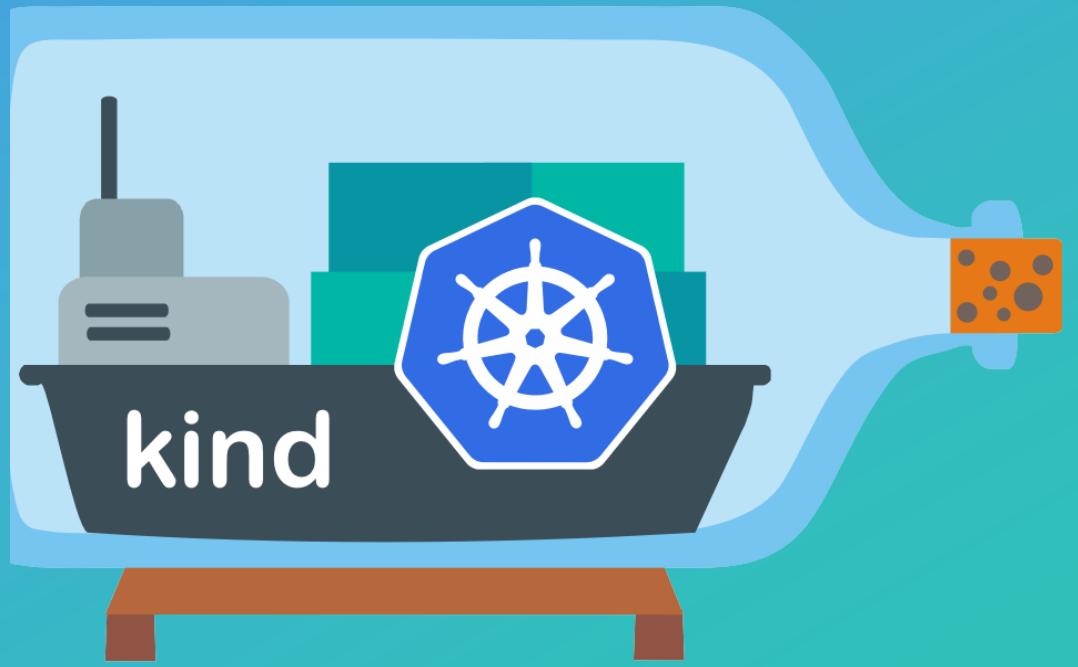
DOCKER AND KUBERNETES

Presented By:

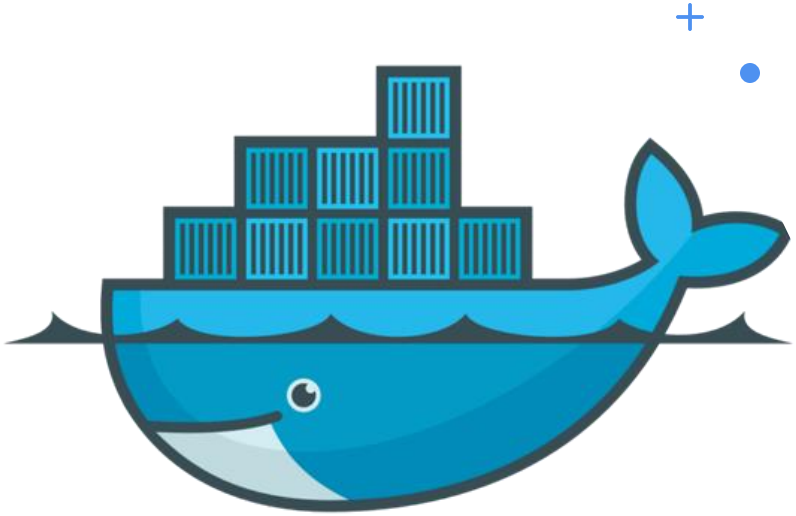
Group# 3

- Abdoon Nur
- Binod Rasaili
- Kirubel



What is Docker?

- Docker is a tool designed to make it easier to build, deploy and run applications by using container



docker

Why Docker?

Before and After Docker

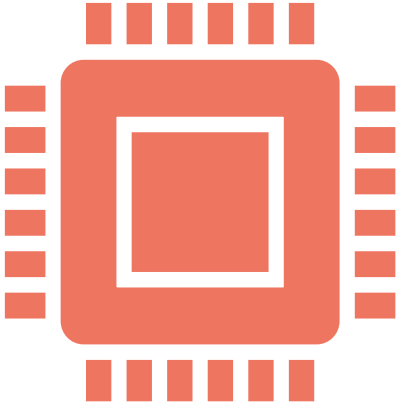
Dependency version mismatch from one development environment to another.

