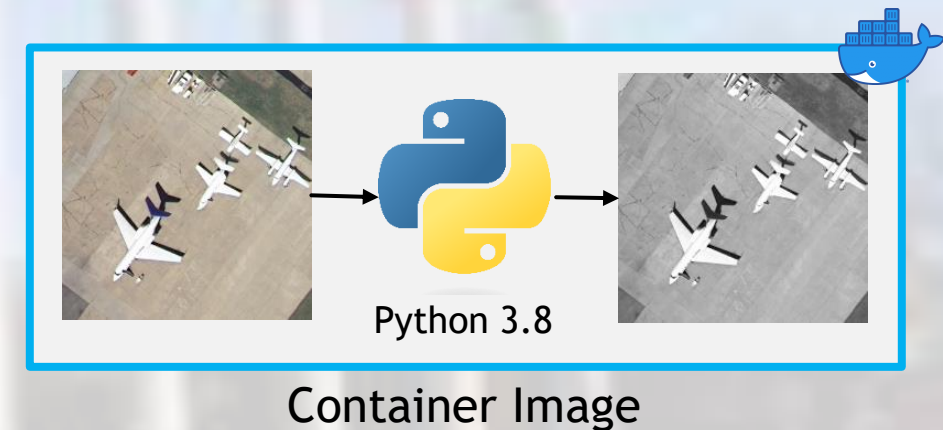
Dockerfiles

▶ Recall: Containers and container images enable reliable portability of software

▶ We need a way to build container images in a distributed and standardized way, so that anyone with a container runtime can build and run our software package (container image)

▶ Dockerfiles are the template, or recipe, for how a container image will be built

▶ Dockerfiles use a custom format and set of commands to describe how a container image will be built

Dockerfile

Docker

Runnable Container Image

University of Missouri
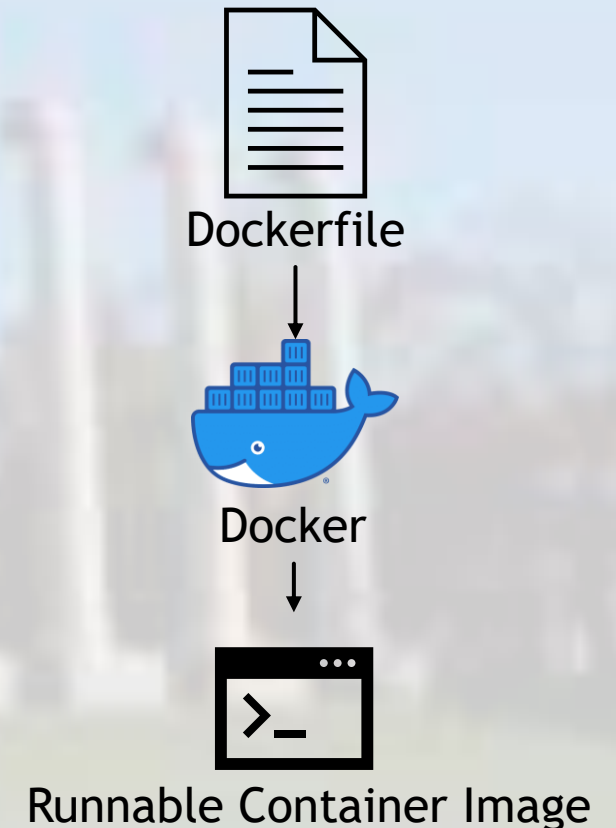
Using Community Published Container Images

▶ Container images are extensible!

   ▶ Containers can be built from other images

▶ We can then build hierarchical docker containers, where each container has a specific set of dependencies and packages installed

▶ Base images enable rapid and reproducible building of even complex docker containers by leveraging the open source, published docker images

Example: Jupyter Docker Images[1]

1. https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html

Dockerfile: Common Commands

▶ Dockerfiles function via a set of directives and their respective arguments:

DIRECTIVE arg1 arg2 …

▶ FROM

- ▶ The FROM command defines the starting point for the Docker build process. If you are building custom software from the ground up, you may be setting this to a Linux operating system, such as ubuntu.

- ▶ If you are working in a framework or programming language, this may also be a certain version of that framework, such as python:3.8, node:8, pytorch:1.8, etc.

- ▶ The docker image specified in the from command must be present either on the public docker hub repository, or be visible to the build context via a full URL.

▶ ENV & ARG

- ▶ The ENV and ARG commands define environment variables during the build process. There are 2 key differences between them:

- ▶ ENV commands persist to the final container, so ENV key value will persist to the launched container, while ARG key value will not

- ▶ ARG commands can be overridden at build time, which allows for templating of Dockerfiles

- ▶ The ENV command is very useful for tasks like ensuring your executable is present on the PATH of the container, while ARG is useful for things like ensuring you are building the correct version of the software.

University of Missouri

Dockerfile: Common Commands

▶ CMD and ENTRYPOINT

- ▶ The CMD and ENTRYPOINT commands both set the command to be run when the container starts via a docker run command.

- ▶ There is 1 key difference between them: a CMD can be overridden via either a downstream Dockerfile (i.e., someone uses your container in their FROM command) or via the command line during container startup. An ENTRYPOINT cannot be as easily overridden, and requires a special flag to be overridden.

- ▶ Best practice is to set a CMD unless you have reasoning behind using ENTRYPOINT

▶ COPY

- ▶ The COPY command is how local files are copied into the container. Recall that Docker containers are mini virtual machines, meaning that it has its own filesystem. In order to build and run software in docker, we often need to copy our code and scripts into the container. To do this, we use the COPY command

▶ RUN

- ▶ The RUN command tells the docker build process to run a command inside the container. This could be everything from installing a package with apt (RUN apt install) to creating and removing files in the container.

- ▶ The commands available for the RUN command are determined by the base container, i.e. apt will only be available in Debian based containers.

University of Missouri

Sample Dockerfile

```
ARG PYVERSION=3.8

FROM python:${PYVERSION}

RUN mkdir -p /workspace

WORKDIR /workspace


COPY /requirements.txt /workspace

RUN pip install -r ./requirements.txt


COPY /*.py /workspace/

CMD /bin/bash
```

Create a build argument for the python version that defaults to 3.8

Start FROM an existing image in a Docker *registry*. In this case, python

Create a directory named /workspace and set it as the working directory

Copy a file named "requirements.txt" from the build directory to /workspace in the container

Use pip to install the requirements defined in requirements.txt

Copy all python files in the build directory to the /workspace directory in the container

Set the default command to run when the container starts to /bin/bash

University of Missouri

Distributing and Sharing Containers

► Container runtimes enable reliable portability of software by building container images

► We can utilize container images built and published by the container community utilizing base images

► How do we share and distribute container images that we have built?

