

Before we start

- This will be recorded, so if you missed something, you will be able to come back to it
- There will be a Q&A at the end, so take note of any questions you have during the talk (or send them to me)
- You can ask questions both in voice chat or text chat, however, questions in voice chat will be recorded too
- Repository containing the files used can be found here: <https://github.com/GryPr/TT-DockerK8S>
- Enjoy!




Introduction to Docker and Kubernetes



kubernetes

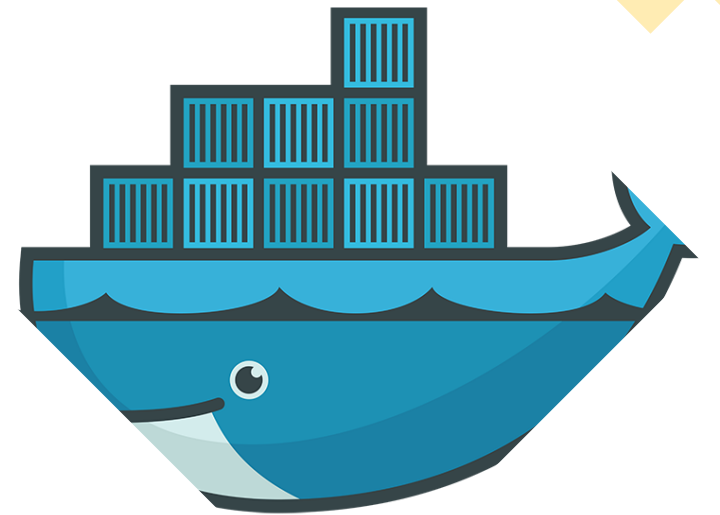


Who I am

- Second-year Computer Engineering student at Concordia University
 - Interests in technology both in modern software and hardware
 - Web dev
 - Cloud
 - Machine Learning
 - Security
 - Virtual Reality
 - 3D Printing
- 