Library corrupted

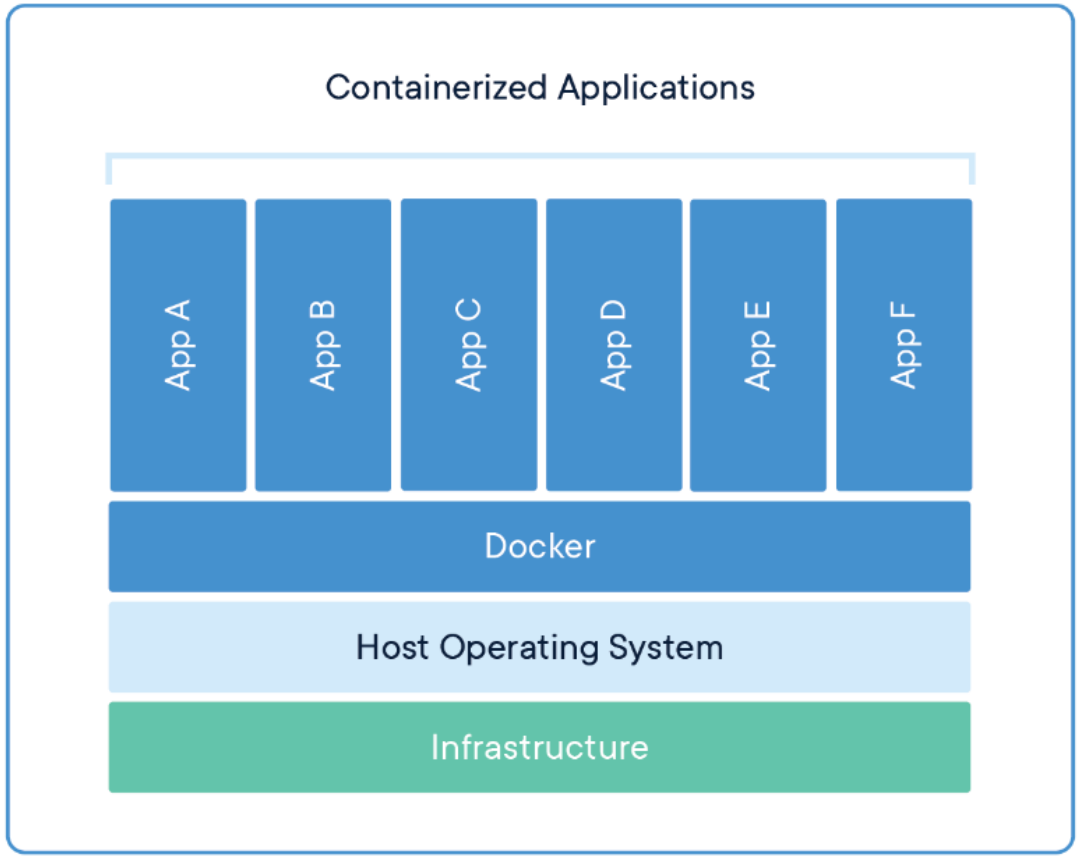
Software upgrade

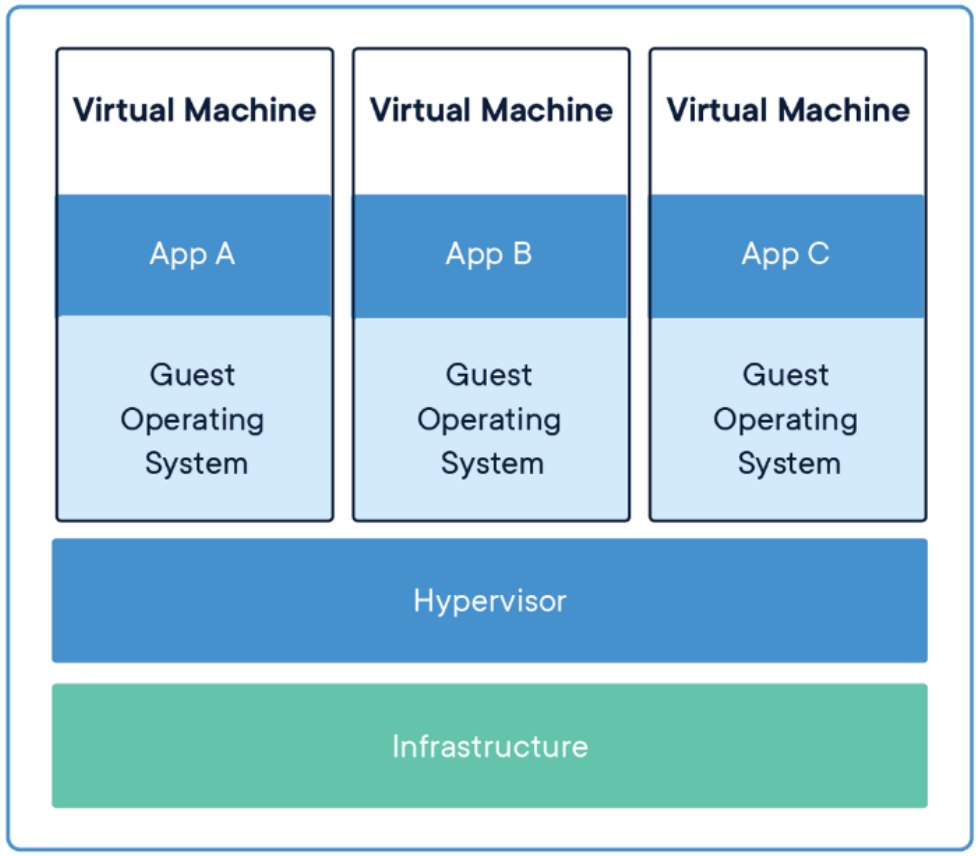
How Docker Resolve this issue

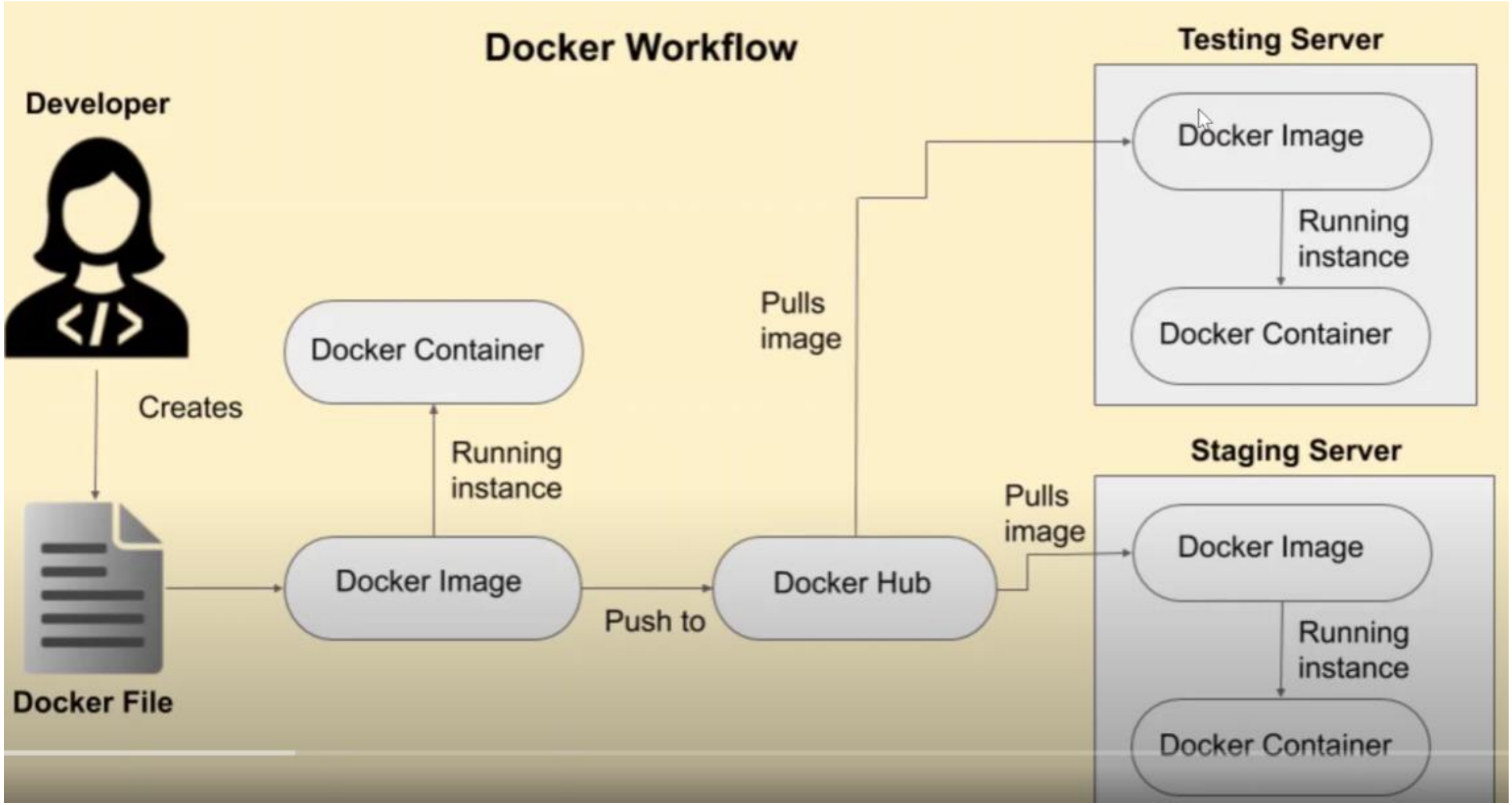
- Containers allow a developer to package up an application with all of the pards it need such as libraries and other dependencies and ship it all out as one package