   ► Container Image Registries

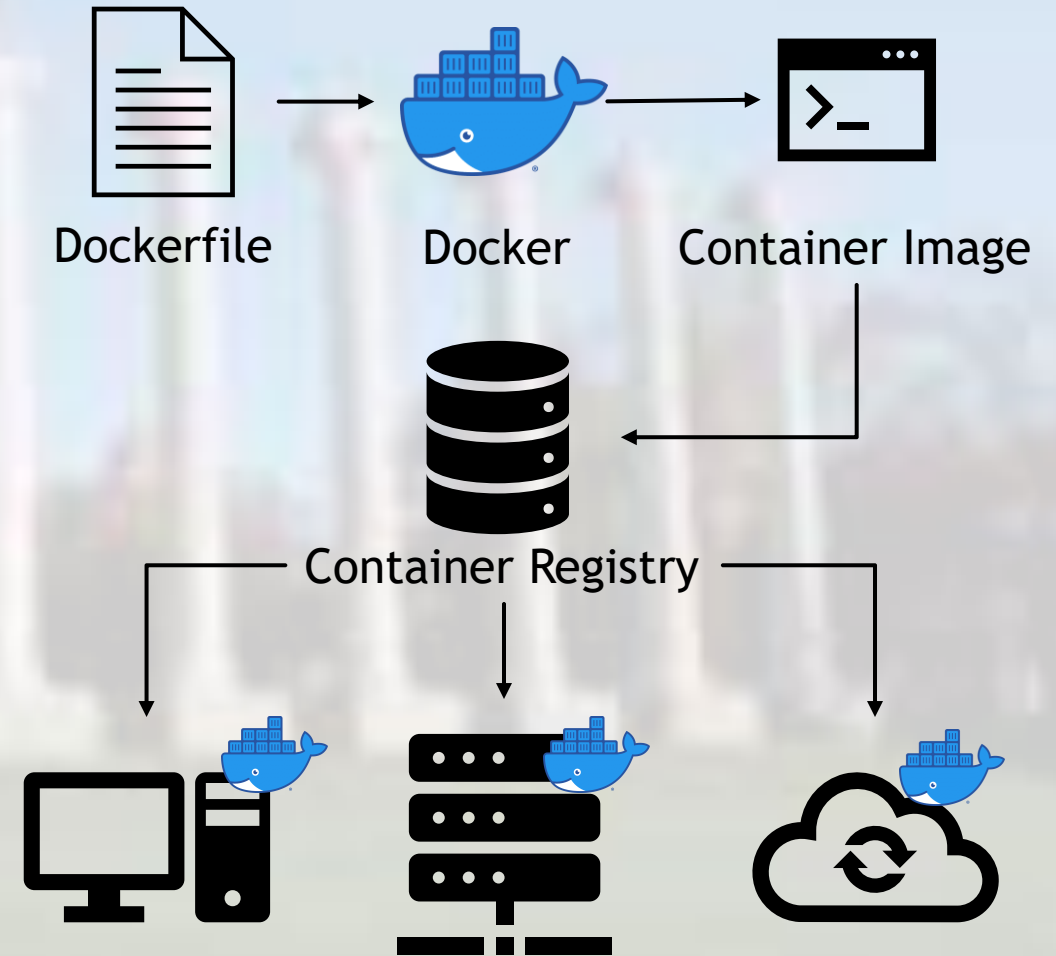Dockerfile   Docker   Container Image

University of Missouri

20
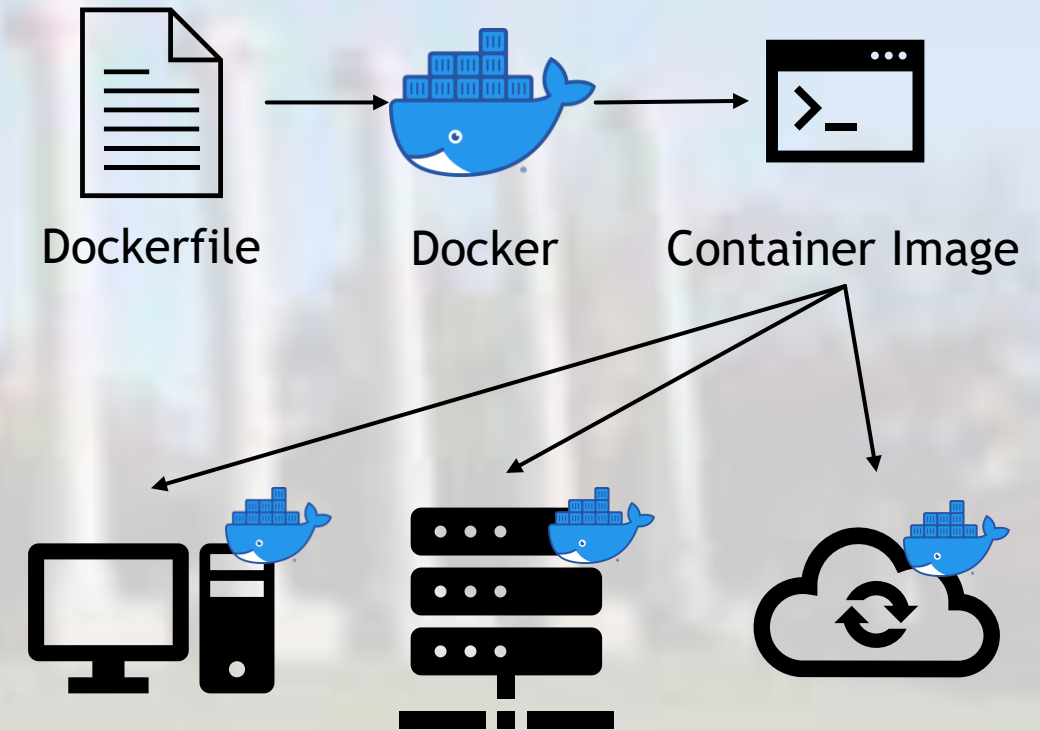
Docker Image Registries

▶ Container Registries are web-enabled storage locations for Docker container images

  ▶ Similar to Google Drive for documents and spreadsheets

▶ Each image published on a registry contains a name and a tag:

  ▶ python:3.8 → Python is the name of image and 3.8 is the tag

▶ If a URL is not specified, the default registry used is Docker Hub

  ▶ https://hub.docker.com/

▶ Other Container Registries

  ▶ Docker can work with third party container registries when given full URL to the image

  ▶ Example: nvcr.io/nvidia/pytorch:22.08-py3

▶ Security and Visibility

  ▶ Container images published on registries can be public or private

University of Missouri

High-Performance Data-Intensive
Computing Systems Laboratory
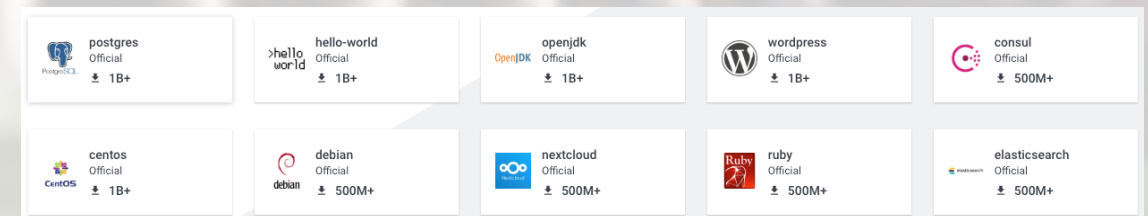
Revisiting our Example

▶ We can publish our container image to a registry, and then pull down our container image in each computing environment in which we want to use it

▶ We can utilize public registries, like Docker Hub, or private ones, to control who we allow to pull down our container

Dockerfile   Docker   Container Image

Container Registry

University of Missouri

Docker Hands-On

Common Commands and Interactive Docker Usage including pulling, building, and running containers

University of Missouri

Common Docker Commands: System Information

▶ To see all running containers:

<div align="center">docker ps</div>

▶ To see all container images on our system:

<div align="center">docker images</div>

▶ Delete a container:

<div align="center">docker rm [container]</div>

▶ Delete a container image:

<div align="center">docker rmi [image]</div>

University of Missouri

Common Docker Commands:
Pulling Containers

▶ To download or update a container image from a registry, we use the docker pull command:

docker pull [Image Name]

▶ We can specify simply the name of the image, and by default, Docker Hub will be used as the registry and latest will be used as the tag. Therefore the following two commands are the same:

docker pull ubuntu

docker pull docker.io/library/ubuntu:latest

▶ We can specify a name + tag to pull a specific tag from Docker Hub:

docker pull python:3.8

▶ Finally, we can specify a full URL to use a custom registry:

docker pull nvcr.io/nvidia/pytorch:23.02-py3

University of Missouri

Common Docker Commands: Running Containers

▶ Running a container: docker run

docker run [options] [Image Name] [command]

   ▶ If the container image is not found on your system, docker will search for that name and tag on Docker Hub and download it before running it

▶ Interactive running a container on Docker Hub:

docker run –it ubuntu:18.04

   ▶ Any options not specified will be set to default, which are set by either the Docker image itself (through the Dockerfile) or by the Docker daemon

▶ Overriding the startup command:

docker run –it python:3.8 /bin/bash

▶ Running a container in the background:

docker run –d [Image Name]

▶ Mounting a local directory to the container:

docker run –it –v /mylocaldir:/containerdir ubuntu:20.04 /bin/bash

▶ Exposing host ports to the container:

docker run –p 8888:8888 -d nginx

University of Missouri

Common Docker Commands: Building Containers

▶ To build a container image, we use the docker build command:

docker build [options] [context path]

▶ To build a container image from a Dockerfile in the current directory:

docker build .

▶ It is common to set a name and tag for the image you are building:

docker build –t myimage:latest .

▶ When using more than 1 Dockerfile, we specify which one with another flag:

docker build –t myimage:latest –f Dockerfile.py3 .

▶ If we are using a single Dockerfile with multiple build stages, we specify that as well:

docker build –t myimage:latest –-target stage2 .

University of Missouri

Docker in Action: ImageMagick

Dockerfile

```
FROM ubuntu:20.04

RUN apt update –y  &&  \
    apt install –y imagemagick wget

RUN mkdir -p  /workspace
WORKDIR /workspace

CMD /bin/bash
```

Bash

```
$ docker build .   –t  imagemagick

$ docker tag   imagemagick jalexhurt/imagemagick

$ docker push  jalexhurt/imagemagick
```

▶ We can combine Docker with VCS & CI/CD systems to automate this process, but for this example we do each step manually

▶ Once we have built and pushed our container, we can see it online on Docker Hub:

  ▶ https://hub.docker.com/repository/docker/jalexhurt/imagemagick/

University of Missouri

Using the Container

▶ Once we have pulled the container, we can run it and use the software packaged inside!

▶ In this example, we download an image from the COCO dataset

▶ Using ImageMagick, we then:

  ▶ read its properties

  ▶ resize and convert it to PNG

  ▶ read the properties of the newly created PNG

Bash

```
$ docker run –it jalexhurt/imagemagick:latest

$ wget http://farm8.staticflickr.com/7441/9539317874_8b108e4489_z.jpg  -O  image.jpg

$ identify image.jpg

$ convert  -resize 512x384 image.jpg image.png

$ identify image.png
```

University of Missouri

Docker in Action:
Live Demo
https://tinyurl.com/gpn-docker

https://gitpod.io/#https://gitlab.nrp-nautilus.io/jhurt/imagemagick

University of Missouri

Advanced Docker Concepts

Automation, Microservices, Docker Compose,
Multi-Stage Builds, and Cloud Computing

University of Missouri

Automation and Docker Registries

▶ Recall: Dockerfiles are a set of instructions to build an image

▶ Continuous Integration / Continuous Deployment (CI/CD) systems can enable automation of publishing docker images

▶ Common CI/CD Services:

  ▶ Gitlab CI

  ▶ Docker Cloud

  ▶ Jenkins

Developer

GitHub

Container Registry

Docker Cloud

University of Missouri
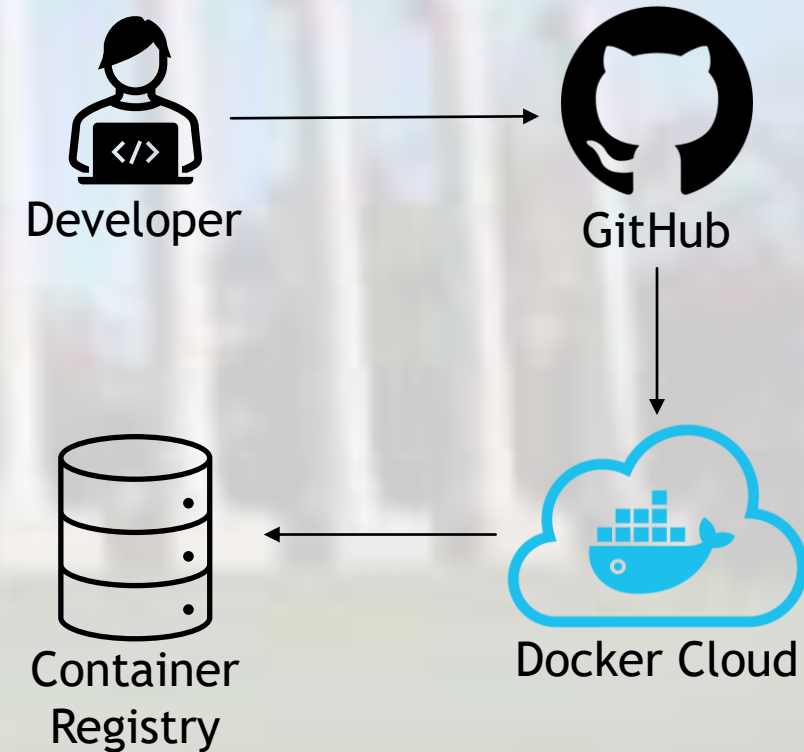
Automation with GitLab