Part 1 - Docker

- What is Docker?
- Containers vs Virtual Machines



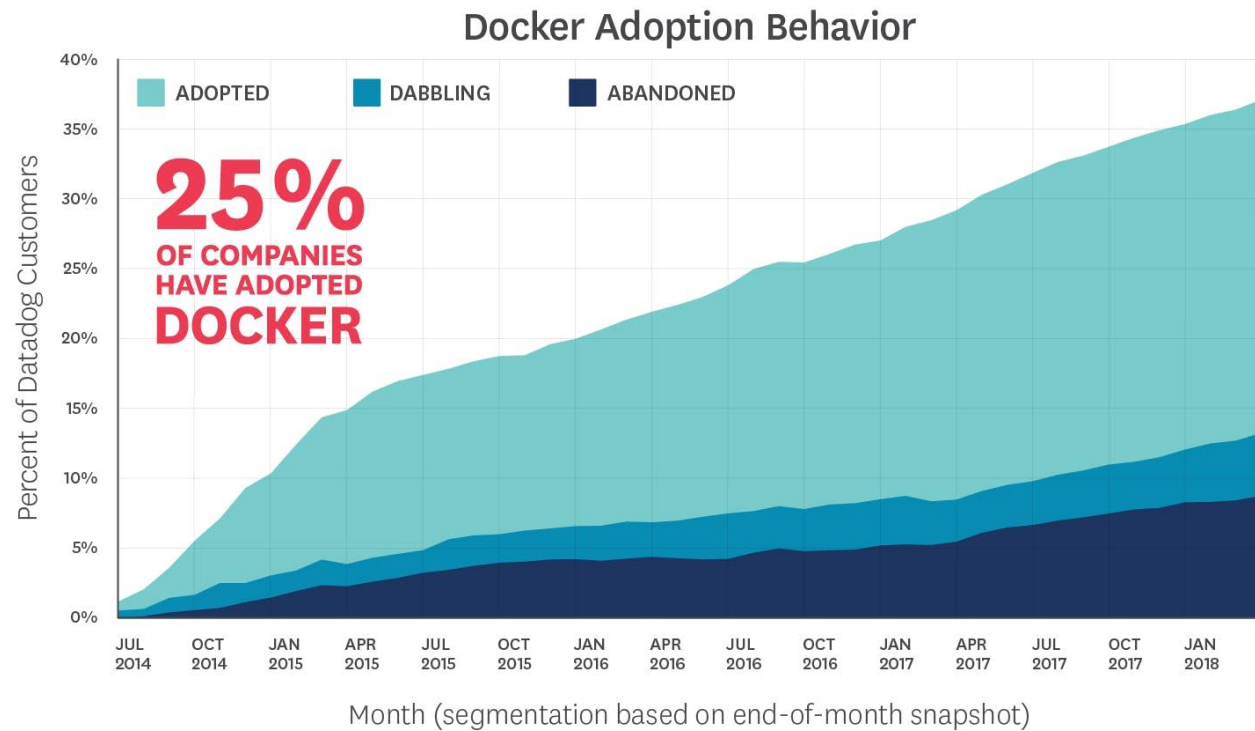


What is Docker?

- OS-level virtualization with similarities to virtual machines
- Allows delivery of software in the form of packages called 'containers'
- Containers run as a “stripped down OS” and are bundled with software, libraries and configuration files
- Multiple containers can run in parallel isolated to each other

Docker Statistics

- Currently the status quo in running containers
- Popularity has skyrocketed over the past 6 years

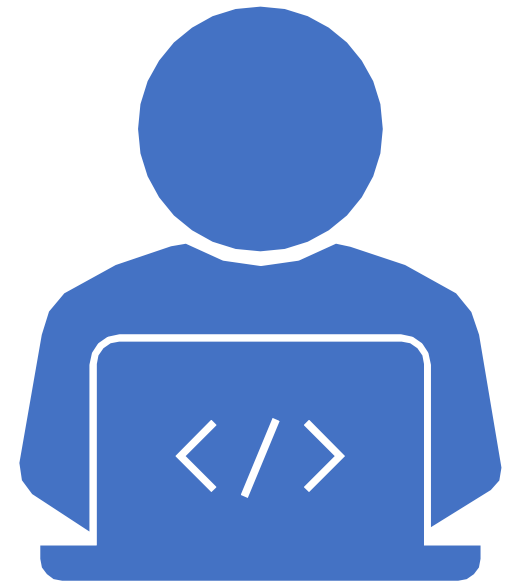


Source: Datadog

Towards Containerization

Why Docker?

- It allows for encapsulation of software
 - Everything needed to run the software is in the container
 - This container can then be run on any machine that has the Docker runtime installed
 - Much less time wasted on installing/configuring the proper dependencies, and dealing with conflicts
- Updates can be released quickly in the form of Docker images, and rolled back easily in case issues arise
- Compared to VMs, Docker uses resources more efficiently



Docker at home

Did you know that Docker is one of the best ways to deploy software at home?

Some examples of software you can deploy in containers on a Linux server machine:

- BitTorrent client (Qbittorrent, Deluge, etc...)
- Media server (Jellyfin, Emby, Plex)
- Game servers (Minecraft, CSGO, etc...)
- Home automation (Huginn, n8n)

Docker compose

- Docker compose is an easy way to declaratively deploy containers
- You write YAML files that declare all the parameters of the deployment
- Allows for the use of version control software such as Git
- Eases the deployment since all the parameters are in the YAML file, and deploying it can be simply done using docker-compose up
- Very useful to deploy single containers, or small group of containers (i.e. application and database)
- Other better software exist for more complex deployments of containers

```
1  ---
2  version: "2.1"
3  services:
4    deluge:
5      image: linuxserver/deluge
6      container_name: deluge
7      network_mode: host
8      environment:
9        - PUID=1000
10       - PGID=1000
11       - TZ=America/Toronto
12       - UMASK_SET=022 #optional
13       - DELUGE_LOGLEVEL=error #optional
14     volumes:
15       - /root/volumes/delugeConfig:/config
16       - /media/ubuntuServer/DownloadCache:/downloads
17     restart: unless-stopped
```