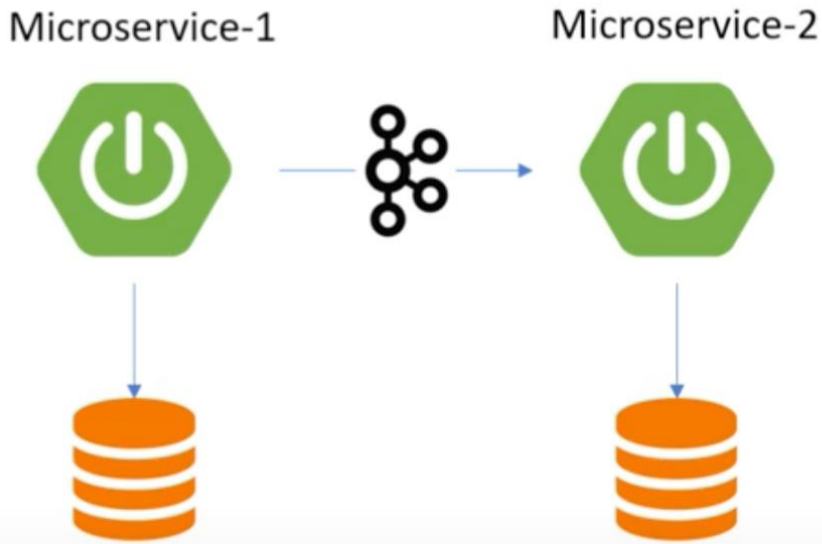


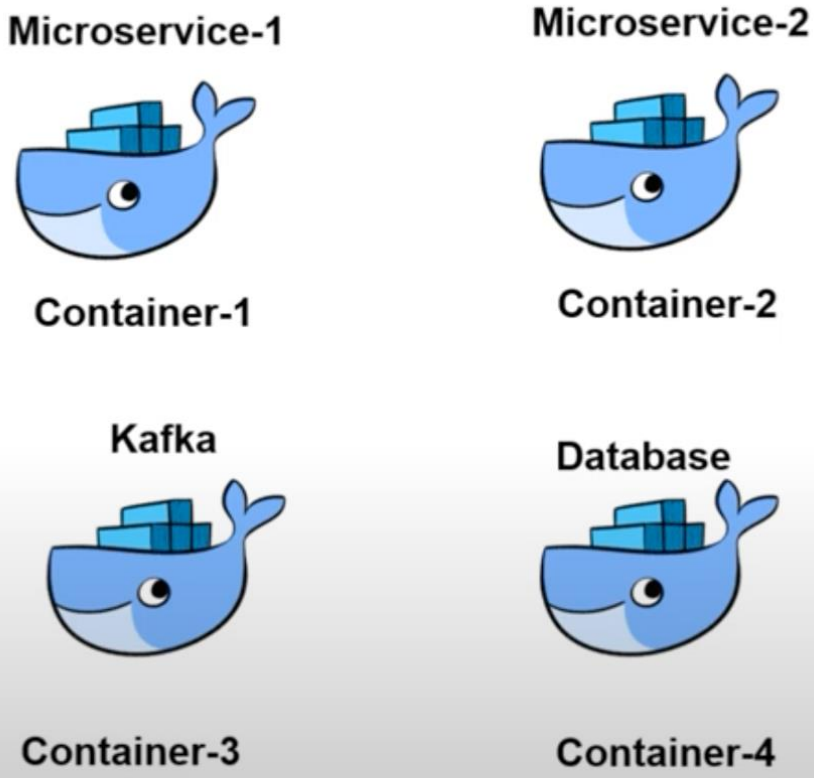
Container vs VMs






Docker Workflow
Developer
Creates
Docker File
Docker Container
Running instance
Docker Image
Push to
Docker Hub
Pulls image
Testing Server
Docker Image
Running instance
Docker Container
Pulls image
Staging Server
Docker Image
Running instance
Docker Container