▶ GitLab is a common tool for CI/CD with Docker because it contains VCS, CI/CD and a built-in image registry with access restrictions

  ▶ Only those added to the repository can access the code, CI/CD build instructions, and the container registry

▶ In our previous example, the VCS, CI/CD, and Registry providers were all independent, but GitLab contains all three of these on a per-repository or per-group basis

University of Missouri

34
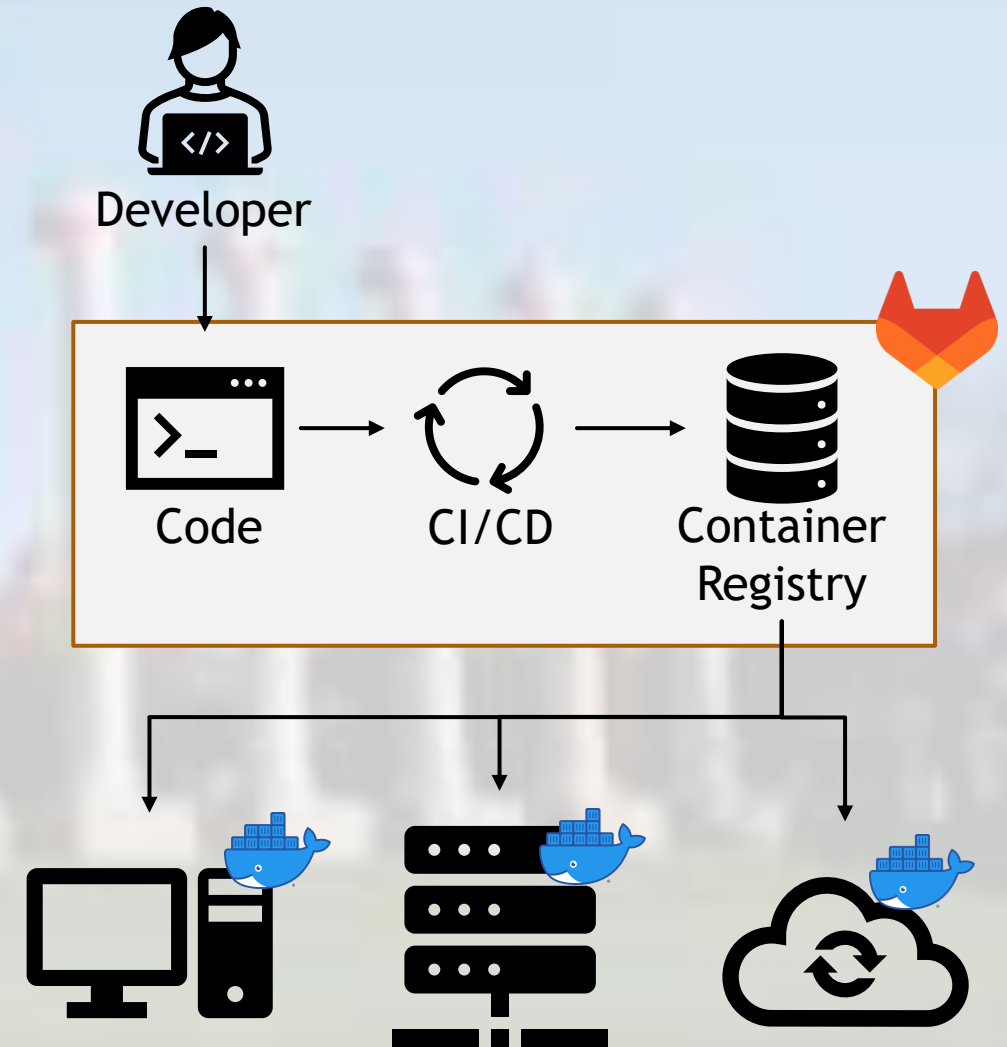
# Automation in Action:
GitLab CI + Registry for Python Application

▶ We configure GitLab to utilize CI/CD and the Container Registry to automatically build and push a container each time we push a new commit to the repository

▶ Each commit has its own tag, and latest is automatically updated to point to the most recent commit

▶ Let's view the repository, the Gitlab CI config, and the container registry!

Code → CI/CD → Container Registry

https://gitlab.nrp-nautilus.io/jhurt/python-ci

University of Missouri
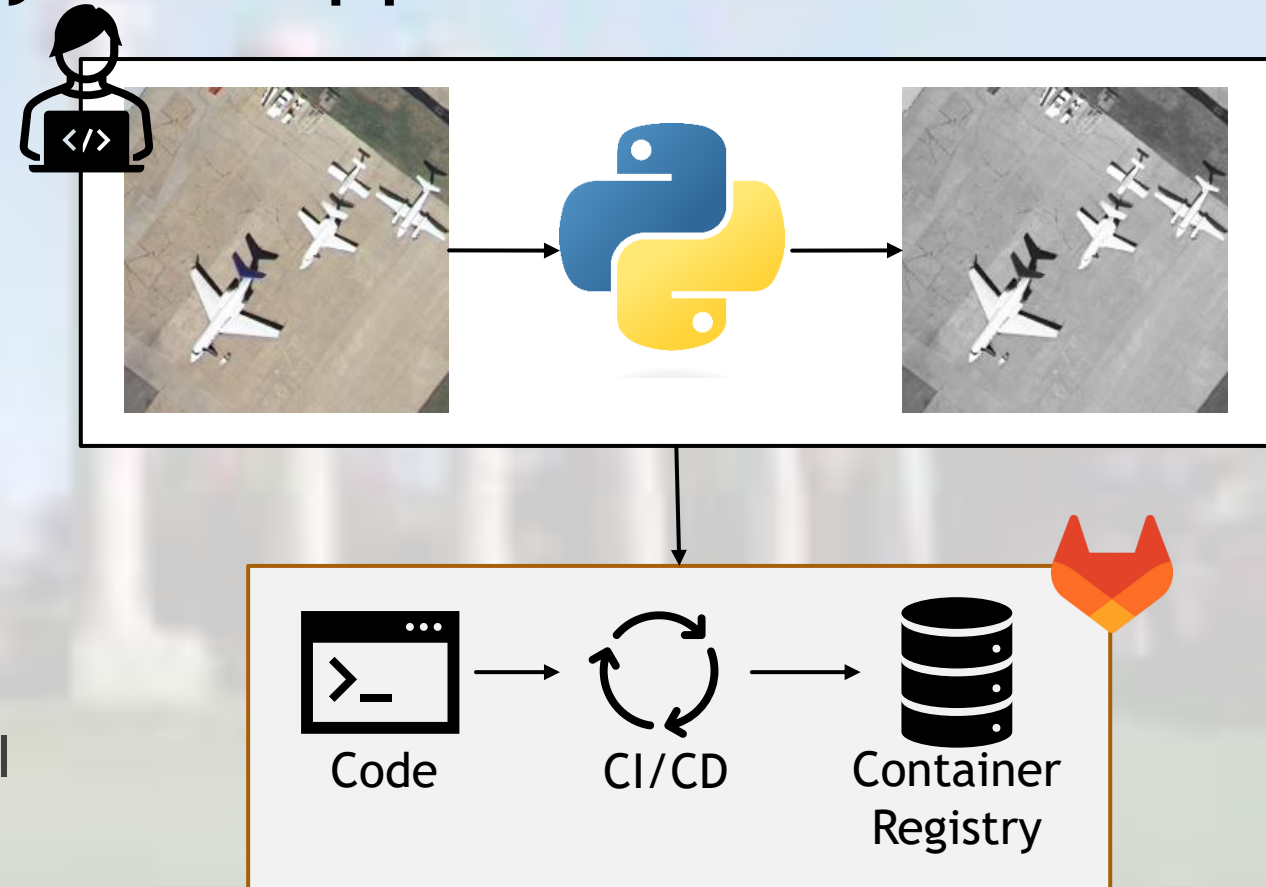
Utilizing Multiple Containers: Docker Compose

▶ Microservices and other software design paradigms can be developed and even deployed using multiple docker containers

▶ One common use case for this is a client-server model, such as a web server

▶ Docker compose is a tool that can enable robust management of multiple containers using YAML specified configuration

Docker Compose YAML

Docker Compose

Service 1   Service 2   Service 3

University of Missouri
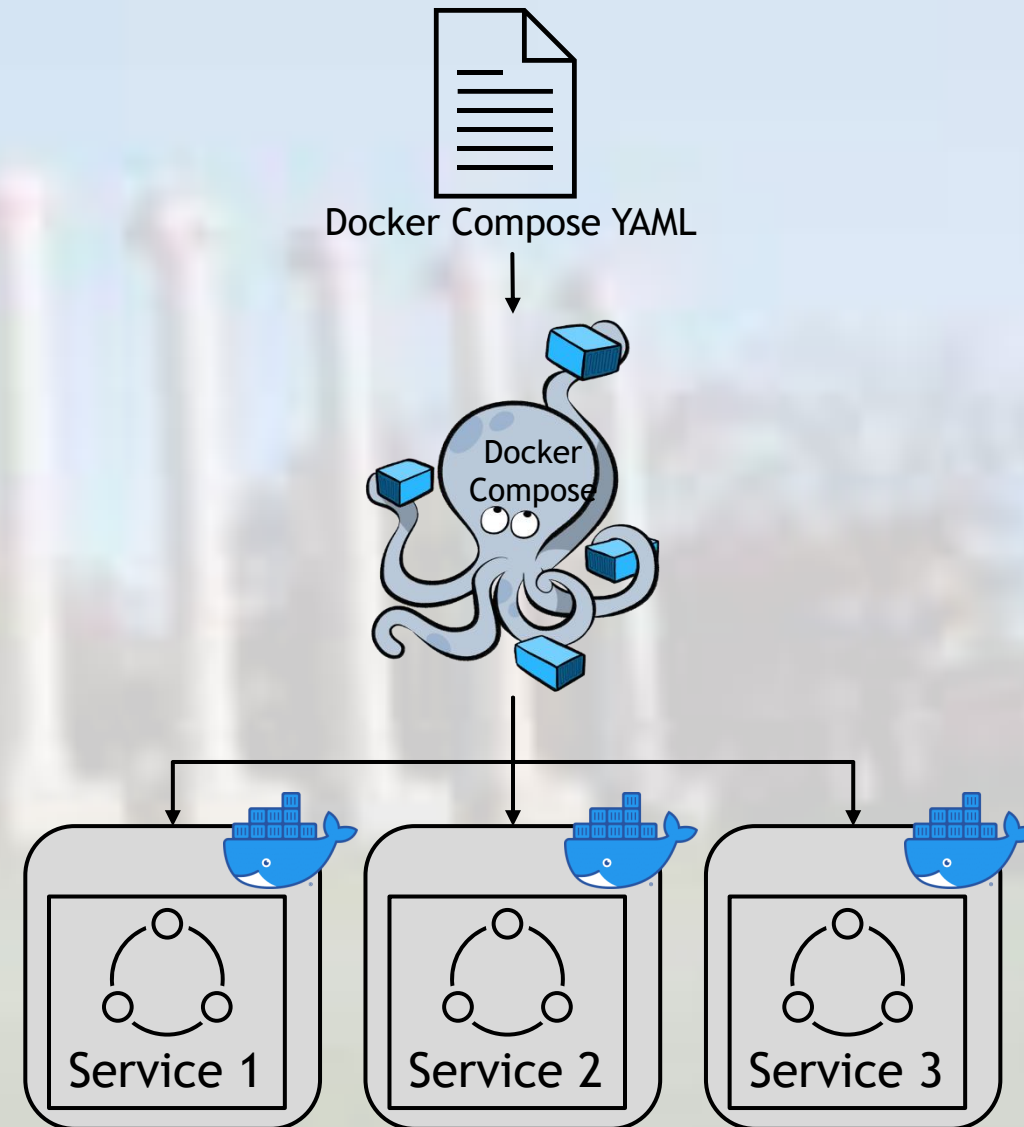
Multi-Stage Builds