File format version

Service name

Image name from library

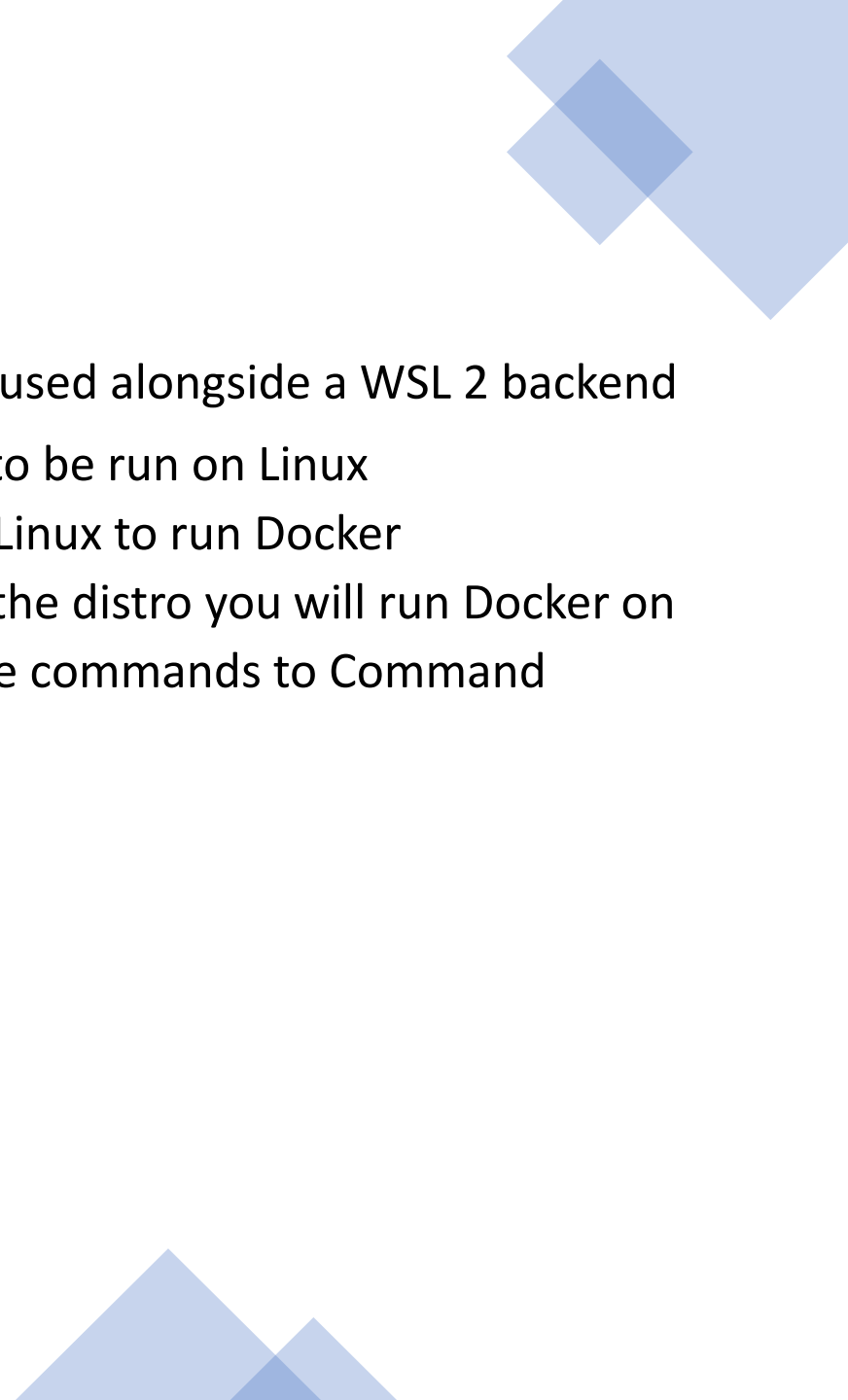
Environment variables

Volumes mounted into container

Docker- compose example

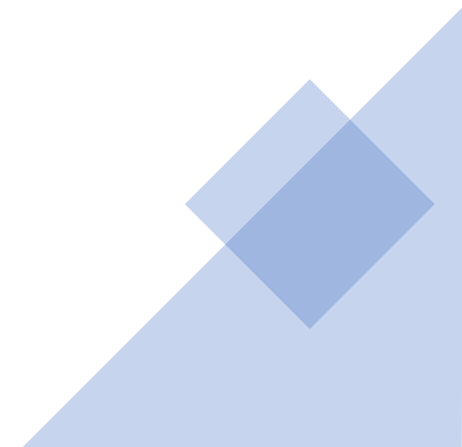


Docker Environment on Windows

- Docker Desktop for Windows is used alongside a WSL 2 backend
 - Be aware that Docker is meant to be run on Linux
 - Docker Desktop virtualizes Linux to run Docker
 - WSL 2 allows you to select the distro you will run Docker on
 - Docker Desktop exposes the commands to Command Prompt and Powershell
- 



