The Hoang

# Containerization, Orchestration and CI / CD

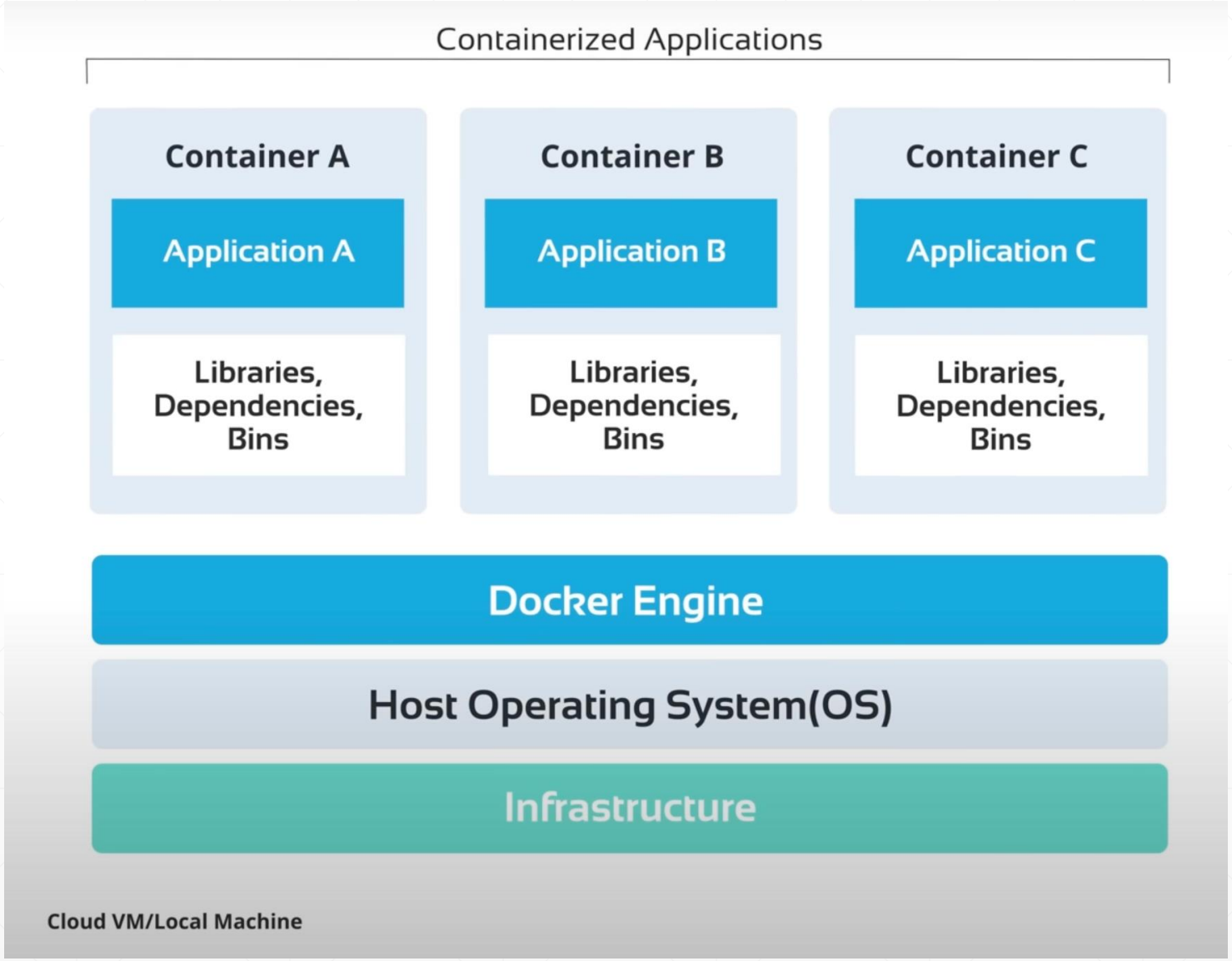
Understand and Implement Production-Grade Machine Learning Operations

# Containerization

Containerization is the process of packaging software code, dependencies, and configurations into a single unit called a container.