▶ In some use-cases, we need to change the base container of our image but keep much of the work performed thus far in the image building process

▶ Alternatively, say there are libraries we need to build our library or application, but we do not need those libraries at runtime

▶ For both of these use-cases, we can utilize **multi-stage** builds to build multiple container images from a single Dockerfile

University of Missouri

Multi-Stage Builds Example 1: Compiling a C++ Executable

▶ Suppose we have built an application in C++, and we want to distribute this application using Docker

▶ We need multiple build tools to generate the final executable, but we do not need those tools to be distributed with our final executable

```
FROM gcc:9 as build
RUN apt update && apt install –y cmake
RUN mkdir –p /app
COPY / /app

RUN mkdir –p /app/build
WORKDIR /app/build
RUN cmake .. && make

FROM gcc:9 as dist

RUN mkdir –p /workspace

COPY –-from=build /app/build/myexecutable /workspace

CMD /workspace/myexecutable
```

University of Missouri

Using Multi-Stage Builds

▶ Once we have built a Dockerfile utilizing a multi-stage build, we can specify which stage to build:

docker build

–-file Dockerfile

–-target dist

–-tag myContainer:dist .

```
FROM gcc:9 as build
RUN apt update && apt install –y cmake
RUN mkdir –p /app
COPY / /app

RUN mkdir –p /app/build
WORKDIR /app/build
RUN cmake .. && make

FROM gcc:9 as dist

RUN mkdir –p /workspace

COPY –-from=build /app/build/myexecutable /workspace

CMD /workspace/myexecutable
```

University of Missouri

Multi-Stage Builds Example 2: Development vs. Production

▶ We may want to build a development container with additional tools to help us during the development process that we do not want to deploy to production

▶ We can use a multi-stage build to install only what we need for each

▶ We can again utilize the -–target parameter to docker build to specify which stage to build.

▶ Multi-stage builds can be further extended to CI/CD and automation pipelines and/or docker-compose architectures and microservices

```
FROM jupyter/datascience-notebook:python-3.10.9 as dev

RUN mkdir –p /app
COPY / /app

RUN pip install /app/requirements.txt

CMD /bin/bash


FROM python:3.10 as prod

COPY –-from=dev /app /app

RUN pip install /app/requirements.txt

CMD /app/myscript.py
```

University of Missouri

Docker Containers in the Cloud

▶ Containerization can be used to ensure portability to many kinds of computing environments, including cloud computing

▶ All major commercial cloud platforms offer services to deploy containers:

   ▶ Amazon Web Services (AWS)

   ▶ Google Cloud Platform (GCP)

   ▶ Microsoft Azure

University of Missouri

AWS Containers services

Sub-category	Use case	AWS service
Container orchestration	Run containerized applications or build microservices	Amazon Elastic Container Service (ECS)
	Manage containers with Kubernetes	Amazon Elastic Kubernetes Service (EKS)
Compute options	Run containers without managing servers	AWS Fargate
	Run containers with server-level control	Amazon Elastic Compute Cloud (EC2)
	Run fault-tolerant workloads for up to 90 percent off	Amazon EC2 Spot Instances
Tools & services with containers support	Quickly launch and manage containerized applications	AWS Copilot
	Share and deploy container software, publicly or privately	Amazon Elastic Container Registry (ECR)
	Application-level networking for all your services	AWS App Mesh
	Cloud resource discovery service	AWS Cloud Map
	Package and deploy Lambda functions as container images	AWS Lambda
	Build and run containerized applications on a fully managed service	AWS App Runner
	Run simple containerized applications for a fixed, monthly price	Amazon Lightsail
	Containerize and migrate existing applications	AWS App2Container
On-premises	Run containers on customer-managed infrastructure	Amazon ECS Anywhere
	Create and operate Kubernetes clusters on your own infrastructure	Amazon EKS Anywhere
Enterprise-scale container management	Automated management for container and serverless deployments	AWS Proton
	A fully managed, turnkey app platform	Red Hat OpenShift Service on AWS (ROSA)
Open-source	Run the Kubernetes distribution that powers Amazon EKS	Amazon EKS Distro
	Containerize and migrate existing applications	AWS App2Container

Part 1-B
Container management with Kubernetes

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023

University of Missouri

Subsection Outline

▶ Introduction to Kubernetes

▶ Kubernetes concepts

▶ Kubernetes usage

▶ Kubernetes hands on

University of Missouri

Introduction to Kubernetes

What is Kubernetes and what is it used for

University of Missouri

Kubernetes

▶ Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.[1]

▶ Kubernetes enables both simple and complex container orchestration

▶ Kubernetes cluster has two main components

  ▶ Master node

  ▶ Worker node

University of Missouri

1. https://kubernetes.io/
2. Image: https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams
3. Logo: https://commons.wikimedia.org/wiki/File:Kubernetes_logo_without_workmark.svg

49
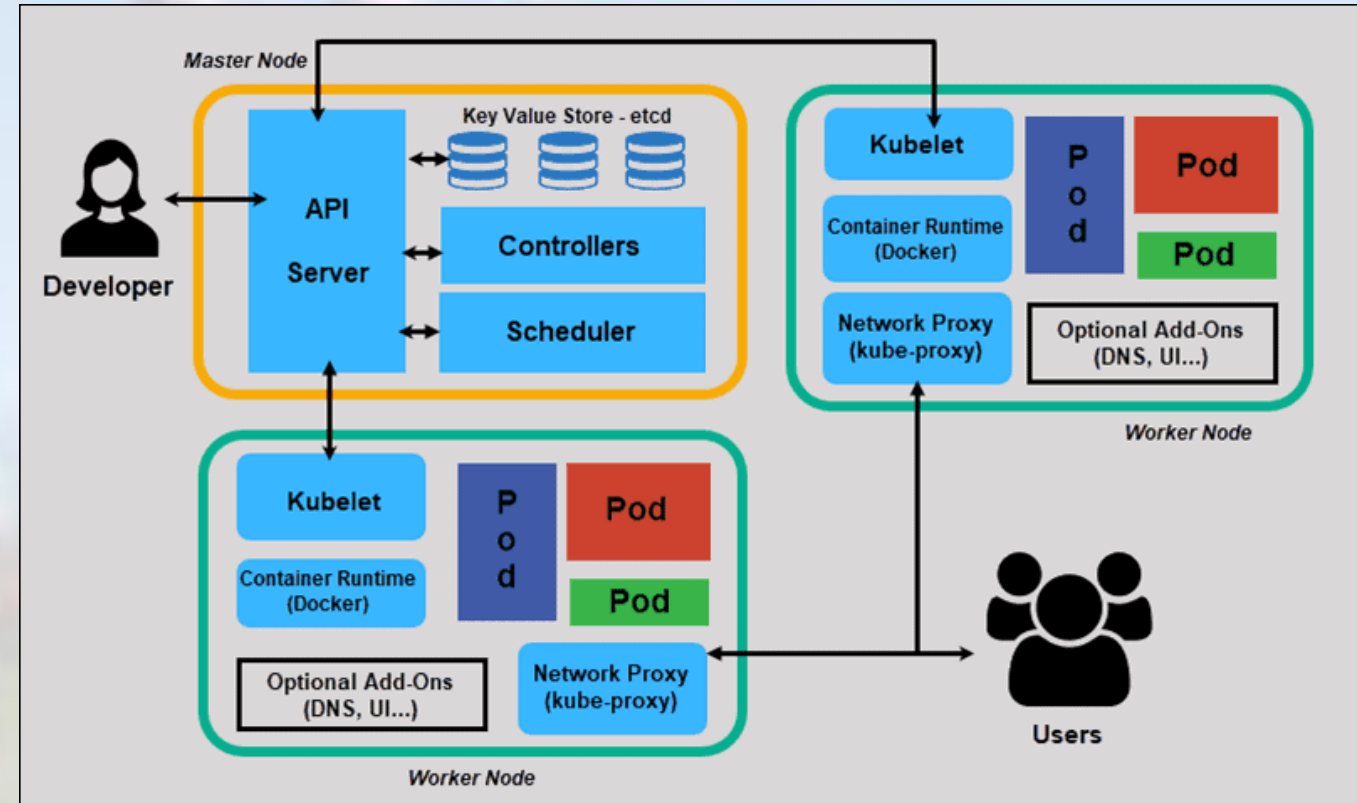
Kubernetes
Master node

▶ Also known as the Control Plane, it is responsible for managing the state of the cluster

▶ API server: interface between master node and the rest of the cluster

▶ etcd: distributed key-value store that stores the cluster's persistent information

