MASTER-ACADEMY

Container Orchestration using Kubernetes Part - 1

Week 4

Course Outline
Module 9

WEEK 1
1. Overview of DevOps
2. Oracle Virtual Box
3. Linux commands and file system

WEEK 2
4. Version Control with Git
5. Continuous Integration with Jenkins
6. Continuous Testing with Selenium

WEEK 3
7. Continuous Deployment: Containerization with Docker
8. Containerization with Docker: Ecosystem and Networking

WEEK 4
9. Container Orchestration using Kubernetes
10. Configuration Management with Ansible

WEEK 5
11. Continuous Monitoring Nagios
12. Introduction to DevOps on Cloud

WEEK 6
13. Introduction to SSH
14. High Performance Server NGINX

Topics

- **Part - 1**
- Container Orchestration Concept
- Introduction to Kubernetes
- Features and Advantages of Kubernetes
- Creating a Kubernetes Cluster
- **Part - 2**
- Deployment in Kubernetes
- Services in Kubernetes
- Rolling updates in Kubernetes
- Kubernetes Dashboard

MASTER-ACADEMY

Objectives

- Understand the basics of Kubernetes
- Understand Kubernetes Architecture
- Set up a Kubernetes Cluster on Ubuntu VMs
- Deploy your first app on Kubernetes using YAML file
- Deploying an On-Prem application on Kubernetes using Dashboard
- Update your application Pod using a Blue Green Deployment on Kubernetes

Revisiting Containers

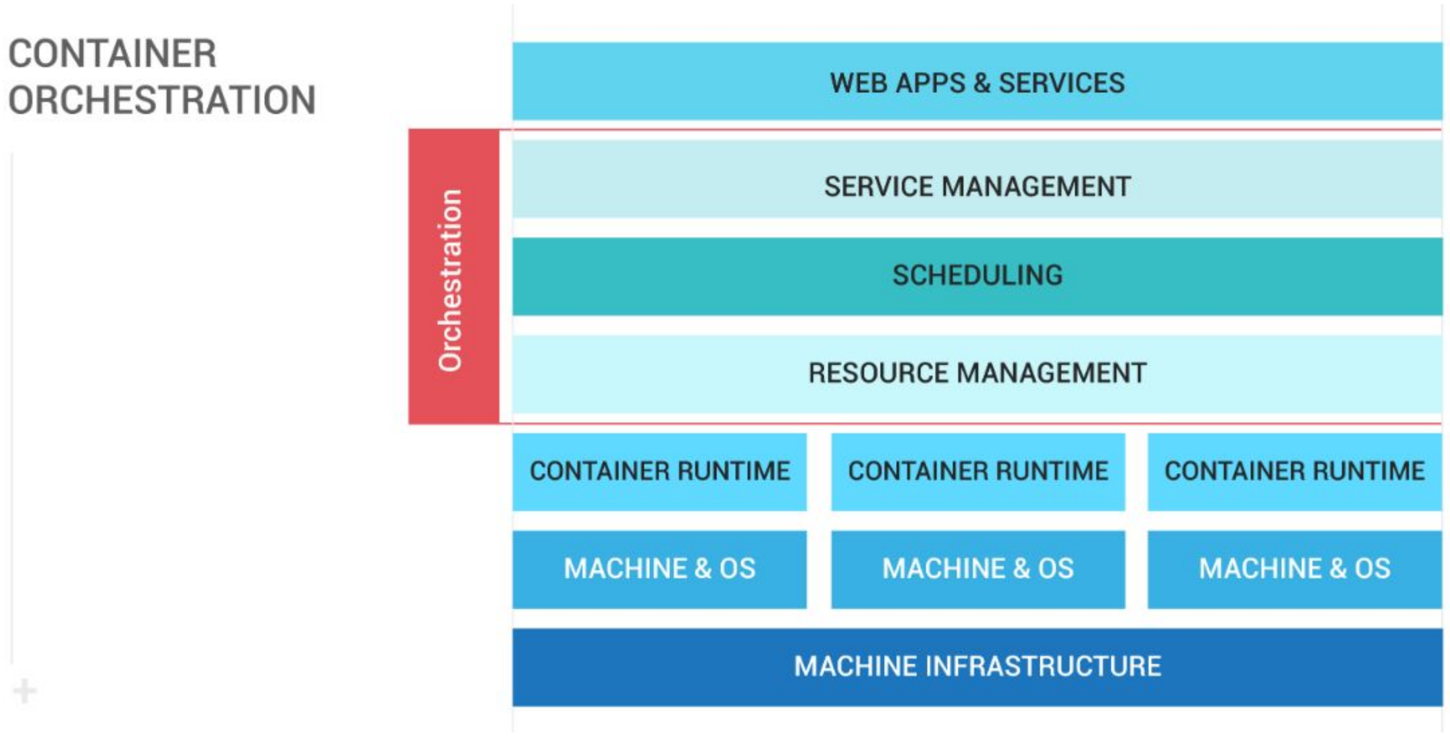
“A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime environment, system tools, system libraries and settings”.