Application to deploy

- We are deploying an endpoint at port 8000 of localhost
 - The app is written in Go with a dependency on gorilla/mux
 - gorilla/mux is not actually necessary for this app
 - Dependency is there only to show how to deal with dependencies in Docker
- 



Creating a Dockerfile

- More information at <https://docs.docker.com/engine/reference/builder/>



Docker Deployment Demo

Let's go an deploy our app on Docker!



Part 2 - Kubernetes

What is Kubernetes?

- Container Orchestration system
- Automatically deploys and manages containers
- Runs health checks, and restarts bad containers
- Automatic scaling based on demand
- YAML declarative management allows setup to be version controlled on Git



Why Kubernetes?

- To move towards a microservice architecture instead of the old monolithic architecture
 - Each part of the application can be modified, updated and scaled independently
 - This allows for more efficient distribution of resources towards components that may require it
 - Components of an app can be run on different machines, and resources can be allocated accordingly
 - This is especially important when hosting an application in the cloud
 - Companies are billed by resource consumption
 - Microservices reduce overall cost



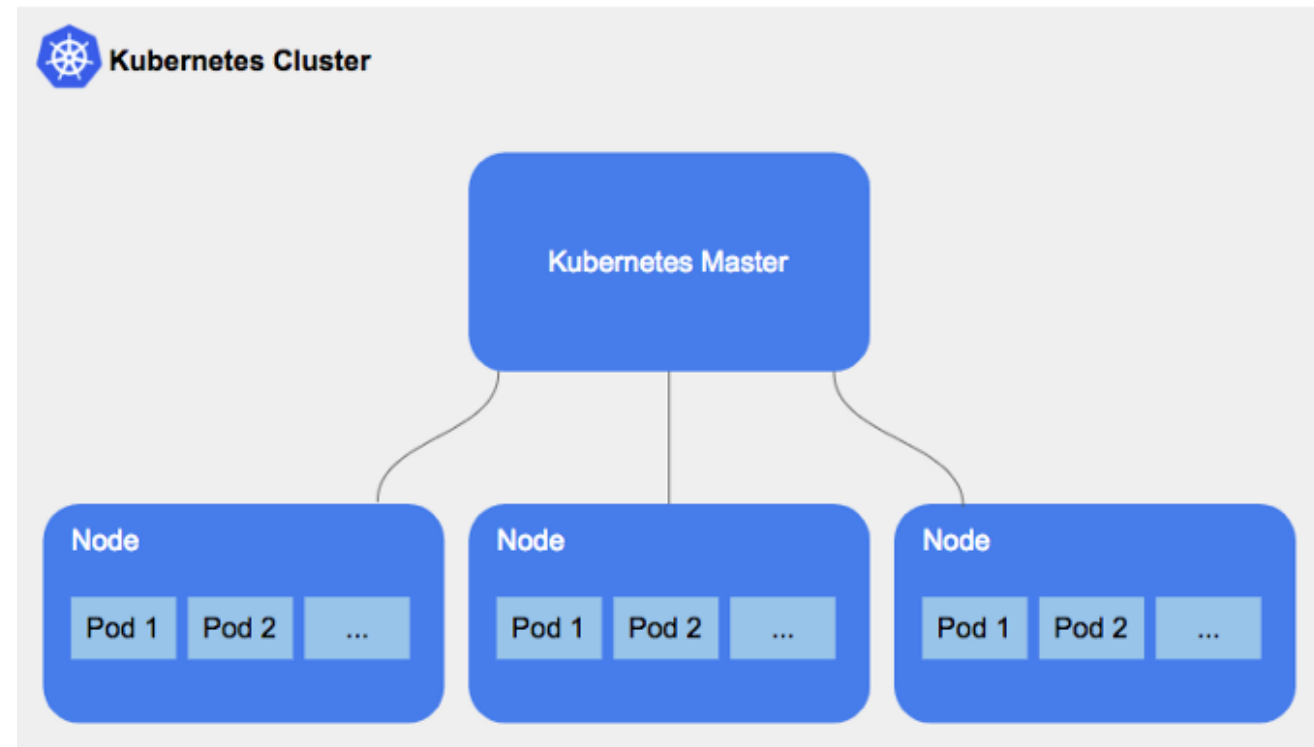
K8S Cluster

What is a cluster?