Dockerize microservice -1

Dockerize microservice - 2

Pull Kafka image Pull Database image

Limitations of Docker

Docker operates on a single host, **limiting scalability.**

Requires **manual intervention for scaling**, complicating large deployments.

Setting up **complex networking** and service discovery is challenging.

No built-in support for **automated rollbacks** or self-healing.

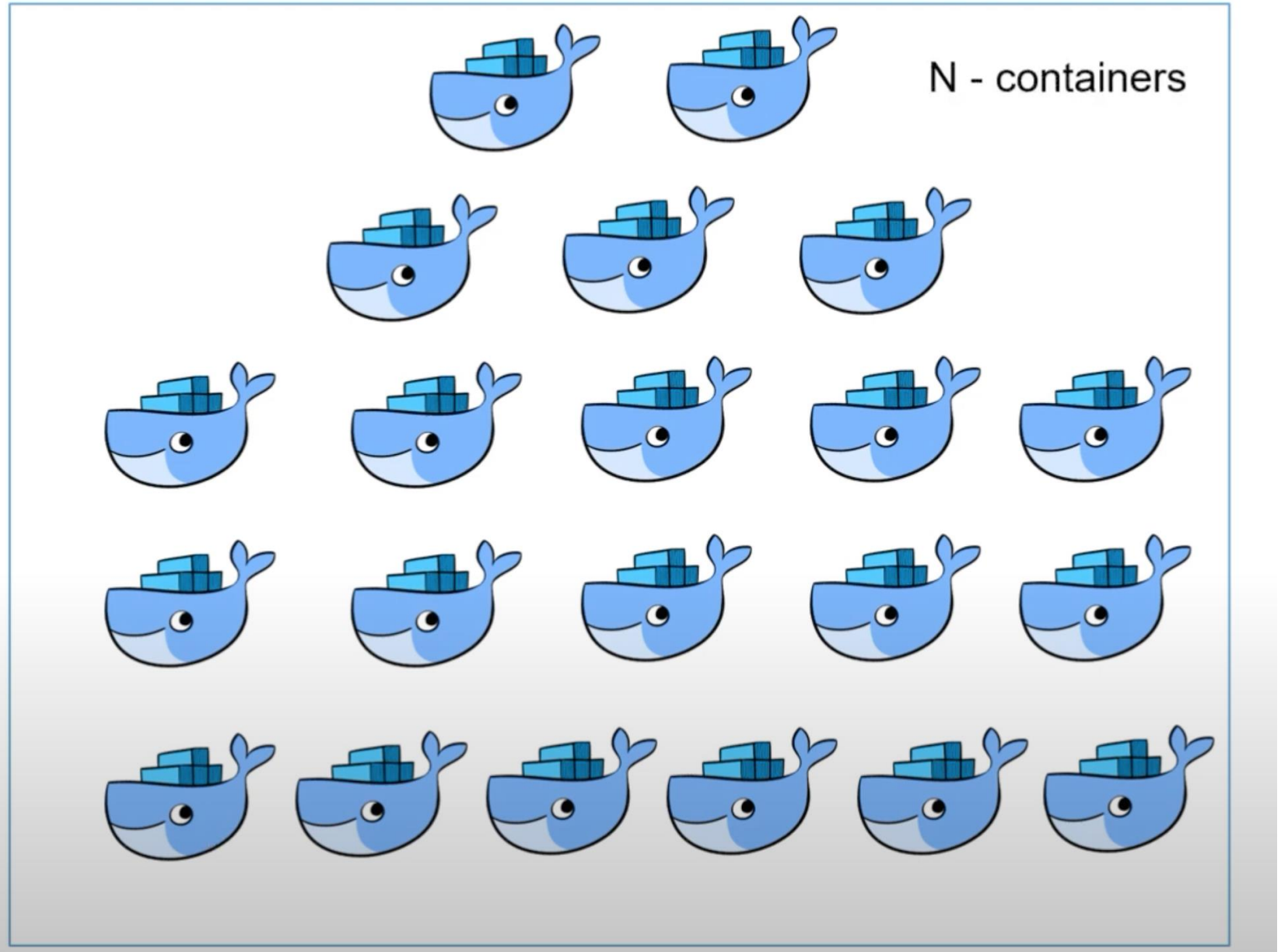
Managing multi-container applications is **complex** without **orchestration** tools.

Inadequate built-in **monitoring and logging** capabilities.

Lacks advanced **resource management** and scheduling.

Implementing rolling updates and **blue-green deployments** is error-prone.