▶ Scheduler: responsible for scheduling pods onto the working nodes

▶ Controller manager: responsible for running controllers that manages the state of the clusters such as replication controller and deployment controller

University of Missouri

1. https://kubernetes.io/
2. Image: https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams
3. Logo: https://commons.wikimedia.org/wiki/File:Kubernetes_logo_without_workmark.svg
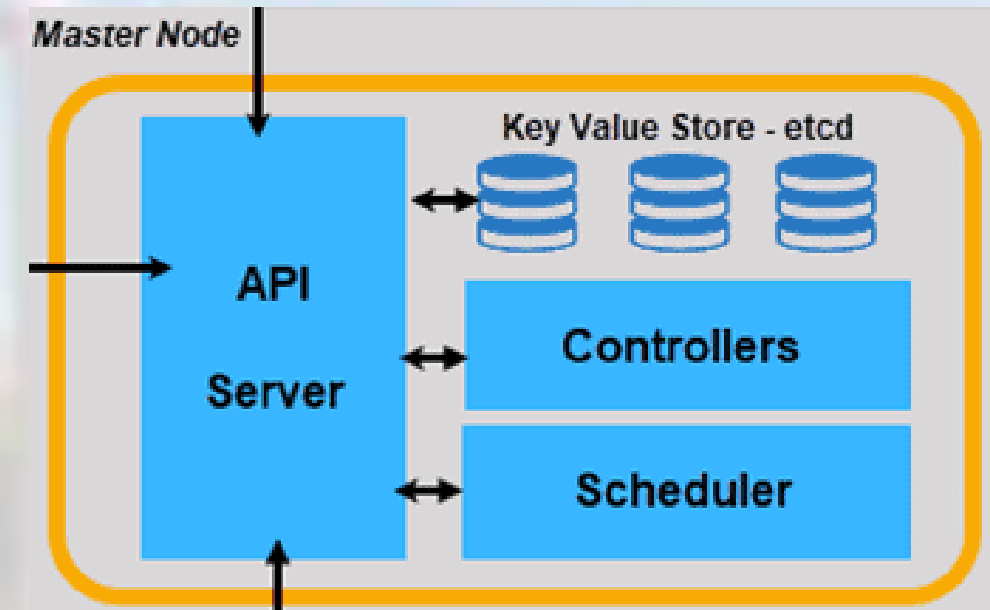
Kubernetes
Worker node

▶ The physical machine where the operations takes place, it can run one or multiple pods

▶ Kubelet: deamon that runs each working node

▶ Container runtime: is responsible for pulling images from the registry, starting and stopping containers, and managing the container resources

▶ kube-proxy: responsible for routing traffic to the correct pod and provides load balancing so that the traffic is distributed evenly between the pods

University of Missouri

1. https://kubernetes.io/
2. Image: https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams
3. Logo: https://commons.wikimedia.org/wiki/File:Kubernetes_logo_without_workmark.svg

High-Performance Data-Intensive
Computing Systems Laboratory
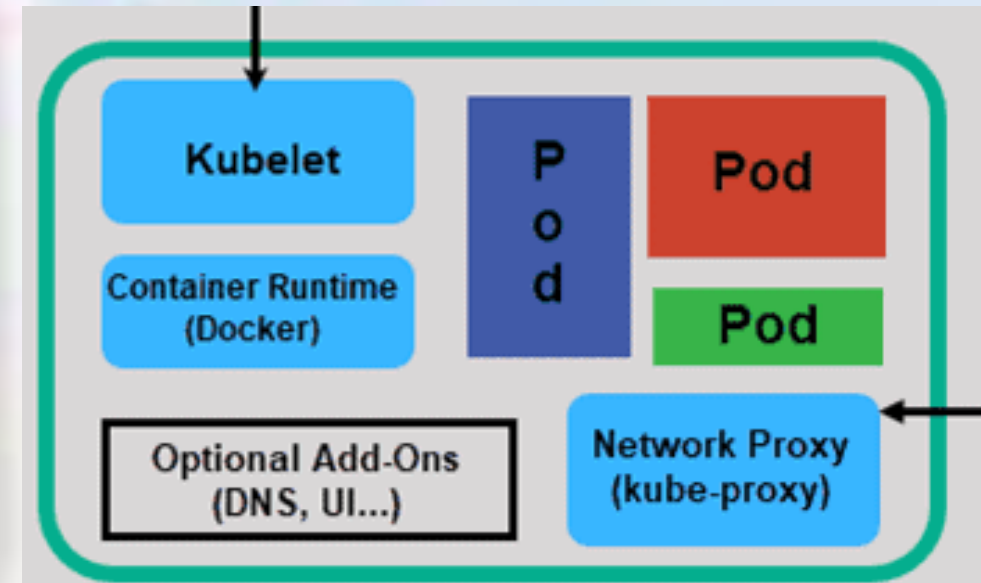
Kubernetes concepts

Node, pod, persistent volume, job, deployment, service

University of Missouri

Key Kubernetes Concepts
Pod

▶ **P**ods are the basic scheduling unit of K8s.

▶ Pods consist of one or more containers running inside. Each pod has a unique IP address to enable micro services or applications

▶ Pods can run custom scripts (initcontainer) at runtime to initialize the pod

▶ Pods generally have limitations on allocated resources and max runtime

▶ Pods are stateless, meaning all data uploaded or generated by the pod is deleted when the pod terminates

Container 1

Container 2

University of Missouri

53

Image: https://aws.plainenglish.io/kubernetes-deep-dive-job-and-cronjob-5ffed1c5fa4e
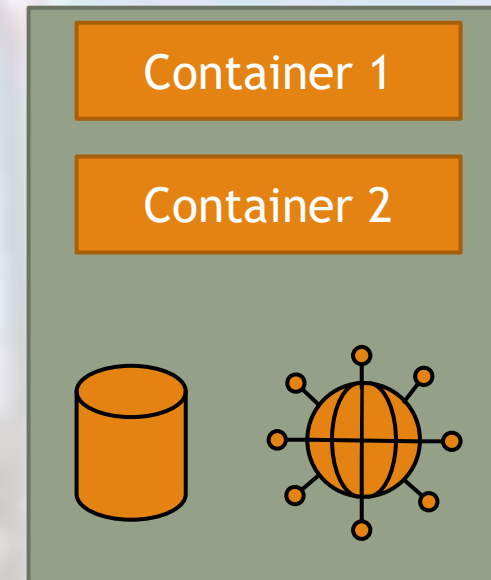
Key Kubernetes Concepts
ReplicaSet and Deployment

▶ **ReplicaSet -** its purpose is to maintain a stable set of replica Pods running at any given time. [1]

▶ **Deployment** - is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.[1]

Container 1	Container 1	Container 1
Container 2	Container 2	Container 2
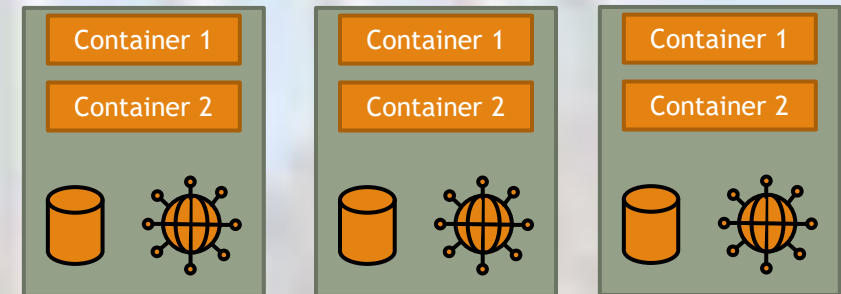
It is recommended to use Deployment instead of ReplicaSets

University of Missouri

1. https://kubernetes.io/

Key Kubernetes Concepts
Jobs

▶ A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate.[1]

▶ A job has virtually access to unlimited resources and can run for extended periods of time

▶ A job may consist of one pod or multiple pods working in parallel

▶ Deleting a job will automatically delete its corresponding pod

▶ A job can create a new pod(s) if any of its pod(s) is deleted or failed for any reason.

▶ Similar to pods, jobs are stateless

University of Missouri

1. https://kubernetes.io/

Key Kubernetes Concepts
Persistent volume

▶ To maintain the data generated a persistent volume (storage) is needed

▶ A **persistent volume** is storage on the cluster that has been provisioned by an administrator or dynamically provisioned using storage classes. [1]

▶ There exists different classes of persistent volumes such as:

   ▶ cephfs

   ▶ Fibre Channel storage

   ▶ NFS storage

▶ There are different access modes:

   ▶ ReadWriteOnce

   ▶ ReadOnlyMany

   ▶ ReadWriteMany

   ▶ ReadWriteOncePod

University of Missouri

1. https://kubernetes.io/

Key Kubernetes Concepts
Services

▶ Applications running within distinct pods communicate over the network using the unique IP address assigned to each pod

▶ Each Pod has a unique IP address assigned at runtime, which changes every time a Pod is restarted, making reliable communication less simple

▶ Services enable communication between applications running in pods within the cluster and with outside users if necessary