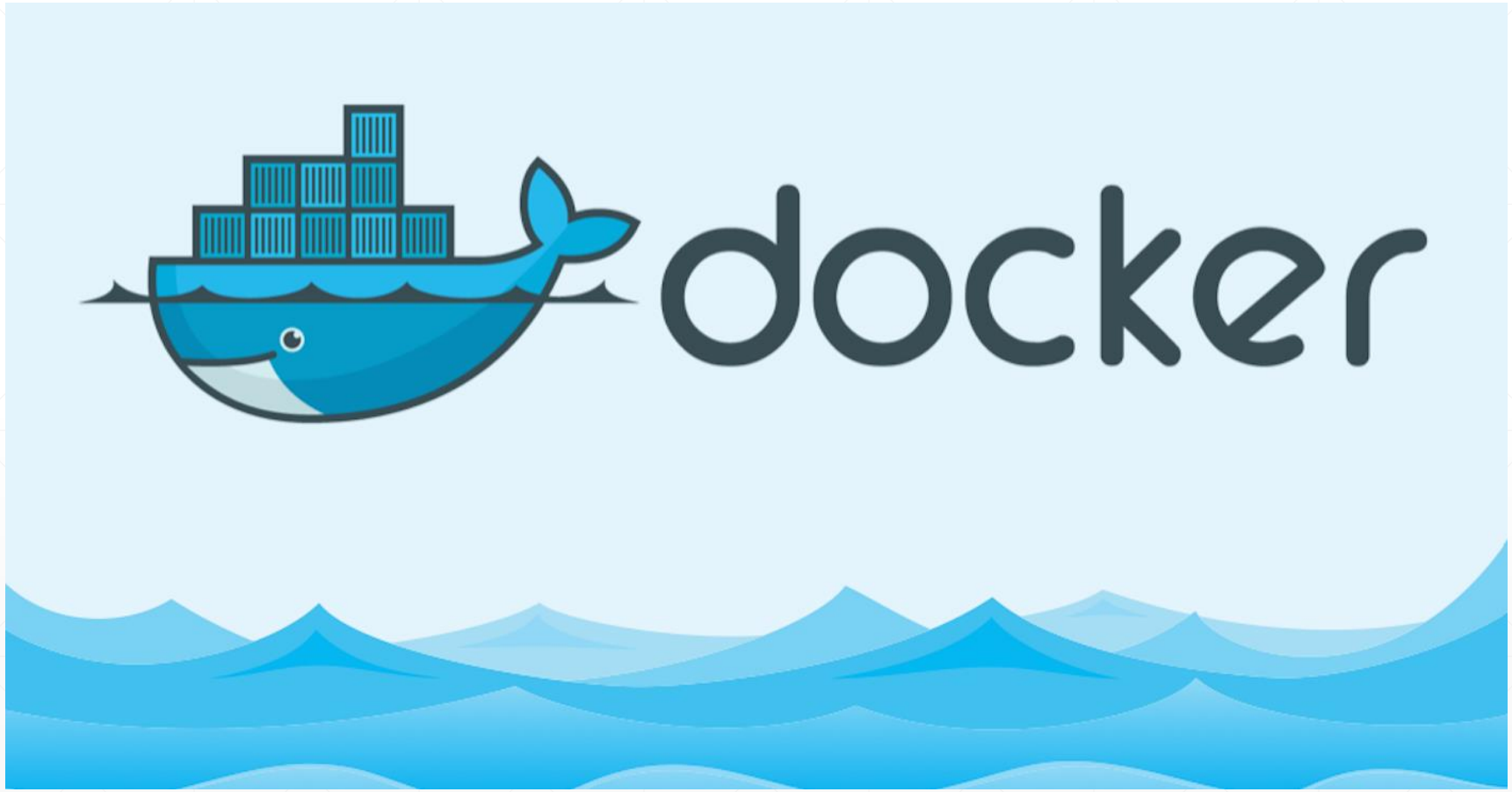
Containers are isolated environments that run the same regardless of where they're deployed (your laptop, a server, the cloud).