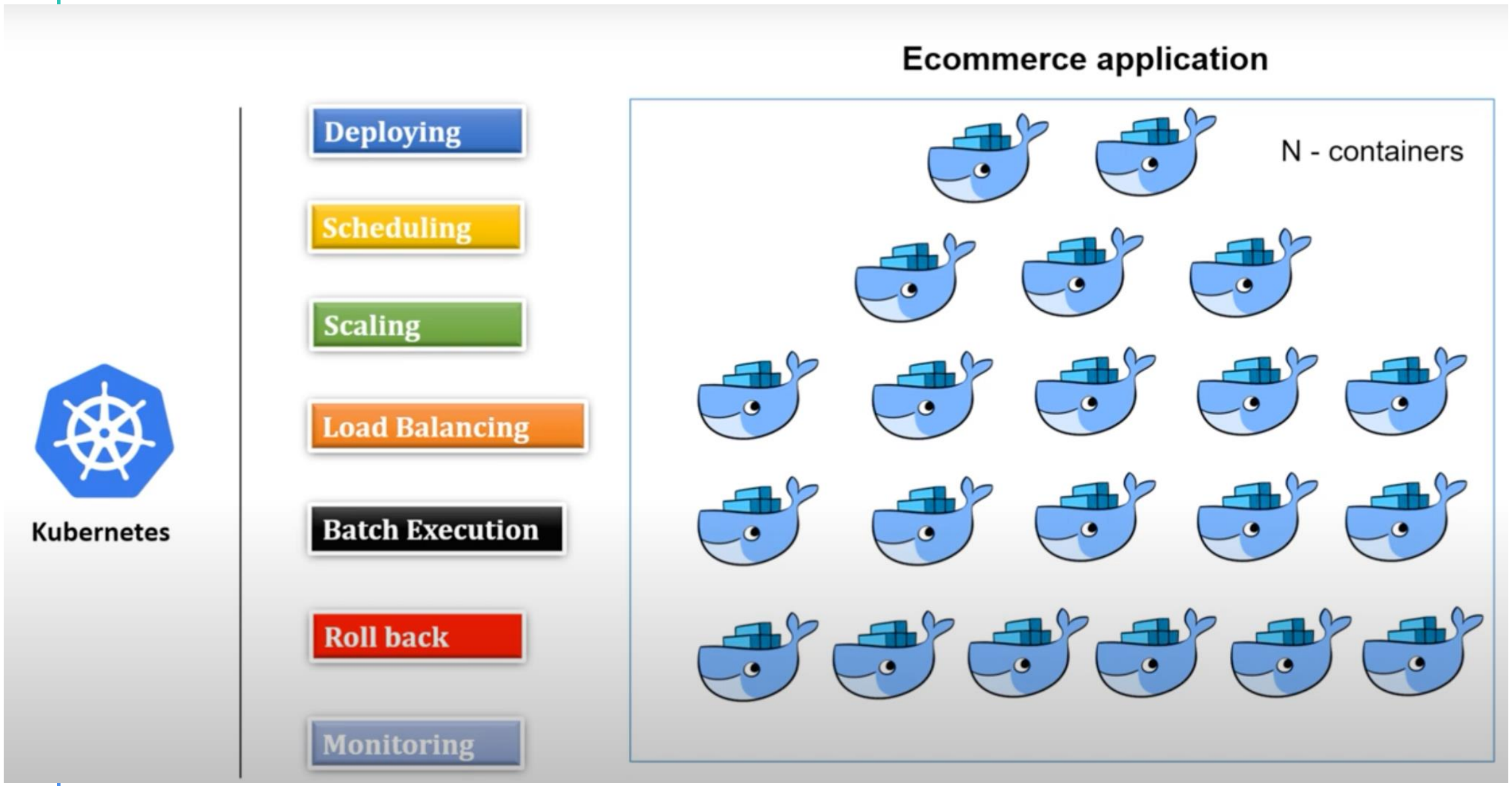
Ecommerce application



Kubernetes

Kubernetes is an open source container orchestration engine or container management tools, it automates deploying scaling and managing containerized applications.

- Kubernetes -> k8s
- Google
- Developed with GoLang


Ecommerce application
N - containers
Kubernetes
Deploying
Scheduling
Scaling
Load Balancing
Batch Execution
Roll back
Monitoring

