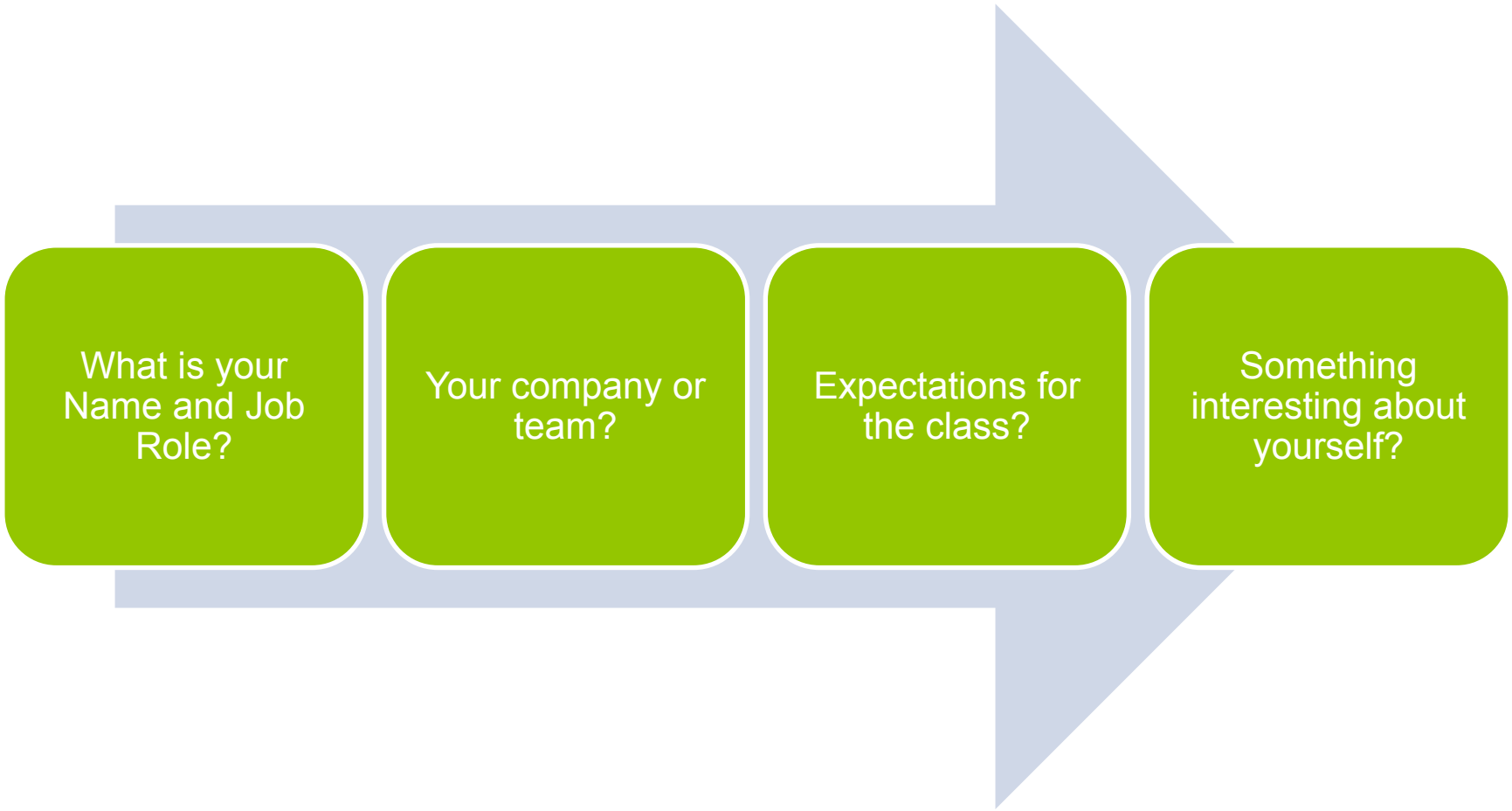
Welcome!

- Logistics (breaks, facilities, lunch, etc.)
- Rules of Engagement
- Introductions
- Let's Get Started!

Who is your instructor?