# Containerization

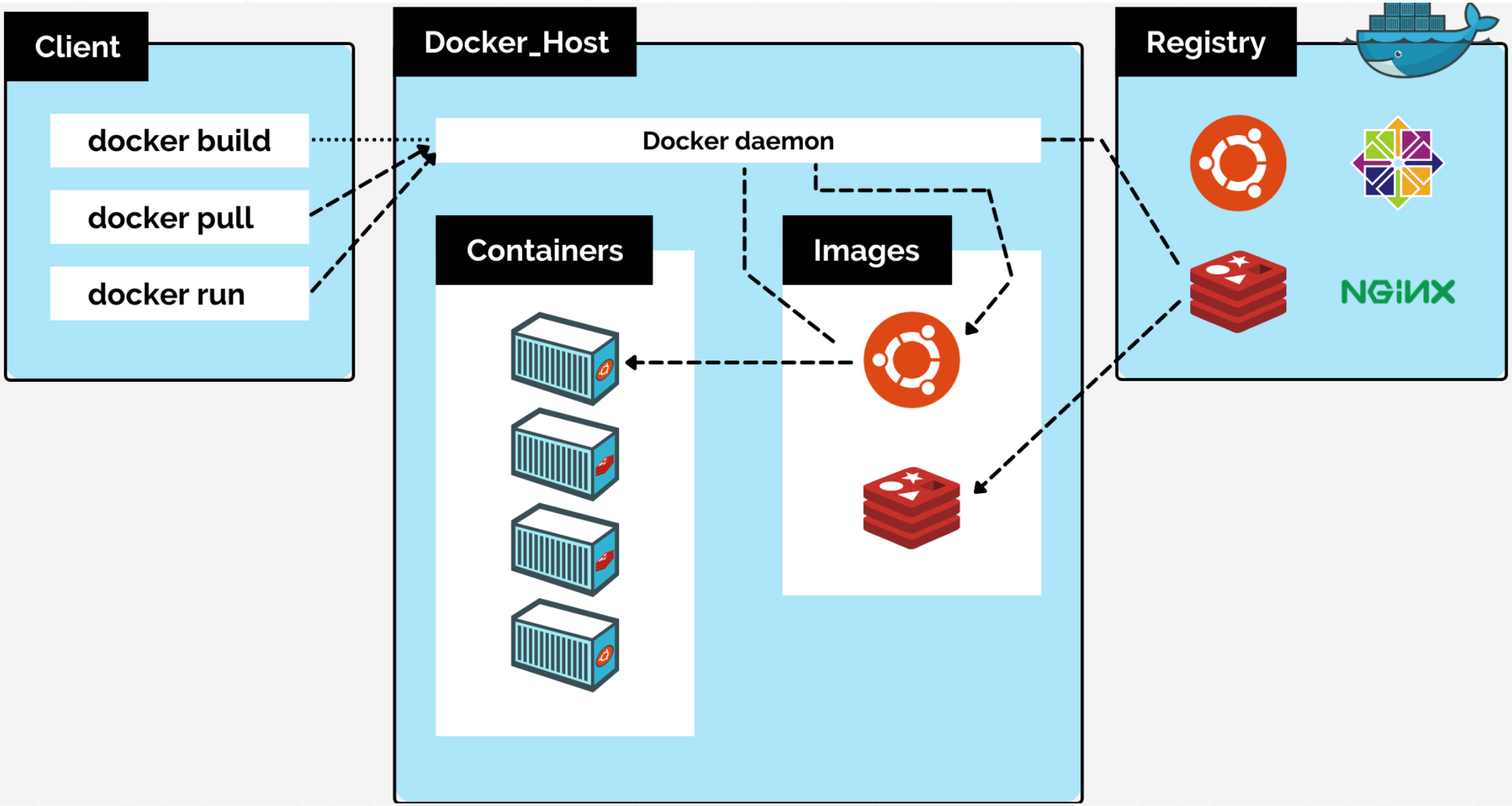


# Containerization

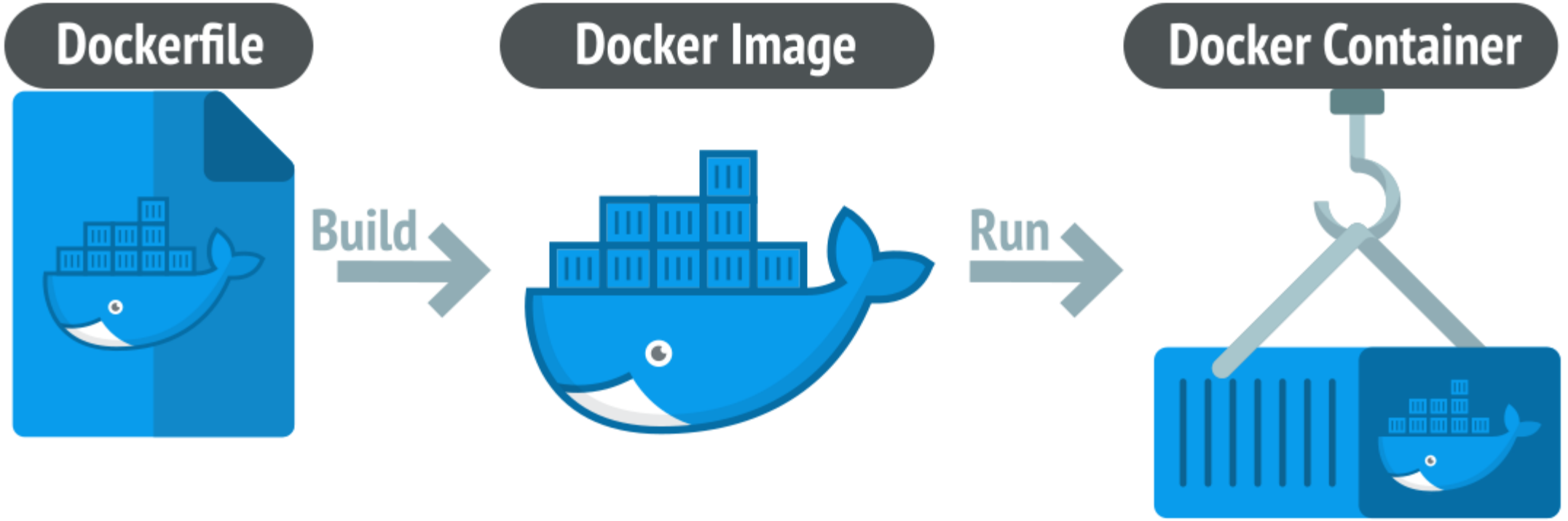
**Docker** is the most popular containerization platform.



