# Kubernetes

## CLD

7 - Container Cluster

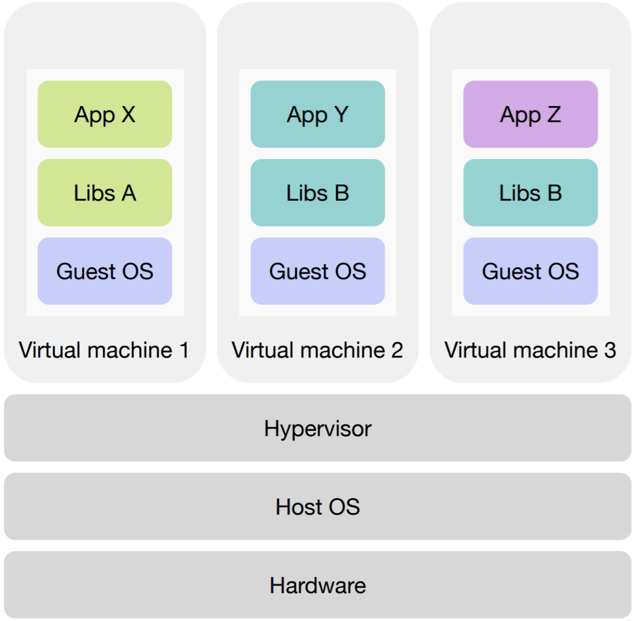
## Document summary

## Table of content

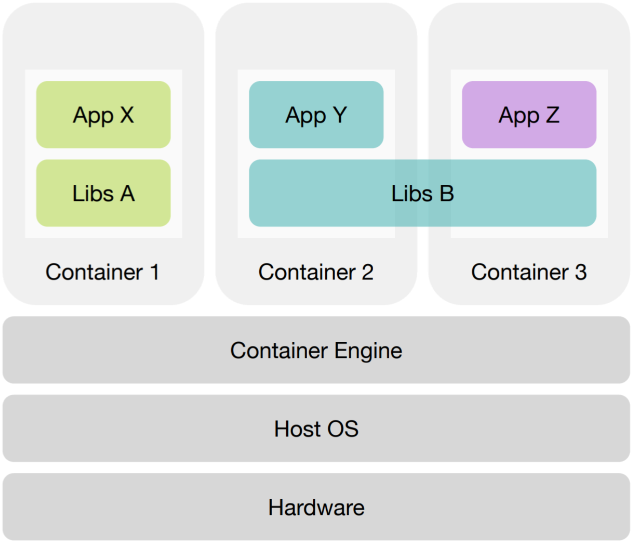
# 1. Software Containers

## 1.1. Containers vs Virtual Machines

- Containers provide a lightweight alternative to virtual machines by sharing the host operating system's kernel while maintaining isolated user spaces.
- Containers are more efficient in terms of resource utilization compared to traditional virtual machines.



Three VMs running on a single host



Three containers running on a single host

## 1.2. Building and Uploading Container Images

- The process involves creating a container image, usually with Docker, and uploading it to a container registry.
- Popular registries include Docker Hub, GitHub Container Registry, Amazon Elastic Container Registry, Azure Container Registry, and Google Artifact Registry.

# 2. Container Cluster Management

## 2.1. Introduction

- Managing container clusters is essential for deploying applications across multiple hosts to ensure robustness and service continuity.
- Key needs include monitoring container health, optimal placement of containers, and handling failures effectively.

## 2.2. Container Scheduling

- Scheduling determines the placement of application containers on cluster nodes based on resource requirements and constraints like affinity and anti-affinity.
- Goals are to increase cluster utilization while meeting application requirements.

# 3. YAML (Yet Another Markup Language)

- The operator can create K8s objects with the command line or he can describe the objects in manifest files.
  - `kubectl` create -f file.yaml
  - File format is JSON, which can also be written as YAML

## 3.1. Structure

- Only two basic data structures: arrays and dictionaries, which can be nested
- YAML is a superset of JSON
- Easier for humans to read and write than JSON
- Indentation is significant
- Specification at http://yaml.org/