▶ There are four type of services supported by K8s

  ▶ ClusterIP
  ▶ NodePort
  ▶ LoadBalancer
  ▶ Ingress

University of Missouri
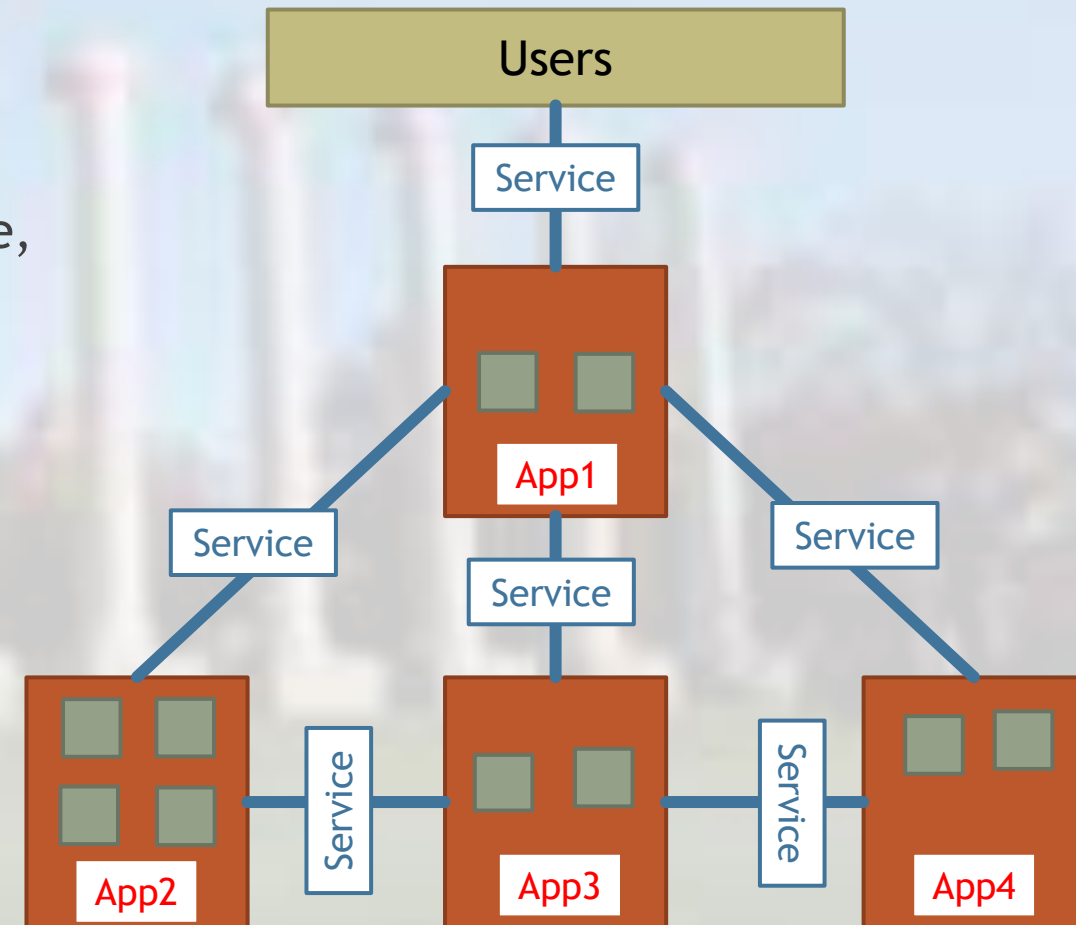
Kubernetes usage

Basics of YAML language, kubectl installation and usage

University of Missouri

Yet Another Markup Language (YAML)

XML	JSON	YAML

``
<Servers>
  <Server>
    <name>Server1</name>
    <owner>John</owner>
    <created>123456</created>
    <status>active</status>
  </Server>
</Servers>
``
	``
	{
	  Servers: [
	    {
	      name: Server1,
	      owner: John,
	      created: 123456,
	      status: active
	    }
	  ]
	}
	``
		``
		Servers:
		- name: Server1
		     owner: John
		     created: 123456
		     status: active
		``

▶ YAML is a key-value pair file format, similar to JSON and XML

▶ Kubernetes operations are performed using **YAML** files, known as a Spec file

  ▶ Creating Persistent Storage

  ▶ Creating Pods

  ▶ Creating Jobs

  ▶ Deploying services

University of Missouri

Synopsis of YAML language[1]

▶ Comments in YAML begins with the (#) character and they must be separated from other tokens by whitespaces.

▶ Indentation of whitespace is used to denote structure.

▶ Tabs are not included as indentation for YAML files.

▶ Lists are important

    ▶ List members are denoted by a leading hyphen (-).

    ▶ List members are enclosed in square brackets and separated by commas.

    ▶ YAML always requires colons and commas used as list separators followed by space with scalar values.

▶ Associative arrays are represented using colon (:) in the format of key value pair. They are enclosed in curly braces {}.

University of Missouri

1. https://www.tutorialspoint.com/yaml/yaml_basics.htm

Structure of YAML file

▶ Each YAML file consists of specific parts, each part is dedicated to hold a type of information that enables us to communicate our needs to the Kubernetes cluster.

▶ **kind** – tells Kubernetes the type of the resource: pod, job, service, storage

▶ **metadata** – data about the resource

    ▶ Name is most common attribute to set

▶ **spec** – the attributes specified in this section depends on the kind

    ▶ We will cover common spec attributes

University of Missouri

Config file (YAML)
Pod without mounted volume

We begin by setting the API Version and the type of object we are creating (Pod), as well as the name of the pod

From here we are defining the container to run in this pod

Set the name of the container, the image the container should run, and the command that should run when the container begins

Here, we define the requested and maximum amount of resources our container needs to run, in this case that is 2 CPU cores and 4 GB of RAM

```yaml
apiVersion: v1

kind: Pod

metadata:

  name: python3-pod

spec:

containers:

  - name: python3-container

    image: python:3.8

    command: ["sleep", "infinity"]

    resources:

     limits:

      memory: 4Gi

      cpu: 2

     requests:

      memory: 4Gi

      cpu: 2
```

University of Missouri

Config file (YAML)
Job without mounted volume

We begin by setting the API Version and the type of object we are creating (Job), as well as the name of the job

From here we are defining the pod that is to be started by this job

Set the name of the container, the image the container should run, and the command that should run when the container begins

Here, we define the requested and maximum amount of resources our container needs to run, in this case that is 2 CPU cores and 4 GB of RAM

```yaml
apiVersion: v1

kind: job

metadata:
 name: python3-job

spec:

 template:

   containers:

   - name: python3-pod

     image: python:3.8

     command: ["sleep", "infinity"]

     resources:

       limits:

           memory: 4Gi

         cpu: 2

       requests:

           memory: 4Gi

         cpu: 2
```

University of Missouri

63

Config file (YAML)
Persistent volume (PVC)

We begin by setting the API Version and the type of object we are creating (PersistentVolumeClaim), as well as the name of the PersistentVolumeClaim

From here we are defining the lass of the persistent volume we are using

Here we define the type of access mode of the persistent volume we are creating

Here, we define the requested size of the persistent volume 50 GB of storage

To increase the size of the persistent volume storage we only need to modify this value
We can only increase but we cannot decrease

```
apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: your_name

spec:

  storageClassName: rook-cephfs

  accessModes:

  - ReadWriteMany

  resources:

    requests:

      storage: 50Gi
```

University of Missouri

Config file (YAML)
Pod with mounted volume

We begin by setting the API Version and the type of object we are creating (Pod), as well as the name of the pod

From here we are defining the container to run in this pod

Set the name of the container, the image the container should run, and the command that should run when the container begins

Here, we define the requested and maximum amount of resources our container needs to run, in this case that is 2 CPU cores and 10 GB of RAM

Information of the mounted volume and how it is defined within the pod

```yaml
apiVersion: v1
kind: Pod

metadata:
  name: pod-name-sso

spec:
  containers:
  - name: pod-name-sso
    image: python:3.8
    command: ["sh", "-c", "echo 'Im a new pod' && sleep infinity"]
    resources:
      limits:
        memory: 12Gi
        cpu: 2
      requests:
        memory: 10Gi
        cpu: 2
    volumeMounts:
    - mountPath: /data
      name: anes-pv
  volumes:
  - name: anes-pv
    persistentVolumeClaim:
      claimName: anes-pv
```

University of Missouri

Config file (YAML)
Job with mounted volume

We begin by setting the API Version and the type of object we are creating (Job), as well as the name of the job

From here we are defining the pod that is to be started by this job

Set the name of the container, the image the container should run, and the command that should run when the container begins

Here, we define the requested and maximum amount of resources our container needs to run, in this case that is 2 CPU cores and 4 GB of RAM

Information of the mounted volume and it is defined within the job

```
apiVersion: v1

kind: job

metadata:
  name: python3-job

spec:

  template:

    containers:

    - name: python3-pod

      image: python:3.8

      command: ["sleep", "infinity"]

      resources:

        limits:

          memory: 4Gi

          cpu: 2

        requests:

          memory: 4Gi

          cpu: 2

      volumeMounts:

      - name: canada

        mountPath: /Canada

    volumes:

    - name: canada

      persistentVolumeClaim:

        claimName: canada2019-3
```

University of Missouri

66

Interfacing with Kubernetes: KubeCTL installation

- https://kubernetes.io/docs/tasks/tools/

- This link provide options for installing kubectl with:
    - Linux
    - macOS
    - windows

University of Missouri

Interfacing with Kubernetes:
KubeCTL installation (Linux)

there are three options to installing kubectl on Linux

- Install kubectl binary with curl on Linux

- Install using native package management

- Install using other package management

University of Missouri

Interfacing with Kubernetes:
KubeCTL installation (Windows)

▶ There are two options to install Kubectl on windows

- Install kubectl binary with curl on Windows

- Install on Windows using Chocolatey, Scoop, or winget

University of Missouri

Interfacing with Kubernetes:
KubeCTL installation (MacOS)

▶ There are many options to install Kubectl on MacOS

- Install kubectl binary with curl on macOS

- Install with Homebrew on macOS

- Install with Macports on macOS

University of Missouri

Interfacing with Kubernetes: KubeCTL

▶ With a published Docker image and prepared YAML Spec file, KubeCTL enables interaction with Kubernetes:

kubectl [command] [TYPE] [NAME] [flags]

where:

- ▶ command: Specifies the operation that you want to perform on one or more resources, for example create, get, describe, delete

- ▶ TYPE: Specifies the resource type, such as pod or job

- ▶ NAME: Specifies the name of the resource, or the path to a Spec file

- ▶ flags: Specifies optional flags, such as --server to specify the address and port of the API server