Components of Kubernetes

POD

NODE

CLUSTER

REPLICATION CONTROLLER
REPLICA SET

SERVICE

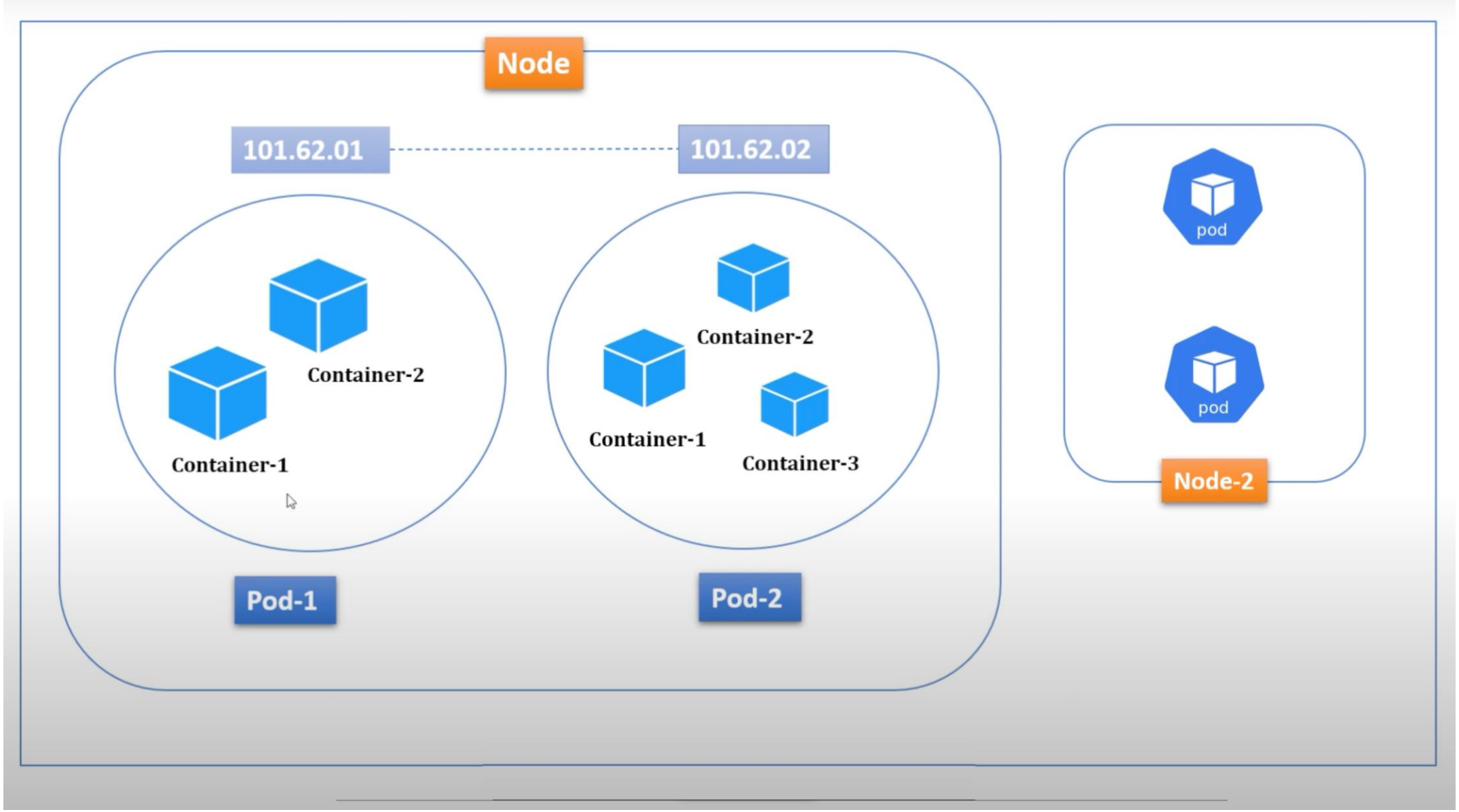
DEPLOYMENT

SECRETS

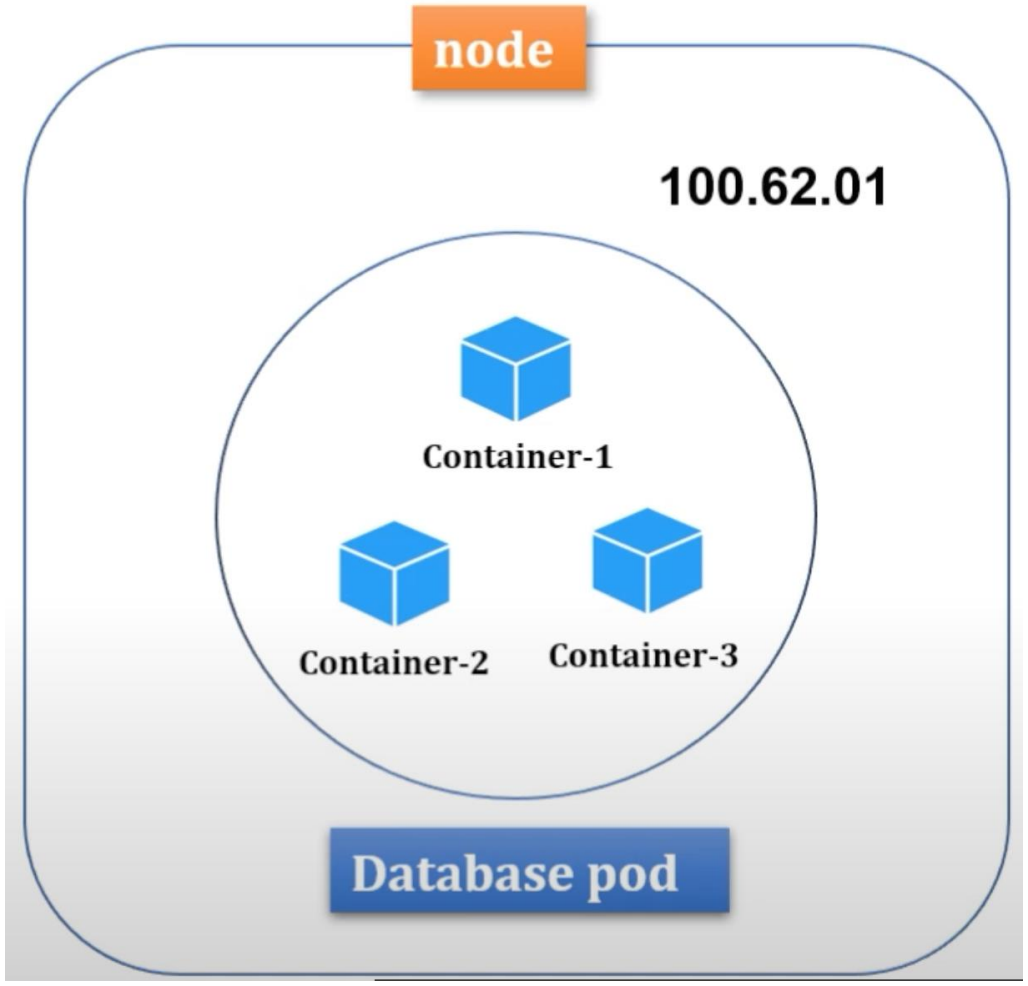
CONFIG MAP

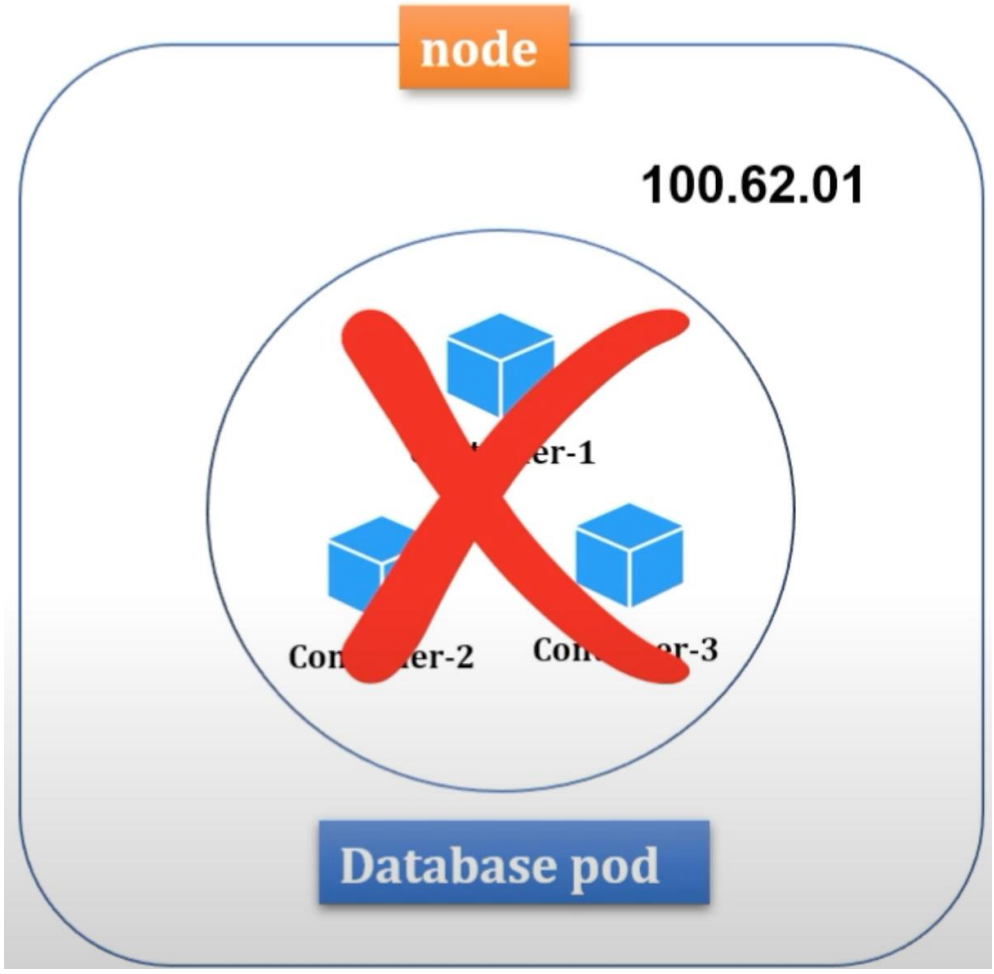