# Docker – Core Components



# Docker – Key Concepts




**Dockerfile**: A script to build Docker images.
**Image**: A snapshot of a container (think: template).
**Container**: A running instance of an image.

# Docker – Key Concepts

**Example**: Simple Dockerfile for a Python ML app

dockerfile

Copy

```dockerfile
# Use an official Python runtime as a parent image
FROM python:3.10

# Set the working directory
WORKDIR /app

# Copy requirements and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of your application code
COPY . .

# Command to run your ML app
CMD ["python", "main.py"]
```

# Docker – Key Concepts

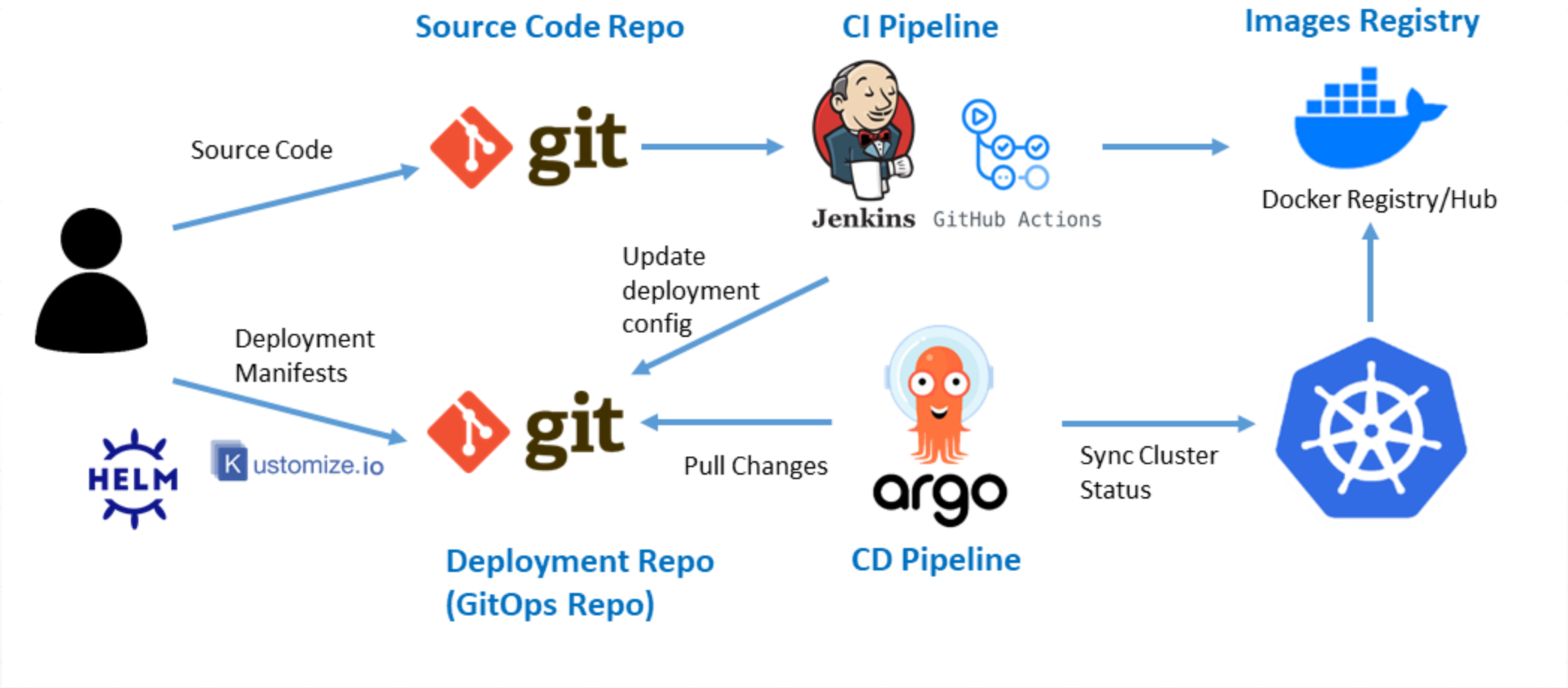
How to build and run

```bash
docker build -t my-ml-app .
docker run -p 5000:5000 my-ml-app
```


# Docker – Practice

- Create a simple FastAPI application
- Create a Dockerfile
- Build Docker image
- Run Docker container
- Login Docker Hub
- Push the Docker Image

# Continuous Integration & Continuous Deployment




## CI/CD Setup

- Setup CI/CD pipeline using Github Action that includes:
  - CI: Build, lint check, unit test
  - CD: Build and deploy image to Docker Hub