A little about me…

Introductions



Kubernetes Installation and Architecture

Table of Content

Containers

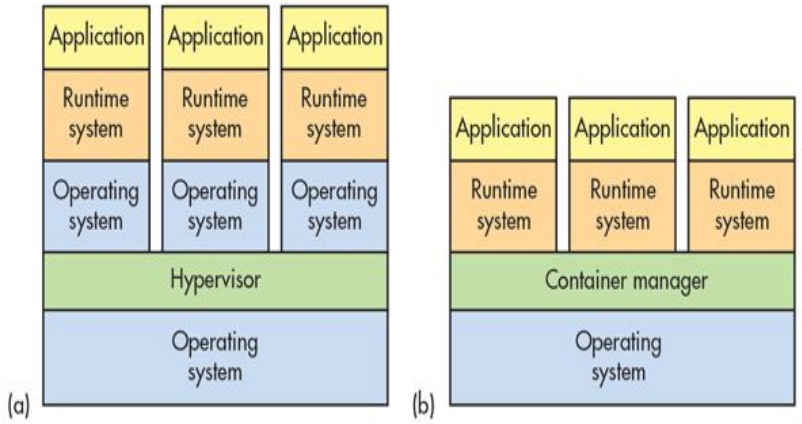
- Containers are becoming the standard unit of deployment
- Each container image has
  - Code
  - Binaries
  - Configuration
  - Libraries
  - Frameworks
  - Runtime
- Developers and Operators love containers

Containers

- Docker has solved the problem of packaging, deploying and running containerized applications
- Docker is great for managing a few containers running on a fewer machines
- Production applications deal with dozens of containers running on hundreds of machines

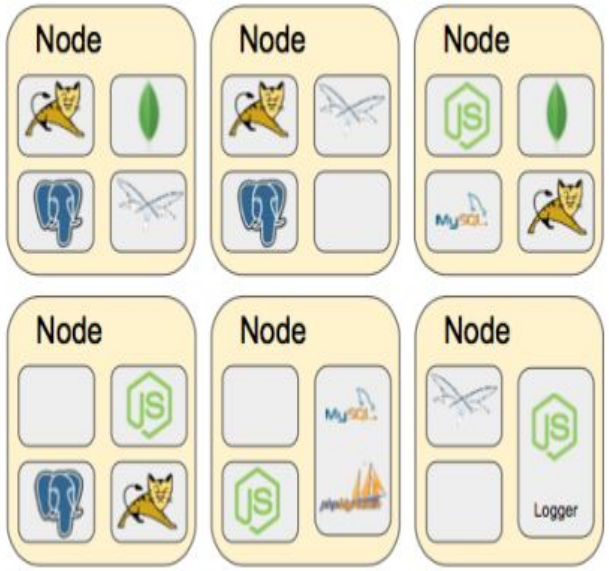
Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify DevOps practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption



2.1 Why need container orchestration

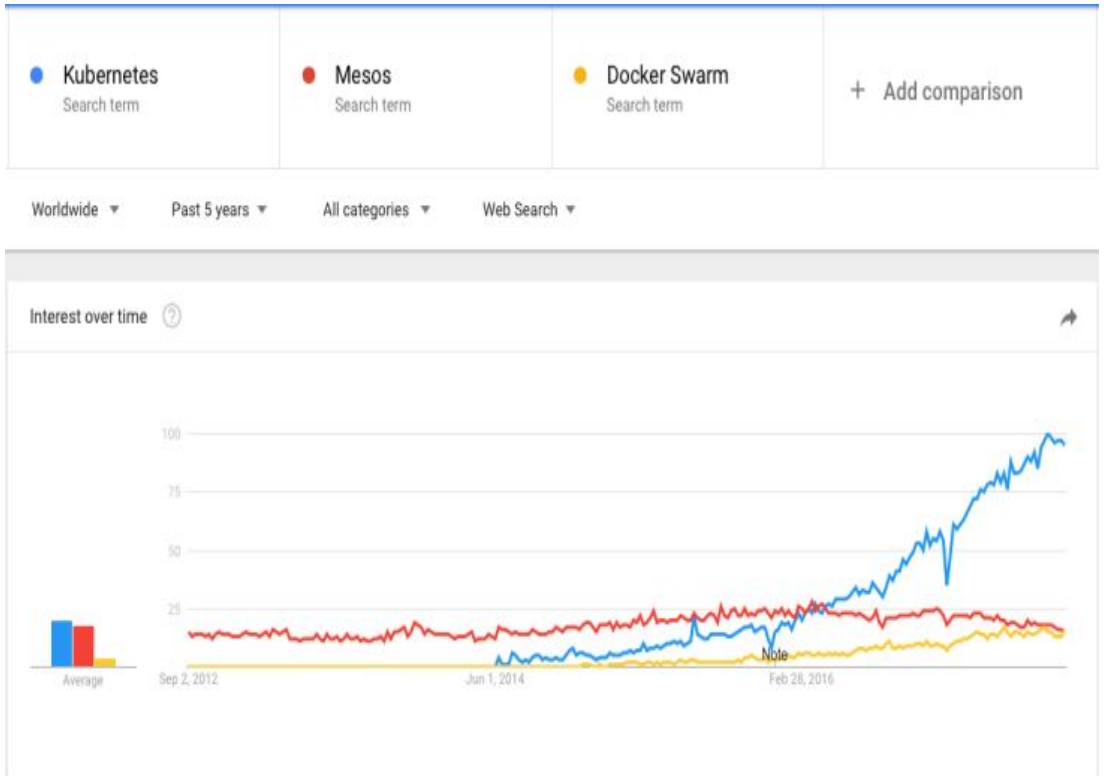
Challenges with multiple containers



- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?

Container Orchestration Tools

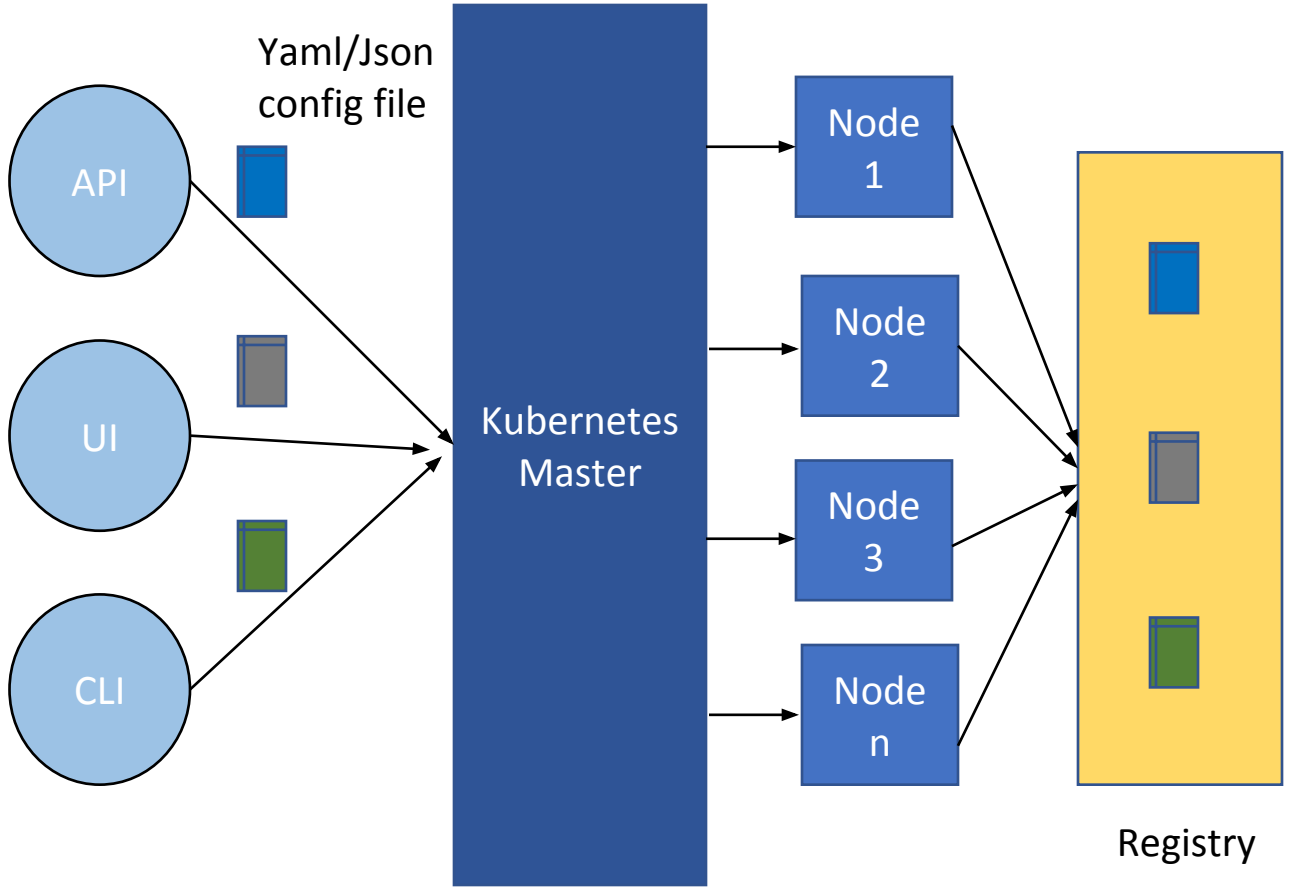
- The three most popular are: Kubernetes, Docker Swam & Mesos
- Kubernetes has become the unofficial standard