ETCD

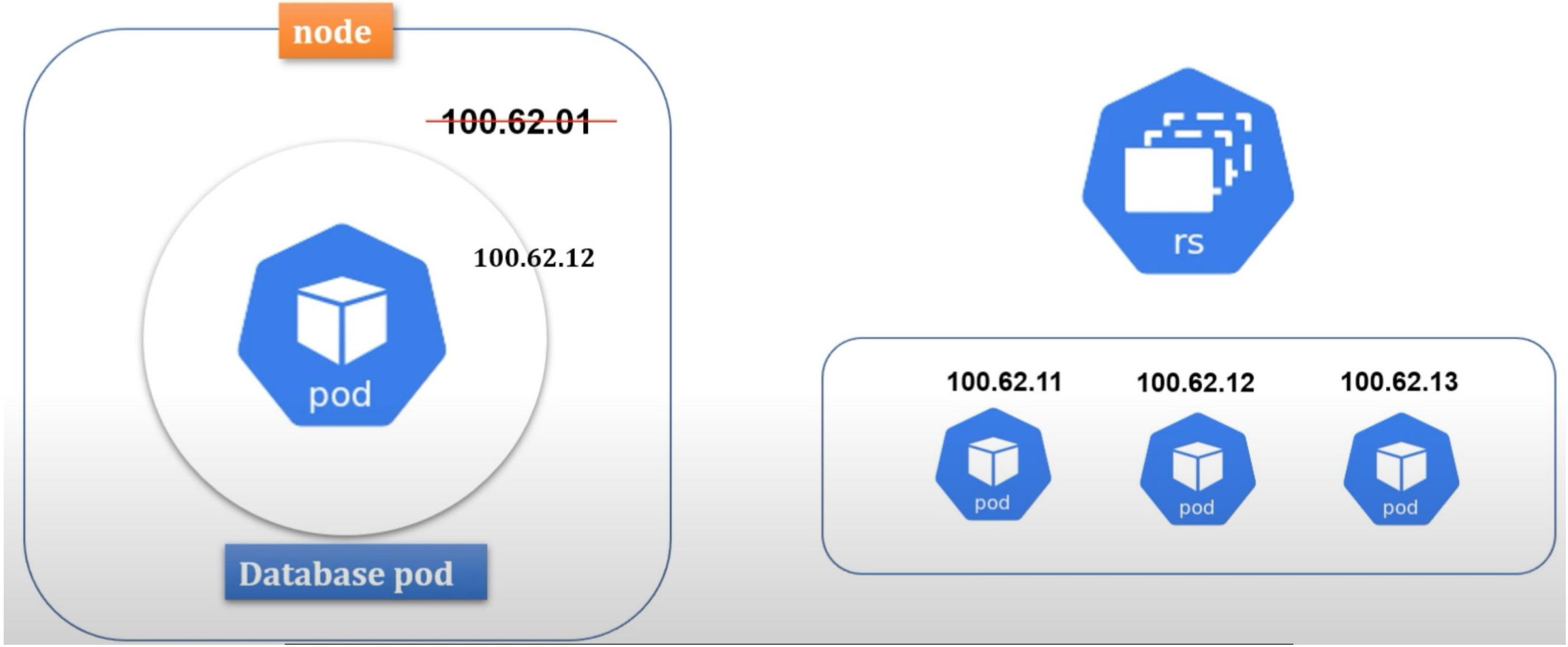
Pods and Nodes



Replication Process





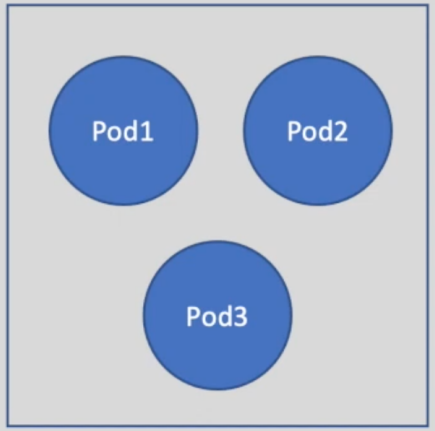

node
100.62.01
100.62.12
pod
Database pod
rs
100.62.11
100.62.12
100.62.13
pod
pod
pod