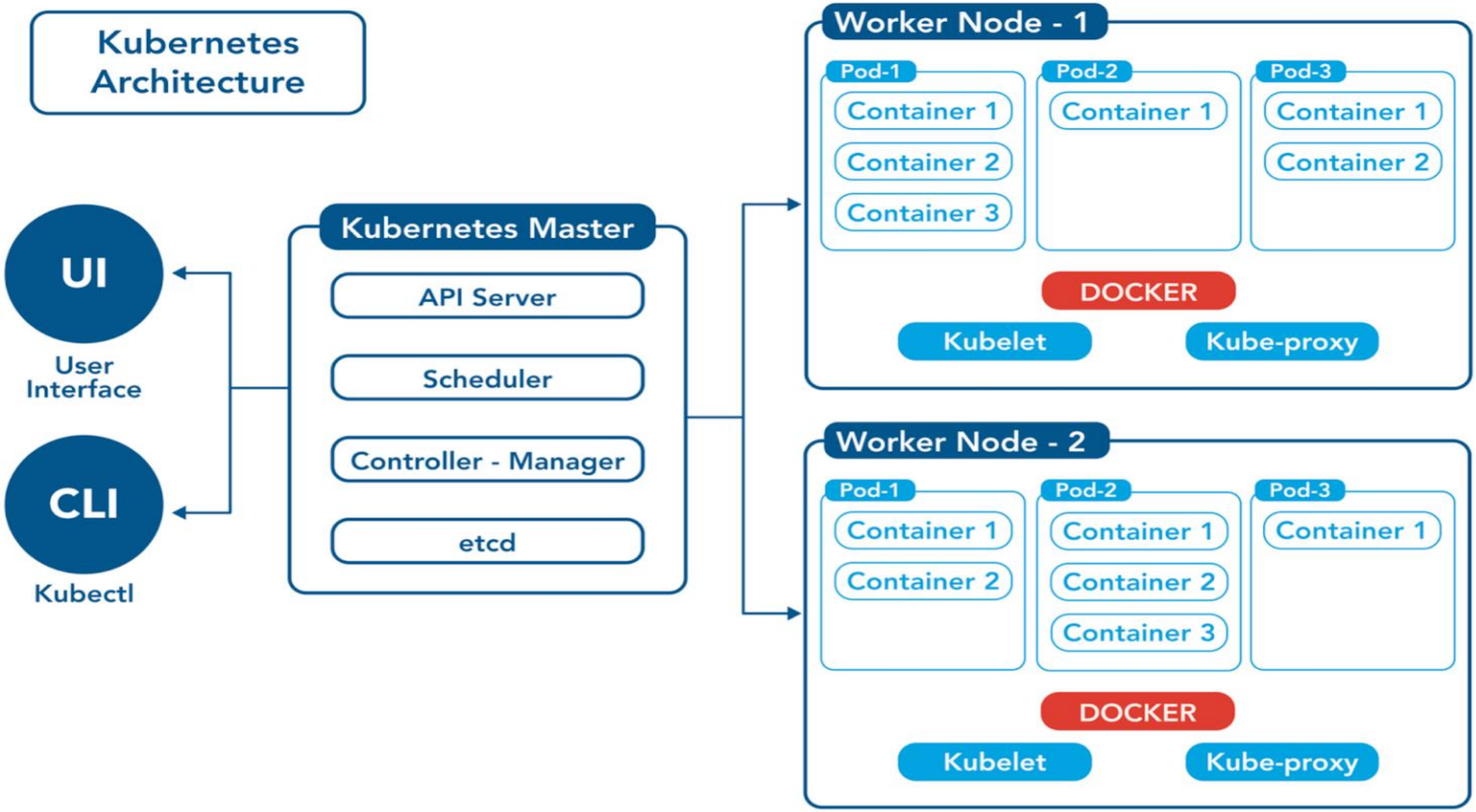
# Container Orchestration

Orchestration tools manage the lifecycle and scaling of containers.

They automate deployment, scaling, networking, and management of containerized apps.

Kubernetes is the leading orchestration platform.

# Kubernetes – Architecture



# Kubernetes – Architecture Explained

**Control Plane**: This manages the overall cluster state and makes global decisions such as scheduling and responding to cluster events. Key components include:

- **API Server**: The central management interface that exposes the Kubernetes API, serving as the gateway for all interactions with the cluster.
- **Scheduler**: Assigns pods (groups of containers) to worker nodes based on resource availability and policies.
- **Controller Manager**: Runs controller processes that monitor the cluster's desired state and take corrective actions to maintain it.
- **etcd**: A distributed key-value store that persistently stores all cluster data and configuration, ensuring consistency and availability

# Kubernetes – Architecture Explained

**Worker Nodes**: These are the machines (physical or virtual) where application containers run. Each node runs:

- **kubelet**: An agent that ensures containers are running as expected and communicates node status to the control plane.
- **kube-proxy**: Manages network routing and load balancing to enable communication between pods and services.
- **Container runtime**: Software responsible for running containers (e.g., Docker, containerd)