- Composed of nodes running containers
- These nodes can be physical or virtual machines
- There is a master node that controls the worker nodes

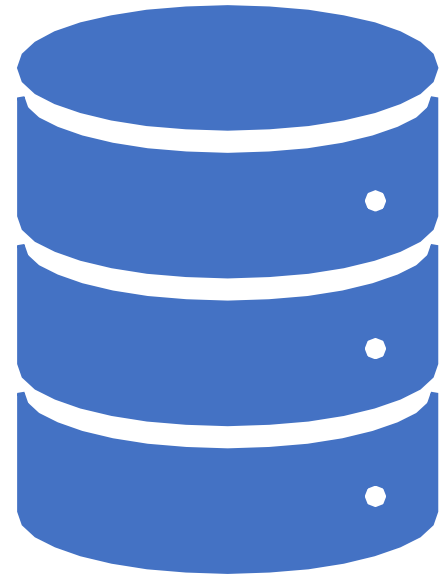
How can we run K8S locally?

- We use minikube which runs a single node cluster
- Typically run on a Linux virtual machine if you have a Windows dev environment



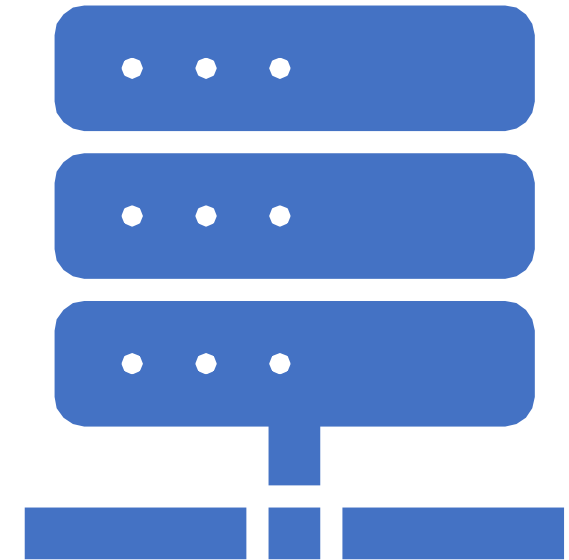
K8S Pods

- Essentially a group of one or more Docker containers
 - Usually only one container in a pod
- For horizontal scaling (getting more resources by running more instance), multiple pods are used instead
 - This is called replication, and can be specified in YAML
- You would usually deploy pods through other K8S resources like deployments
- A pod has an IP address, and all containers in a pod share it.



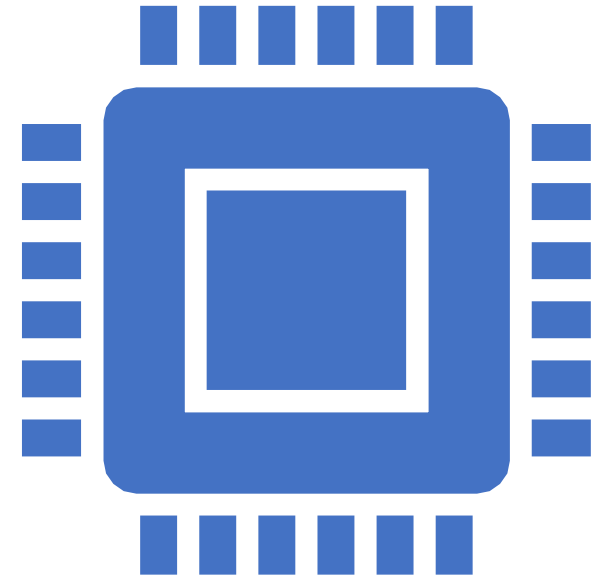
K8S Services

- Pods are stateless and can be dynamically destroyed and respawned
 - This changes the IP address of the pods
- Problem: what if a pod depends on another pod
 - i.e. Frontend & backend pods
 - How can it find a pod with an IP that always change?
- Services allow for the abstraction of pods.
 - It defines a set of pods, and how to access them.
- To access a pod, you would first pass through a service so that you can access the pod that provides you with what you need