University of Missouri

Interfacing with Kubernetes: KubeCTL cheat sheet

▶ To create pod

kubectl create –f pod.yaml

kubectl apply –f pod.yaml

▶ To create job

kubectl create –f job.yaml

kubectl apply –f job.yaml

University of Missouri

Interfacing with Kubernetes: KubeCTL cheat sheet

▶ **To check pod status**

kubectl get pods

kubectl describe pod pod-name

▶ To check job status

kubectl get jobs

kubectl describe job job-name

▶ To debug pod

kubectl logs pod-name

University of Missouri

Interfacing with Kubernetes: KubeCTL cheat sheet

▶ **Access Pod interactively**

kubectl **exec -it** pod-name -- /bin/bash

▶ **Copy data from Nautilus to local machine**

kubectl cp pod-name:path/to/data local/path/

▶ **Copy data to Nautilus from local machine**

kubectl cp local/path/ pod-name:path/to/data

▶ **Exit interactive Pod mode**

Press ctrl+D

University of Missouri

High-Performance Data-Intensive
Computing Systems Laboratory

Interfacing with Kubernetes: KubeCTL cheat sheet

▶ To delete pod

kubectl delete –f pod.yaml

Kubectl delete pod-name

▶ To delete job

kubectl delete –f job.yaml

Kubectl delete job-name

University of Missouri

Interfacing with Kubernetes: KubeCTL cheat sheet

▶ To create persistent volume

kubectl create –f pvc.yaml

Kubectl apply –f pvc.yaml

▶ To increase the size of persistent volume

kubectl apply –f pvc.yaml

▶ To delete persistent volume

kubectl delete –f pvc.yaml

kubectl delete pvc-name

University of Missouri

Part 1-C
National Research Platform
Nautilus Research Cluster

Introduction to Containers and Kubernetes

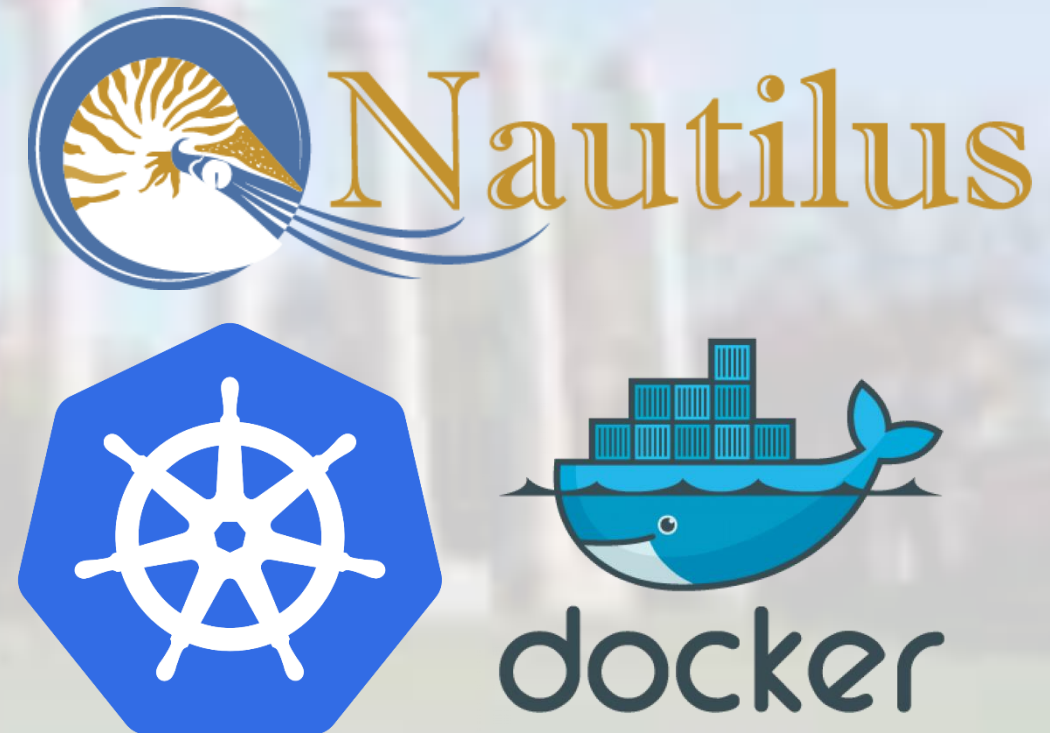IEEE International Conference on Big Data

December 2023

University of Missouri

A quick note on Kubernetes Clusters, the NRP, Commercial Clouds, and other K8s Clusters

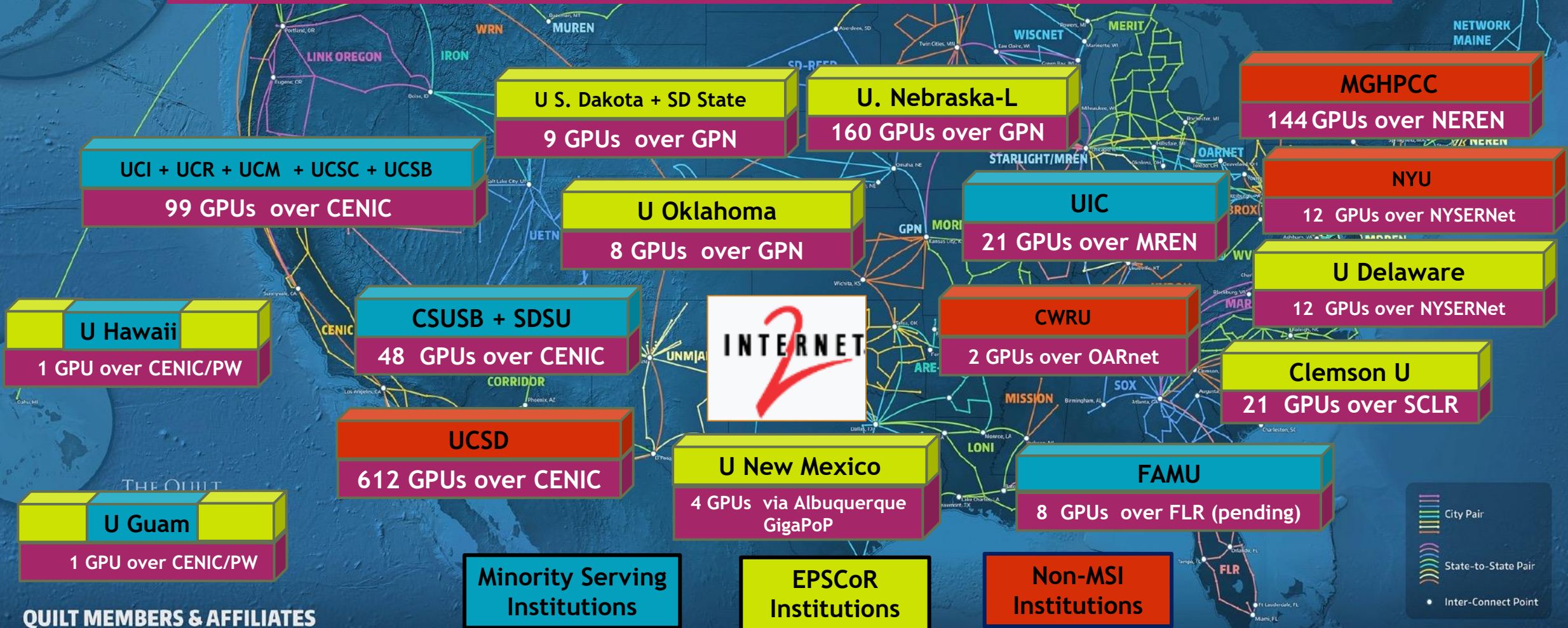▶ All commercial cloud providers support Containers and Kubernetes

▶ The concepts and examples in this tutorial may require minor modifications to adapt to other environments

▶ We are using the US National Research Platform solely for demonstration and tutorial purposes

▶ All Container and Kubernetes concepts are portable to commercial clouds or other research Kubernetes platforms

University of Missouri

NSF NRP Nautilus HyperCluster

► The NSF Nautilus HyperCluster is a Kubernetes cluster with vast resources that can be utilized for various research purposes:

  ► Prototyping research code

  ► S3 cloud storage for data and models

  ► Accelerated small-scale research compute

  ► Scaling research compute for large scale experimentation

► Resources Available:

  ► CPU Cores: 14,462

  ► RAM: 69 TB

  ► NVIDIA GPUs: 1150

University of Missouri

Nautilus Logo: https://portal.nrp-nautilus.io/
Docker Logo: https://developers.redhat.com/blog/2014/05/15/practical-introduction-to-docker-containers
Kubernetes Logo: https://commons.wikimedia.org/wiki/File:Kubernetes_logo_without_workmark.svg

79

Nautilus ~1,100 GPUs Distributed over US Networks—Fall 2022

U S. Dakota + SD State — 9 GPUs over GPN

U. Nebraska-L — 160 GPUs over GPN

MGHPCC — 144 GPUs over NEREN

UCI + UCR + UCM + UCSC + UCSB — 99 GPUs over CENIC

U Oklahoma — 8 GPUs over GPN

UIC — 21 GPUs over MREN

NYU — 12 GPUs over NYSERNet

U Delaware — 12 GPUs over NYSERNet

U Hawaii — 1 GPU over CENIC/PW

CSUSB + SDSU — 48 GPUs over CENIC

CWRU — 2 GPUs over OARnet

Clemson U — 21 GPUs over SCLR

UCSD — 612 GPUs over CENIC

U New Mexico — 4 GPUs via Albuquerque GigaPoP

FAMU — 8 GPUs over FLR (pending)