Kubernetes Auto Scaling

- **Horizontal Pod Autoscaler (HPA)**: Automatically adjusts the number of pods based on CPU utilization or other metrics, ensuring efficient resource use.
- **Vertical Pod Autoscaler (VPA)**: Dynamically adjusts the CPU and memory requests and limits for containers, optimizing resource allocation.
- **Cluster Autoscaler**: Automatically adjusts the number of nodes in the cluster based on the resource requirements of your workloads, maintaining balance and performance.

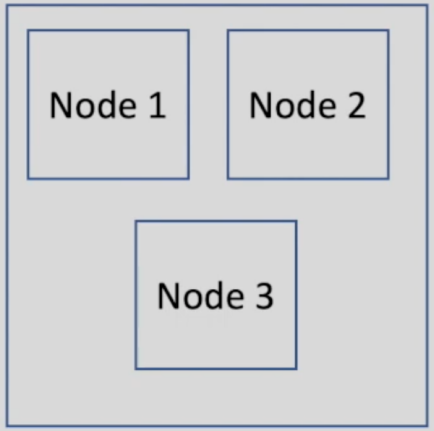
Types of Autoscalers:

HPA



VPA

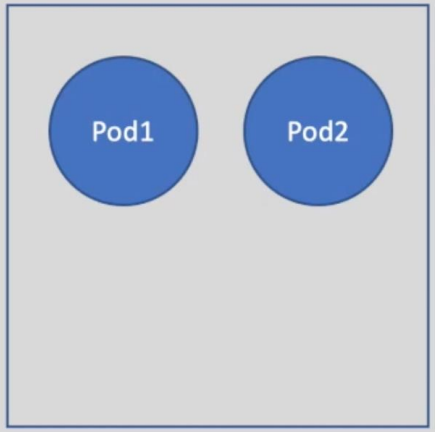
CA



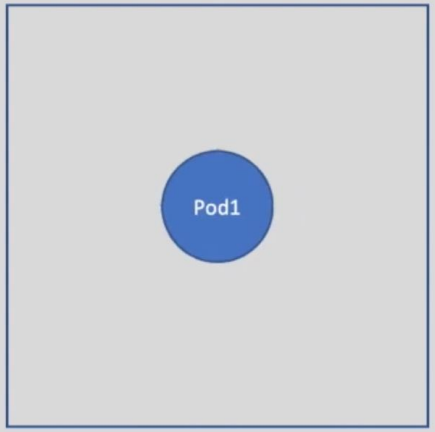
Scale Up

Types of Autoscalers:

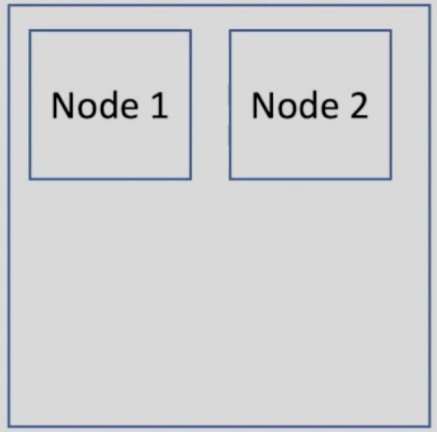
HPA



VPA



CA



Scale Down

What Problems Are Solved by Kubernetes and Docker together?

Docker:

- **Consistency**: Ensures applications run the same in development, testing, and production.
- **Isolation**: Prevents conflicts by packaging applications with their dependencies.
- **Portability**: Allows applications to run anywhere with Docker Engine.

Kubernetes:

- **Scalability**: Automatically scales applications up or down based on demand.
- **Self-Healing**: Restarts failed containers and replaces unhealthy ones.
- **Load Balancing**: Distributes traffic to ensure high availability and performance.
- **Automated Rollouts/Rollbacks**: Manages application updates with zero downtime.

Important Points of the Demo

Docker:

- Simplicity in creating and running containers.
- Ensures application consistency and isolation.
- Portability and Scalability
- Version Control and Collaboration
- Enables running multiple containers on the same host without significant overhead

Kubernetes:

- Automates deployment, scaling, and management of applications.
- Provides self-healing and load balancing for high availability.

Strengths and Weakness

Strengths

Docker:

- Easy to use and set up.
- Lightweight and efficient.
- Ensures consistent environments.

Kubernetes:

- Powerful orchestration and management.
- Scales applications seamlessly.
- High availability and resilience.

Weaknesses:

- **Docker**:
  - Not suitable for managing large-scale applications alone.
  - Limited orchestration capabilities.
- **Kubernetes**:
  - Steeper learning curve.
  - More complex to set up and manage.
  - Requires additional resources.

When to Use Kubernetes and Docker for Our Project

When to Use Docker:

- For development and testing environments.
- For simple applications needing isolation and consistency.
- When quick and lightweight deployment is required.

When to Use Kubernetes:

- For large-scale, distributed applications.
- When automated scaling and self-healing are necessary.
- For applications requiring high availability and complex orchestration.

Summary and Conclusion

- Docker and Kubernetes are powerful tools for modern application development and deployment.
- Docker ensures consistency and portability, while Kubernetes provides robust orchestration and management.
- Together, they simplify the process of building, deploying, and running applications, making them ideal for both simple and complex projects.