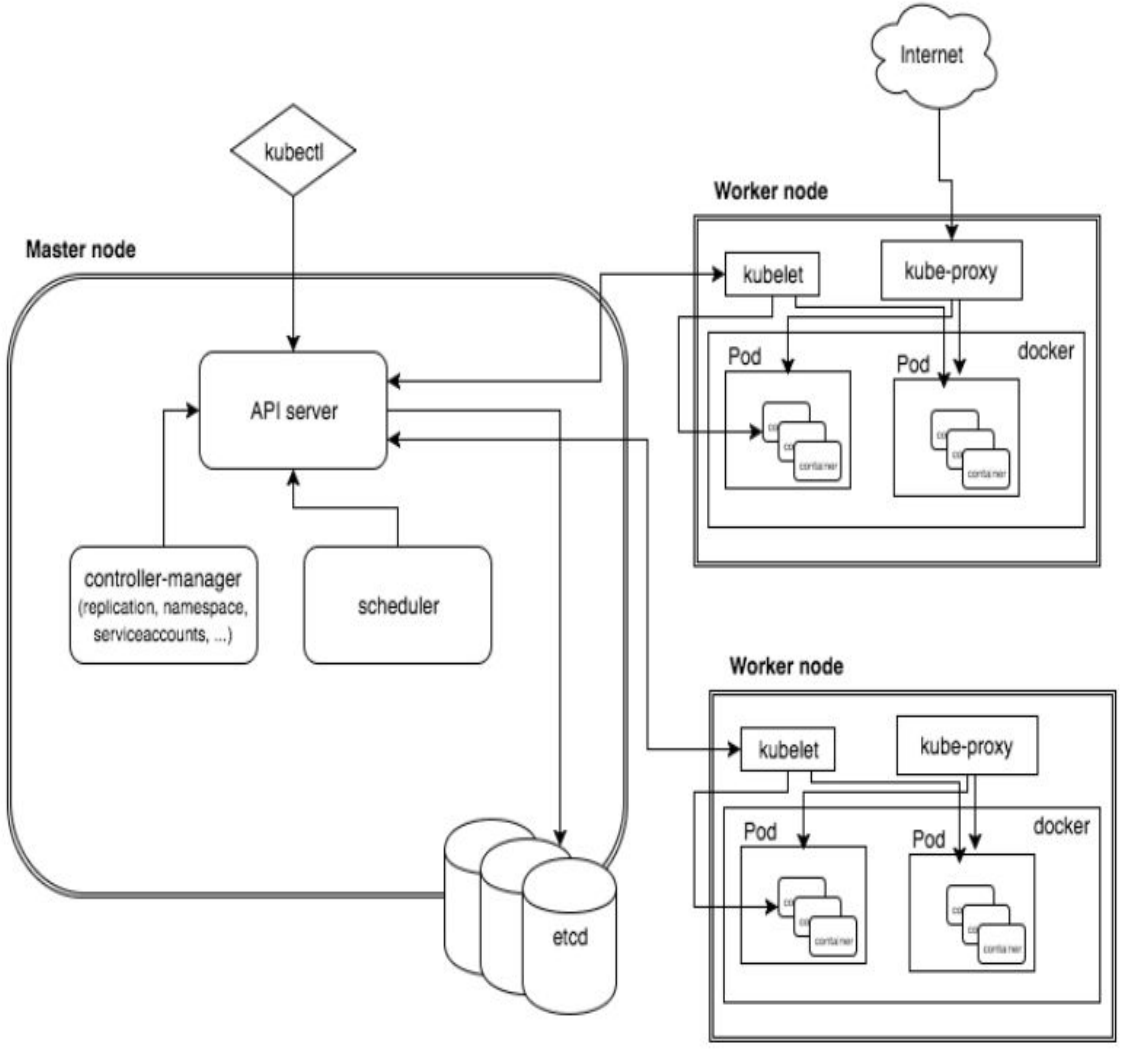
What is Kubernetes?