K8S Deployment

- Provides a way to declare how pods are deployed
 - Which image is used
 - How many pod replicas
 - How they should be updated
- Allows deployment of pods to be repeatable and automated according to the YAML
- Allows you to automatically update the deployment using the K8S backend instead of manually doing it with scripts

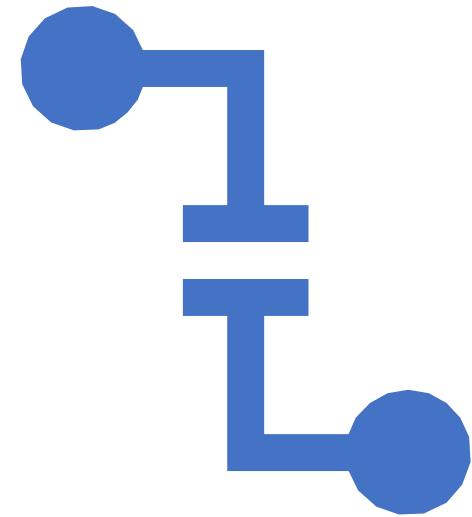


Namespaces

- It is a virtual cluster within the physical cluster
- It scopes the name of resources – the name must be unique in a given namespace, but not across multiple namespaces
- Resources can only belong to one namespace
- The most common use is simply as a way to organize resources in large clusters to facilitate administration
- It also permits role-based access control (RBAC) where people can be granted roles specific to certain namespaces, and not to other namespaces

Ingress

- Ingress is a Kubernetes object that manages external access to the Kubernetes cluster
- It exposes HTTP and HTTPS routes from outside towards services in the cluster
- It can give services an URL, and does not expose arbitrary ports. Instead, it can designate a path in the URL towards a service
 - i.e. `example.com/service1`, `example.com/service2`
- It can also be paired with load balancing to distribute incoming requests



Kubernetes Environment on Windows

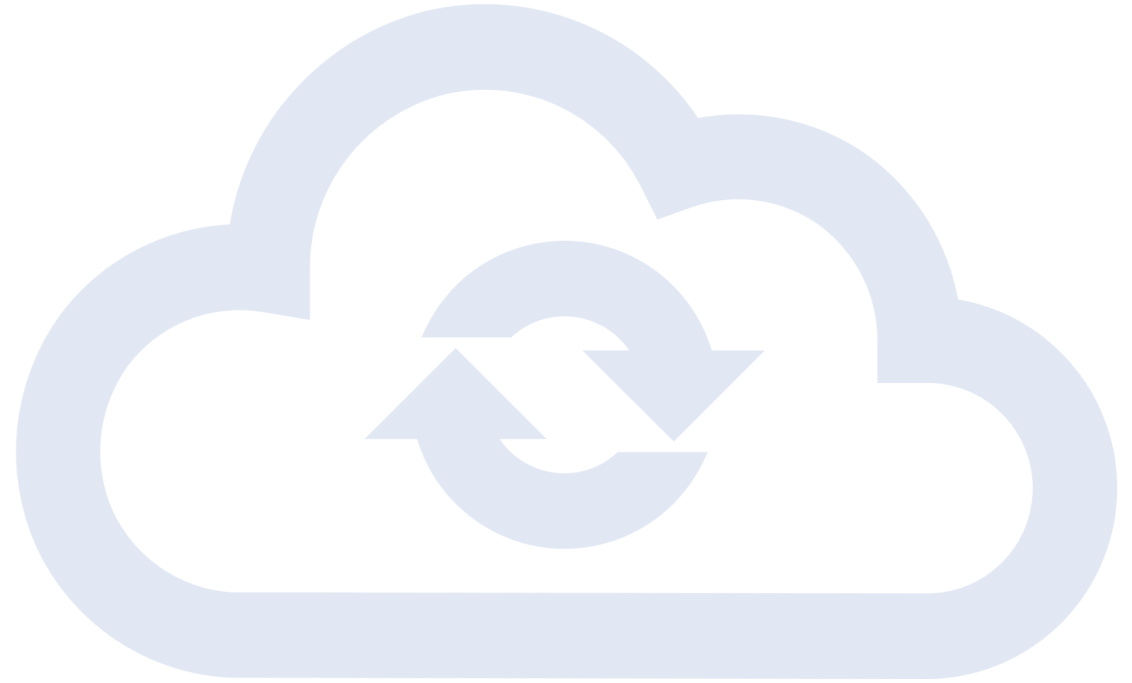
- Kubernetes is run on minikube, a tool for local K8S development
- Otherwise, the same environment is used as with Docker





Kubernetes Deployment Demo

Let's go an deploy our app on
Kubernetes!