# Kubernetes – Key Concepts

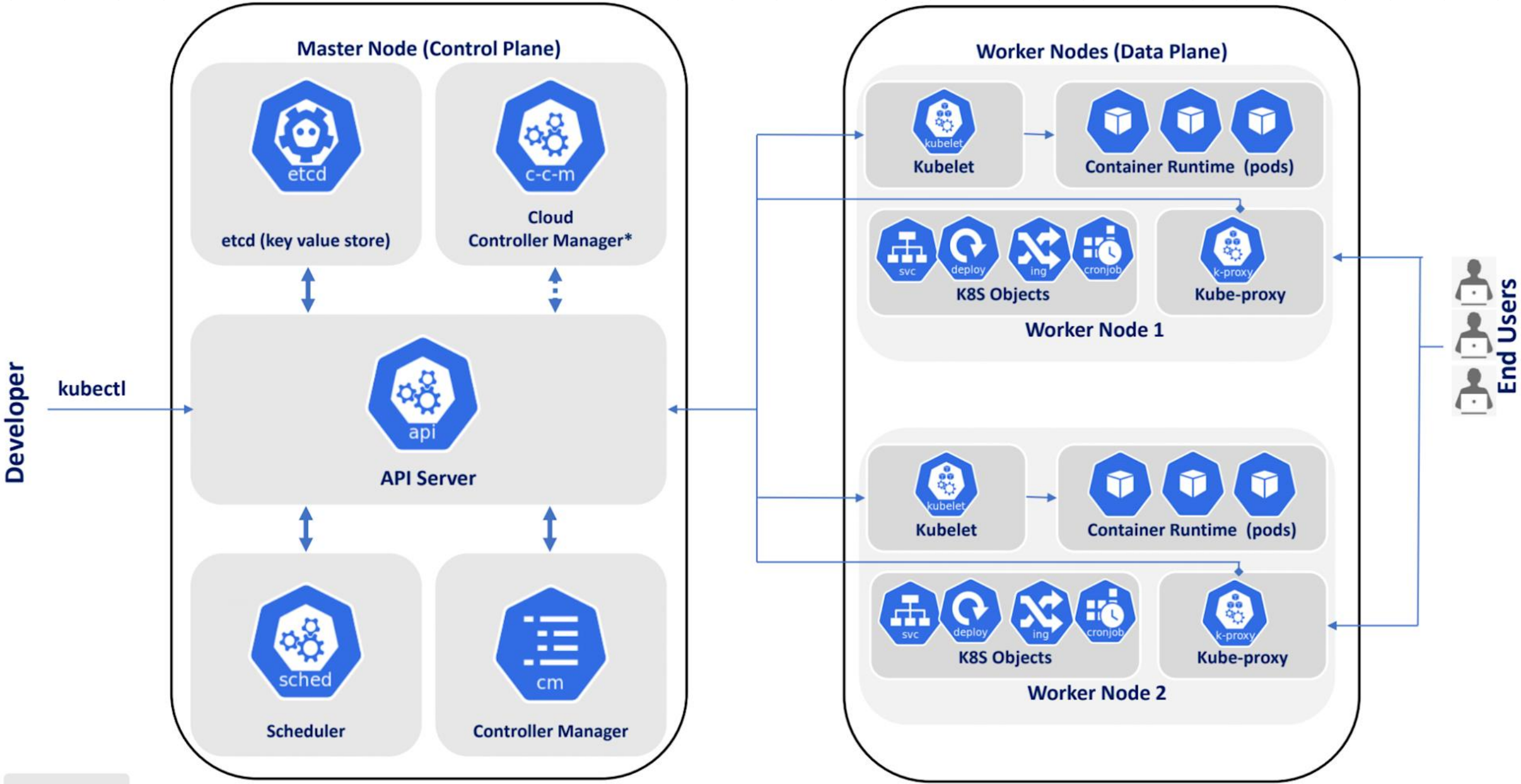
**Pods**: The smallest deployable units in Kubernetes, pods are groups of one or more containers that share storage, network, and specifications on how to run the containers. Containers in a pod share the same network namespace, allowing them to communicate via localhost.

**Clusters**: A set of nodes (worker machines) managed by the control plane. Clusters can span physical or virtual machines and can be deployed on-premises or in cloud environments.