- Kubernetes is inspired from an internal Google project called **Borg**
- Open source **container orchestration** project managed by the Linux Foundation
- Unified API for deploying web applications, batch jobs, and databases
- Decouples applications from machines through containers
- Declarative approach to deploying applications
- Automates application configuration through service discovery
- Maintains and tracks the global view of the cluster
- APIs for deployment workflows
    - Rolling updates, Autoscaling

Kubernetes Architecture (Bird's eye view)



K8s Architecture – detailed



Install Kubernetes on AWS using KubeAdm

Lab: Single node cluster installation using KubeAdm

*** Master node need to have at least 2 CPUs*

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/kubernetes_single_node_cluster_installation.md

*** Do this lab the first thing – so that we all have machines with Docker which will be used to run exercises in lab 1*

*** Instructions for installing multi-node k8s cluster is provided in the above link*

Kubectl

Kubectl

- **kubectl** is Kubernetes command-line tool for running commands against Kubernetes clusters.
- It can be used to deploy and manage applications on Kubernetes.
- Using kubectl, you can inspect cluster resources; create, delete, and update components; look at your new cluster; and bring up example apps etc.

Kubernetes Java client
https://github.com/kubernetes-client/java#installation

Kubectl – Installation

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

On Ubuntu/Debian

sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y **kubectl**

To check installation (execute):

kubectl version

*** Note: We have already done it while K8s installation*

Kubernetes: Any Node

Any kubernetes **node** contains:

- Addresses
  - Hostname
  - External IP
  - Internal IP
- Condition - describe condition of nodes – OutOfDisk, Ready, MemoryPressure, DiskPressure, NetworkUnavailable
- Capacity - describe the resources available on the nods – CPU, memory, maximum number of pods that can be scheduled onto the node
- Info – general information about the node – kernel version, Kubernetes version, kubelet and kube-proxy version, Docker version, OS name