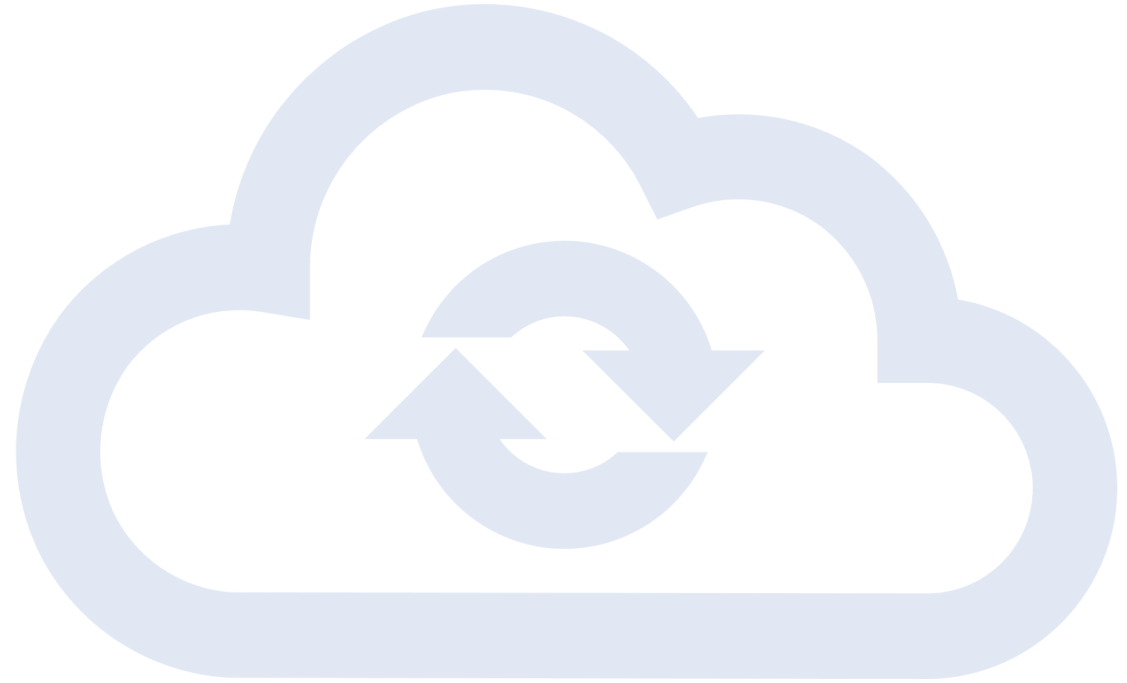


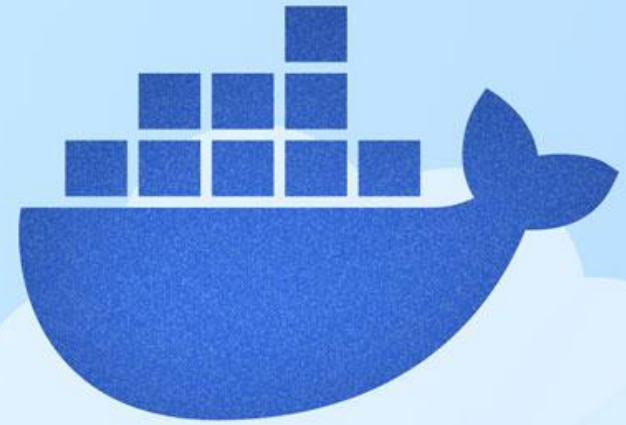
Helm



Helm Demo

Let's package our deployment with Helm!