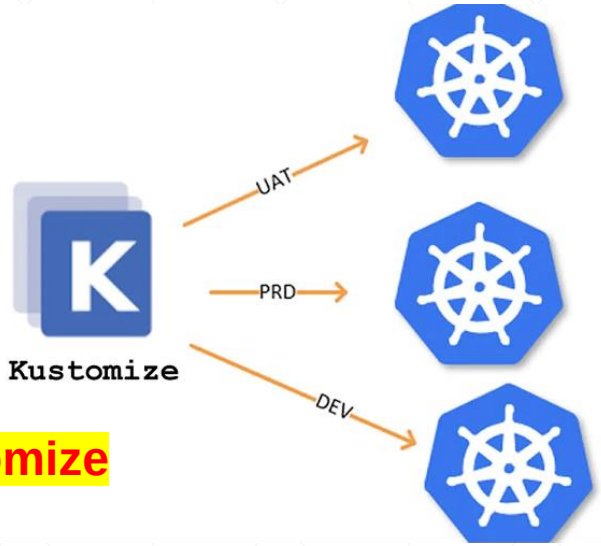
**Namespaces**: Logical partitions within a cluster that allow for resource isolation and organization. Namespaces enable multiple teams or projects to coexist in the same cluster without resource conflicts.

**Services**: Abstractions that define a logical set of pods and a policy by which to access them. Services enable stable networking endpoints and load balancing for pods that may be ephemeral.