## 3.2. YAML Example

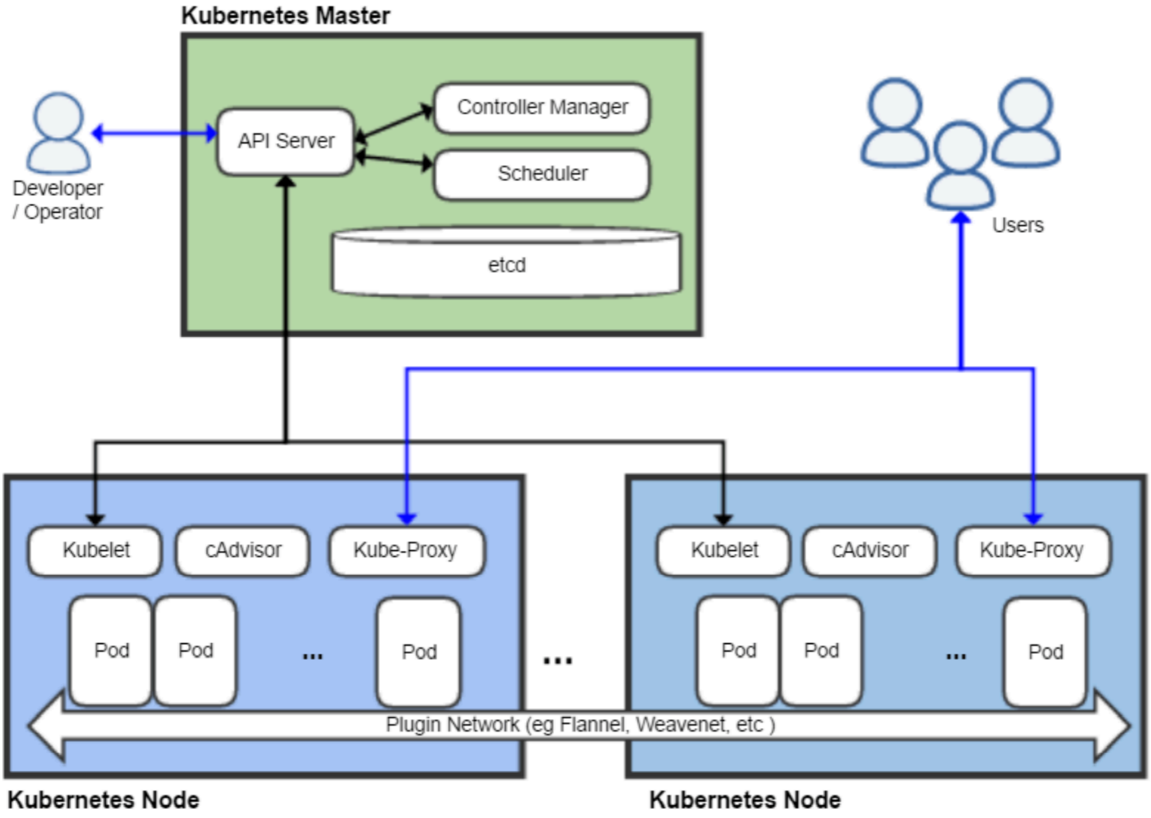
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    component: redis
    app: todo
spec:
  containers:
  - name: redis
  image: redis
  ports:
  - containerPort: 6379
  resources:
    limits:
      cpu: 100m
  args:
  - redis-server
  - --requirepass ccp2
  - --appendonly yes
```

# 4. Kubernetes

## 4.1. Introduction

- Kubernetes is an open-source platform for automating the deployment, scaling, and management of containerized applications.
- Originally developed by Google, it is now maintained by the Cloud Native Computing Foundation (**CNCF**).

## 4.2. Anatomy of a Cluster



### 4.2.1. Master Node Components

- **etcd**: A key/value store for cluster configuration data.
- **API Server**: Serves the Kubernetes API.
- **Scheduler**: Decides the nodes on which pods should run.
- **Controller Manager**: Runs core controllers like the Replication Controller.

### 4.2.2. Worker Node Components

- **Kubelet**: Manages the state of containers on a node.
- **Kube-proxy**: Handles network routing and load balancing.
- **cAdvisor**: Monitors resource usage and performance.
- **Overlay Network**: Connects containers across nodes.

## 4.3. Main Concepts

- **Cluster**: A set of machines (nodes) where pods are deployed and managed.
- **Pod**: The smallest deployable unit, consisting of one or more containers.
- **Controller**: Manages the state of the cluster.
- **Service**: Defines a set of pods and facilitates service discovery and load balancing.
- **Label**: Key-value pairs attached to objects for management and selection.

## 4.4. Common Concepts

- Kubernetes objects can be created and managed using YAML or JSON files.
- YAML is a human-readable format used to describe Kubernetes objects in configuration files.

## 4.5. Deploying an Application: IaaS vs Kubernetes

- Traditional IaaS involves manual steps like launching VMs, configuring them, and setting up load balancers.
- Kubernetes simplifies this process with container images and manifests, allowing automated deployment and scaling.

## 4.6. Kubernetes YAML Example

- Every Kubernetes object description begins with two fields:
  - **kind**: a string that identifies the schema this object should have
  - **apiVersion**: a string that identifies the version of the schema the object should have
- Every object has two basic structures: Object Metadata and Specification (or Spec).
- The Object Metadata structure is the same for all objects in the system
  - **name**: uniquely identifies this object within the current namespace
  - **labels**: a map of string keys and values that can be used to organize and categorize objects
- **Spec** is used to describe the desired state of the object

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    component: redis
    app: todo
spec:
  containers:
  - name: redis
    image: redis
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: 100m
    args:
    - redis-server
    - --requirepass ccp2
    - --appendonly yes
```