U Guam — 1 GPU over CENIC/PW

Minority Serving Institutions

EPSCoR Institutions

Non-MSI Institutions

City Pair
State-to-State Pair
Inter-Connect Point

QUILT MEMBERS & AFFILIATES

5 PB Nautilus Ceph Storage Over Networks—2022

UCR + UCSC + UCSB
940 TB over CENIC

UN-L
400 TB over GPN

UIC
175 TB over MREN

NYU + NYSERNet
400 TB over NYSERNet

USC + Stanford + Caltech
737 TB over CENIC

U Kansas
200 TB over GPN

SDSU
109 TB over CENIC

OneNet
200 TB over GPN

U Delaware
200 TB over NYSERNet

U Hawaii
266 TB over CENIC/PW

UCSD + UCLA
901 TB over CENIC

U Arkansas
200 TB over GPN

FAMU + FIU
318 TB over FLR

U Guam
118 TB over CENIC/PW

Minority Serving Institutions

EPSCoR Institutions

Non-MSI Institutions

City Pair

State-to-State Pair

Inter-Connect Point

Resource Dashboard

University of Missouri

Access to Nautilus

https://gp-engine.nrp-nautilus.io/
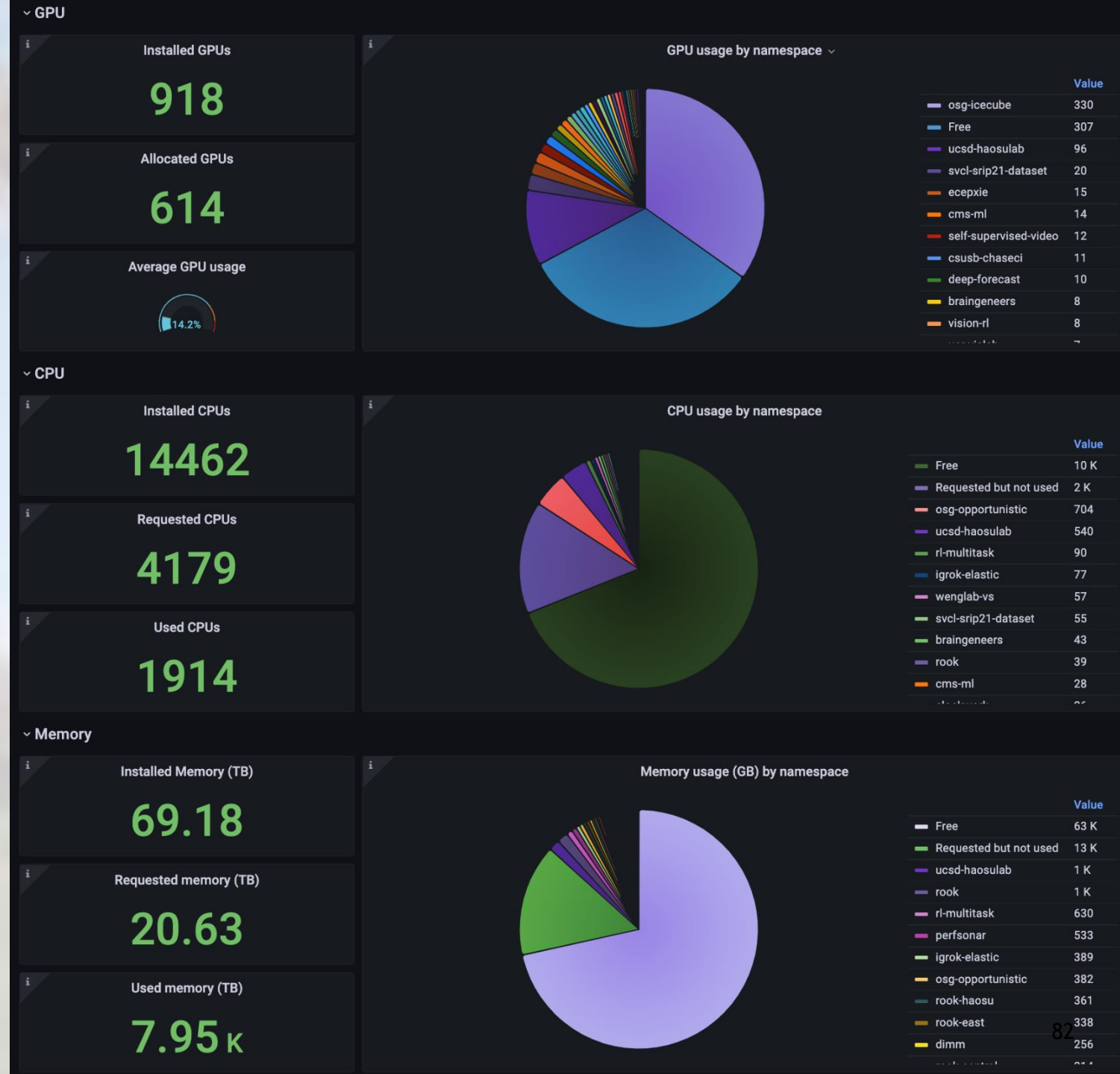
Nautilus Access

- **MU-managed Jupyter Hub**
  - Advantages
    - No manual account creation
    - Ease of use
  - Disadvantges
    - Limited customizability

- **Direct Access with KubeCTL**
  - Disadvantages
    - Account and namespace creation
    - KubeCTL installation
  - Advantages
    - High scalability and customizability

University of Missouri

Access to Nautilus

▶ Follow the steps in getting started

  ▶ https://ucsd-prp.gitlab.io/userdocs/start/get-access/

▶ Step1: Access Nautilus portal at https://portal.nrp-nautilus.io

▶ Step 2: Click on login

Namespaces overview   Resources   Login

NRP Kubernetes portal

Here you can get an account in National Research Platform kubernetes portal by logging in with your university's credentials and requesting access in [matrix]
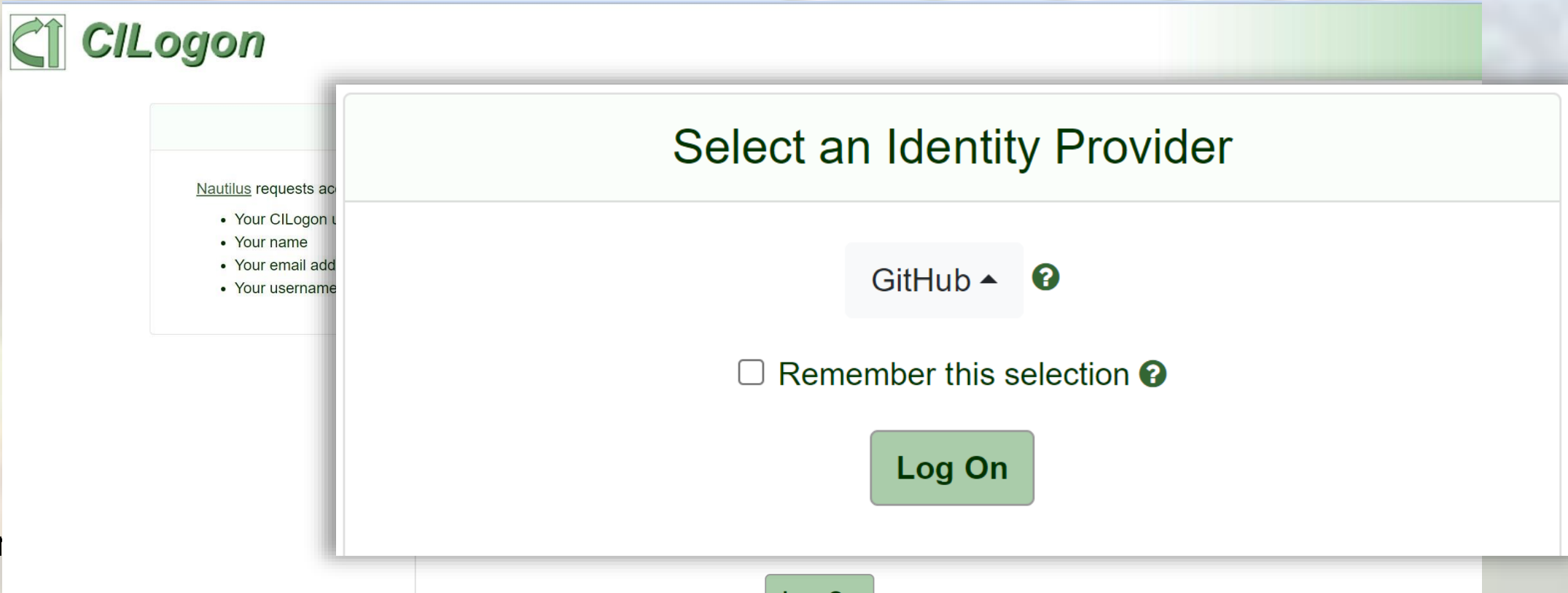
Documentation: https://docs.nationalresearchplatform.org/

You can easily join your node in our cluster - request instructions in [matrix] #general channel.

The National Research Platform currently has no storage that is suitable for HIPAA, PID, FISMA, FERPA, or protected data of any kind. Users are not permitted to store such data on NRP machines.
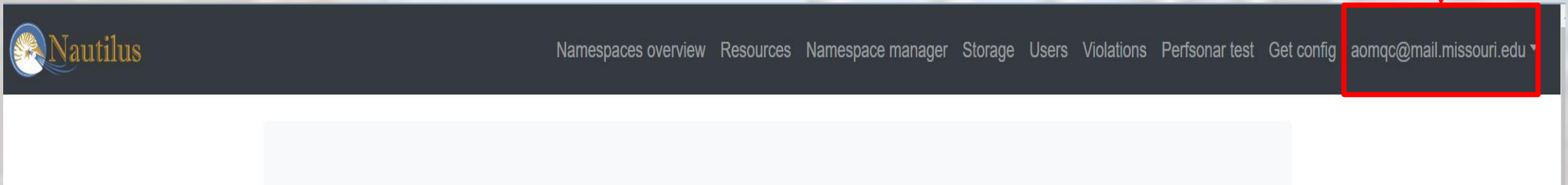
Access to Nautilus

▶ Follow the steps in getting started

   ▶ https://ucsd-prp.gitlab.io/userdocs/start/get-access/

▶ **Step 3:** Select identity provider – Either your institution, ORCID, GitHub, or Google

Access to Nautilus

▶ Follow the steps in getting started

    ▶ https://ucsd-prp.gitlab.io/userdocs/start/get-access/

▶ Step 4: Contact a Nautilus Namespace Admin

    ▶ Email needs to be visible

Nautilus   Namespaces overview   Resources   Namespace manager   Storage   Users   Violations   Perfsonar test   Get config   aomqc@mail.missouri.edu ▾

▶ You need to be manually added to a namespace

    ▶ As admins, we can add you to existing namespace or create a namespace for you

University of Missouri

Hands on Kubernetes

Life cycle of pods and jobs

University of Missouri

Hands on Kubernetes

▶ Demonstration of pod life cycle (without persistent volume)

  ▶ Creation, interactive access, simple operation, closing, and deletion

▶ Creation of persistent volume

▶ Demonstration of pod life cycle (with persistent volume)

  ▶ Creation, interactive access, simple operation, closing, and deletion

▶ Pod monitoring and debuging

▶ Demonstration of job life cycle

University of Missouri

MU-HPDI/Nautilus

▶ Sample Dockerfiles

▶ Sample Kubernetes YAML File

▶ Wiki with detailed walkthroughs for:

    ▶ Getting Started

    ▶ Creating PVC

    ▶ Creating Pods

    ▶ Creating Jobs

▶ Tutorial Repository of Jupyter Project Pages, Code Samples, YAML, etc.

https://github.com/MU-HPDI/bigdata-2023

▶ Git Clone Link:

https://github.com/MU-HPDI/bigdata-2023.git

Over a short break, we will ensure everyone has cloned this Repo into their JupyterLab environment

University of Missouri