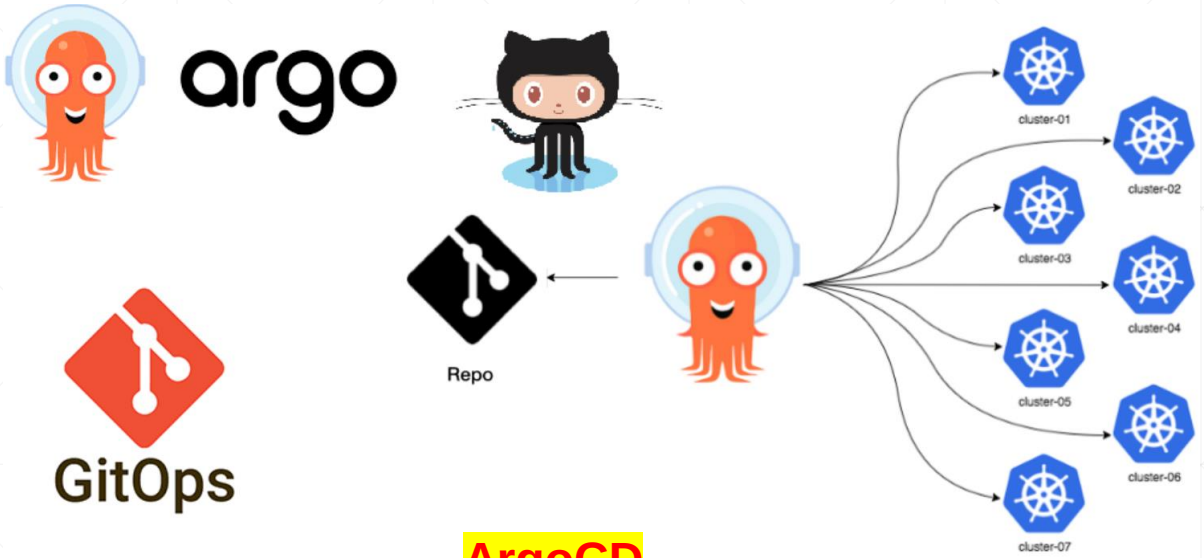
# Kubernetes – Architecture



# Over To You – Research

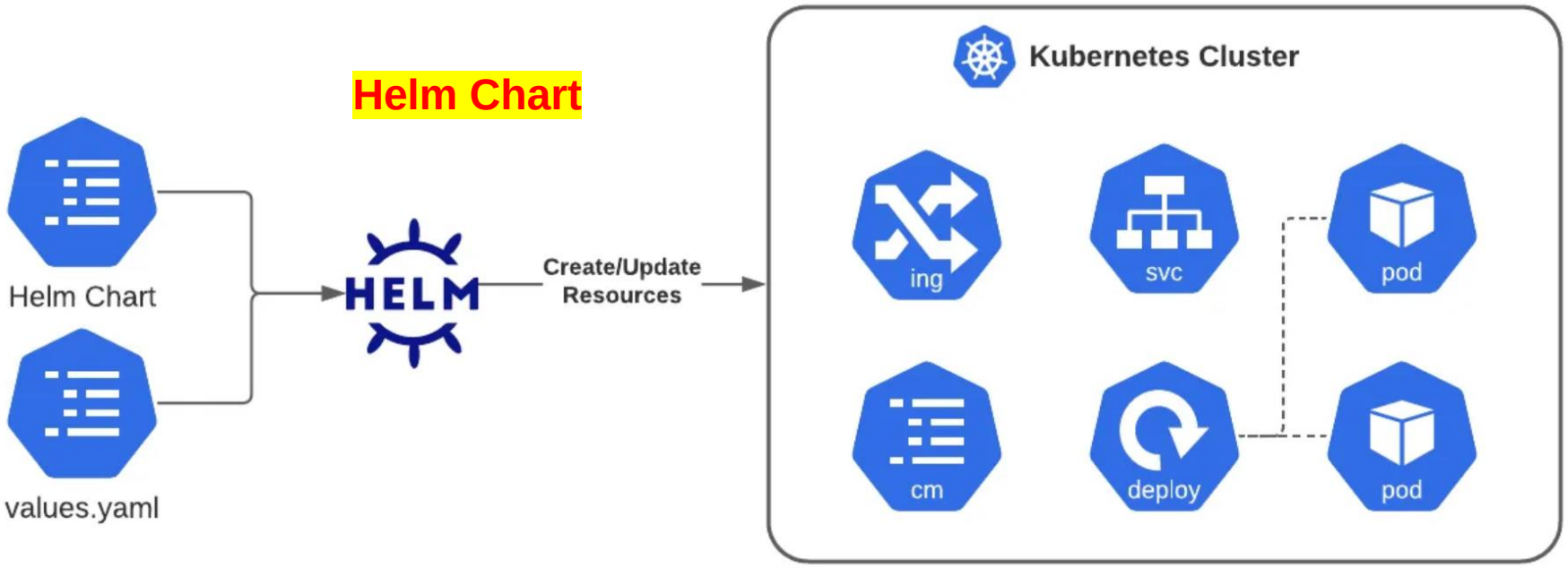



Kustomize




ArgoCD

Helm Chart



# Kustomize – What is Kustomize?

Kustomize is a Kubernetes-native configuration management tool that helps you customize Kubernetes YAML manifests without using templates. It works by layering configurations using a concept of bases and overlays, allowing you to maintain a single source of truth while adapting deployments for different environments like development, testing, and production.

# Kustomize – Key Concepts?

**Base**: A set of common Kubernetes resource files (Deployments, Services, ConfigMaps) that represent the core application configuration.

**Overlay**: Environment-specific customizations that modify or extend the base configuration (e.g., changing replica counts, resource limits).

**kustomization.yaml**: The main configuration file where you declare resources, patches, and transformations.

**Patches**: Changes applied to base resources using strategic merge or JSON patches.

**Transformers**: Add or modify metadata like labels or namespaces across all resources.

**Secret and ConfigMap Generators**: Automatically generate Kubernetes Secrets and ConfigMaps from literals or files.

# Kustomize – Why use Kustomize?

- Avoids duplication of YAML files for different environments.
- No templating language, so YAML stays clean and readable.
- Integrated with kubectl (e.g., kubectl apply -k <dir>).
- Supports layering and modularity for complex deployments.
- Helps prevent configuration drift by managing overlays declaratively.

**Example use case**:

- You have a base deployment manifest for your app. For production, you want 5 replicas and a rolling update strategy, but for development only 1 replica without rolling updates. Using Kustomize, you keep the base manifest and create overlays for dev and prod that override just the necessary fields

# Helm Chart - What is Helm?

- Helm is a package manager for Kubernetes that uses templated YAML files to define, install, and upgrade Kubernetes applications. Helm packages are called charts. A Helm chart contains all the resource definitions needed to run an application, configurable via parameters[no direct search result but common knowledge].

# Helm Chart - Key Concepts?

- **Chart**: A collection of files that describe a related set of Kubernetes resources.
- **Templates**: YAML files with Go templating syntax to allow dynamic content based on input values.
- **Values.yaml**: A file where you specify configuration values that customize the templates.
- **Releases**: Instances of a chart deployed in a Kubernetes cluster.

# Helm Chart - Why is Helm?

- Simplifies deployment of complex applications with many Kubernetes resources.
- Supports parameterization and reuse via templates.
- Easy upgrades and rollbacks of applications.
- Large ecosystem of pre-built charts for common software.

**Difference from Kustomize:**

- Helm uses templating, which can be powerful but adds complexity.
- Kustomize is template-free and focuses on layering and patching YAML.
- Helm is more like a package manager, while Kustomize is a configuration customization tool.

# ArgoCD – What is ArgoCD?

- ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It continuously monitors your Git repositories containing Kubernetes manifests (or Helm charts, Kustomize directories) and automatically syncs the desired state to your Kubernetes cluster.

# ArgoCD – Key Features?

- **GitOps-based**: Uses Git as the single source of truth for Kubernetes manifests.
- **Declarative**: Desired state is defined in Git; ArgoCD applies and maintains it on the cluster.
- **Supports multiple config formats**: Plain YAML, Kustomize, Helm charts, Jsonnet.
- **Automated sync**: Automatically deploy changes when manifests in Git are updated.
- **Visual dashboard**: Web UI to monitor application status, sync history, and health.
- **Rollbacks and manual sync**: Supports manual interventions and rollbacks.

# ArgoCD – How ArgoCD fits in CI/CD?

- Your CI pipeline (e.g., GitHub Actions) builds and pushes Docker images and updates manifests in Git.
- ArgoCD detects manifest changes and deploys them to Kubernetes automatically.
- This separation of concerns improves reliability and auditability.