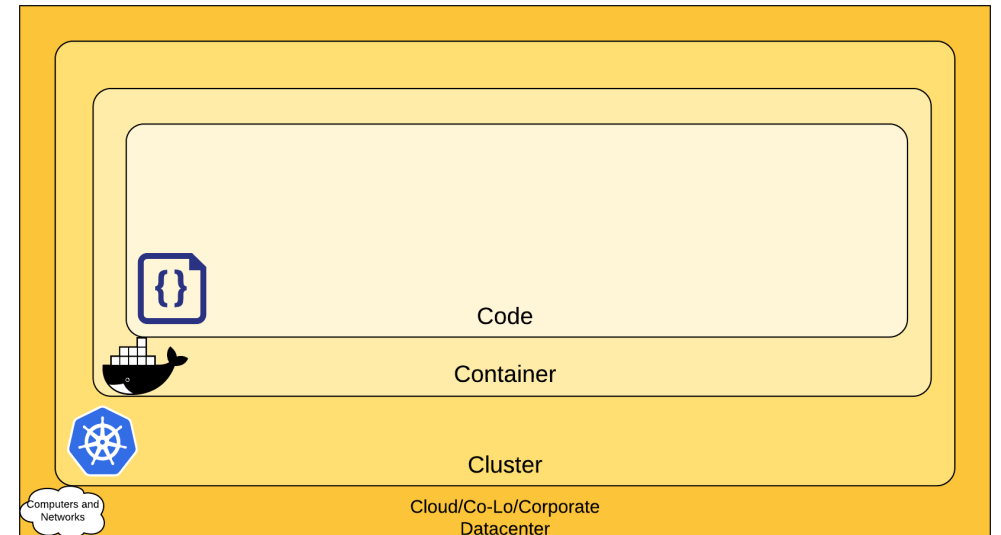




Security in Docker and Kubernetes

Four C's of Cloud Native Security

- Defense in depth by having more layers of security to penetrate
- The inner layers' security is complemented by the outer layers' security
- Each layer must implement proper security practices to reinforce the security of the whole software system



Docker Content Trust

- Transport of images often cannot be trusted
- How do we know if an image wasn't tampered in transit from a trusted source?
- **Solution: attaching digital signatures to tags of images when sending/receiving images to/from remote Docker registries.**
- These signatures can be used to verify on the receiving side to confirm the integrity and authenticity of the image



How does signing images work?

First, the publisher generates a pair of public/private cryptographic keys

- These are used to decrypt/encrypt data
- Data encrypted with the private key can only be decrypted by the public key, and vice-versa
- The private key is used to sign the tag of an image

What is a tag?

- Alias tied to an image ID

What is an image ID?

- Unique to the image's JSON configuration object
- SHA256 hash of the JSON
- Hash is completely different if even a single bit of the JSON config is changed

Basics of signature cryptography

How does the image publisher sign an image?

- We encrypt the image ID (SHA256 hash) of the image
- We then publish the public key for others to verify the image

How do users validate an image?

- Users verify the integrity of the image by verifying the hash
 - The hash is unique to the image, and cannot be replicated with a modified image
 - The user hashes the JSON config in SHA256, and checks if it matches the provided image ID
 - If the hash matches, then the integrity is verified
- Users verify the origin of the image by verifying the signature
 - The user decrypts the hash with the public key provided by the publisher
 - If the public key decrypts the private key, then the identity of the publisher is validated



Docker Content Trust Demo

Let's sign our image, and publish it on
Docker Hub

Docker Security Practices

- If you pull a 3rd party image, make sure that the contents of the image are trustworthy
- Control access to the Docker daemon
 - By default, only root users can access it
 - Attackers with access to the daemon can start a container that shares the / directory of the host, and can read/write with no restriction
 - Avoid exposing the Docker API over HTTP (use HTTPS and certificates)
- As previously stated, enable and use Docker Content Trust to avoid receiving tampered images

Kubernetes Security

- Kubernetes introduces a new attack surface
- Proper Kubernetes security practices must be used on top of the previously mentioned Docker security practices
- Kubernetes has very powerful features which can be a double-edged blade

Controlling Access to the Kubernetes API

Goal: prevent unauthorized users from gaining access to the API

- Authentication – verify the identity of users
 - Client certificate (X509)
 - Bearer Token
 - Static Token File
 - In HTTP request header
 - Bootstrap Token
 - Service Account Token
 - Static Password File
 - OpenID Connect
- Authorization – tightly controlling user access to resources
 - Role-based access control (RBAC)
 - Attribute-based access control (ABAC)
 - Node



Q&A