Command to get node details:

```
kubectl describe nodes <node_name>
```

Master Node

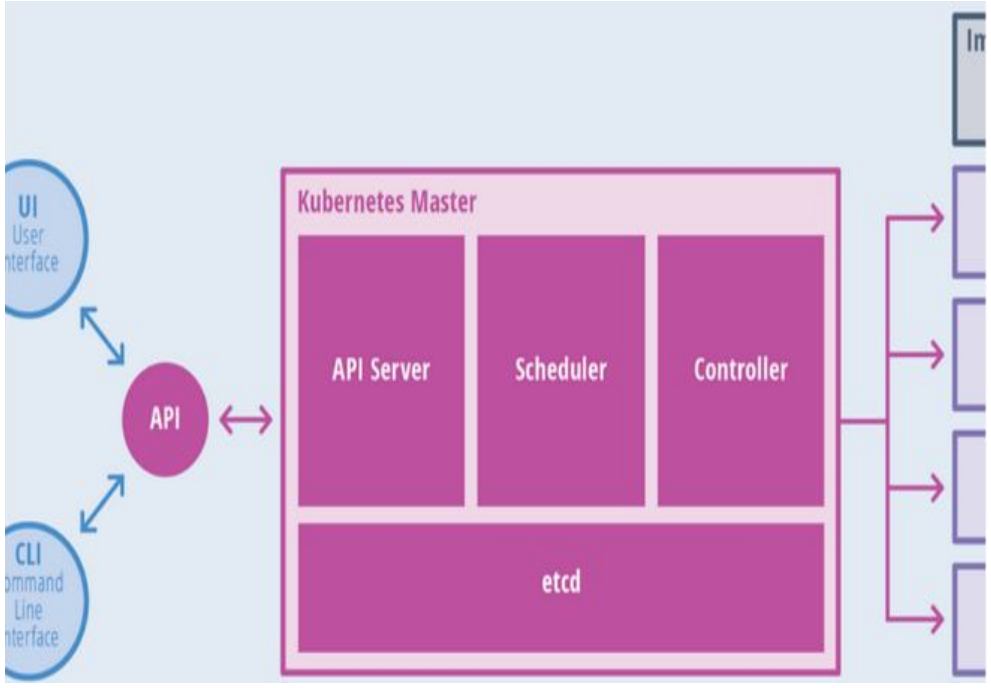
Master Node Role

- Master components make global decisions about the cluster (for example, scheduling), and detecting and responding to cluster events (starting up a new pod when a replication controller's 'replicas' field is unsatisfied).
- Master components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all master components on the same machine, and do not run user containers on this machine.

Master Node Components

Components of master:

- API Server
- Scheduler
- Controller
- etcd



Master Node Components

- The **API server** is the entry points for all the REST commands used to control the cluster. It processes REST requests, validates them, and executes the bound business logic. The result state has to be persisted in the “etcd” component.

- **Etcd** is an open source, distributed key-value database; it acts as a single source of truth (SSOT) for all components of the Kubernetes cluster. Masters query etcd to retrieve various parameters of the state of the nodes, pods and containers. Etcd is considered a metadata service in Kubernetes.

Master Node Components

- **Controller Manager** is responsible for most of the collectors that regulate the state of the cluster. In general, a controller can be considered a daemon that runs in nonterminating loop and is responsible for collecting and sending information to the API server. It works toward getting the shared state of cluster and then making changes to bring the current status of the server to the desired state. The key controllers are **replication controller**, **namespace controller**, and **service account controller**. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.

- **Scheduler** is one of the key components of Kubernetes master. It is responsible for distributing the workload, tracking resource utilization on cluster nodes and selecting the nodes for the workloads to run. In other words, this is the mechanism responsible for allocating pods to available nodes.

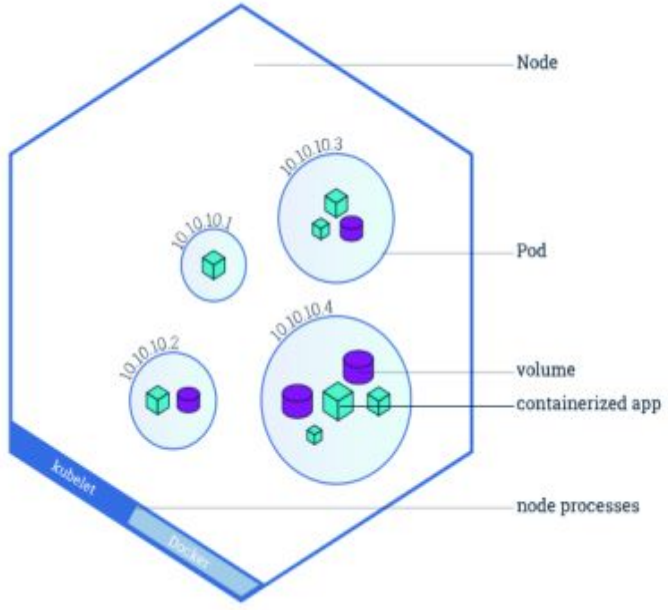
Worker Node

Worker Node Roles

A node is a worker machine in Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include **Docker**, **kubelet** and **kube-proxy**.

Allows Pods to be scheduled. A basic worker physical or virtual machine of Kubernetes.

- Must be managed by a master
- May host multiple pods
- Internal IP Address endpoint
- Can be tagged and filtered using labels



Worker Node Components

- **kubelet** Service interacts with etcd store to read configuration details and to write values **via api-server**. It communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.
- kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

Worker Node Components

- ***kube-proxy*** enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding.
- Kubernetes **Proxy** Service is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.