- Containers isolate software from its surroundings
- It reduces conflicts between teams running different software on the same infrastructure

Why Container Orchestration?

- Container orchestration manages the **availability**, **scaling** and **networking** of containers
- It helps in monitoring the cluster i.e., group of hosts
- It helps in managing the timing of container creations
- It helps in container configuration in order to allow containers to communicate with one another

Container Orchestration



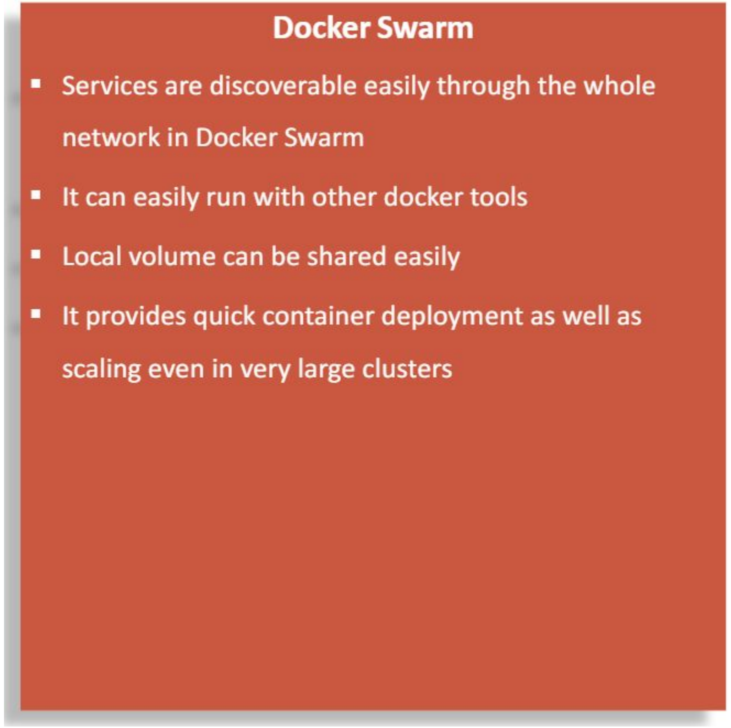
Top 3 Container Orchestrators

Docker Swarm

Mesos

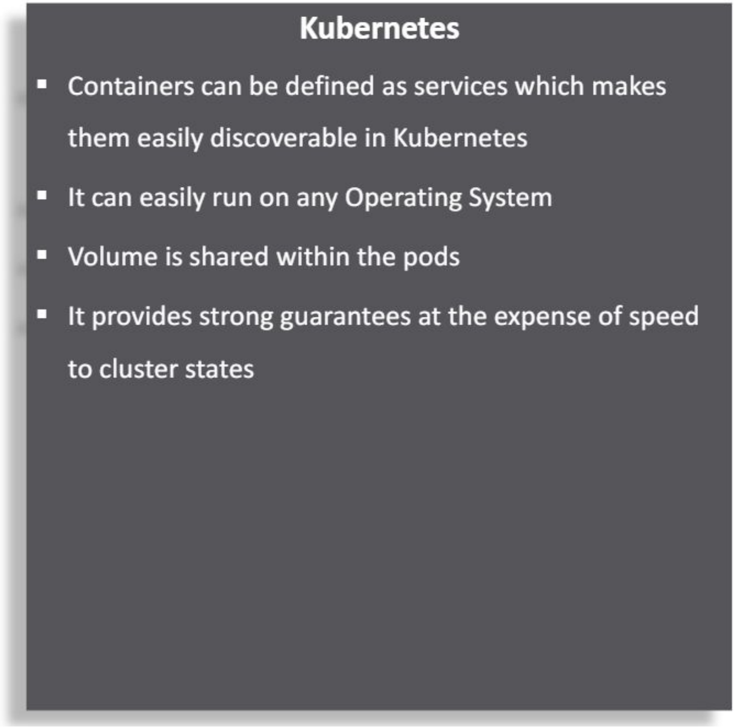
Kubernetes

Docker Swarm Vs Kubernetes



Docker Swarm

- Services are discoverable easily through the whole network in Docker Swarm
- It can easily run with other docker tools
- Local volume can be shared easily
- It provides quick container deployment as well as scaling even in very large clusters



Kubernetes

- Containers can be defined as services which makes them easily discoverable in Kubernetes
- It can easily run on any Operating System
- Volume is shared within the pods
- It provides strong guarantees at the expense of speed to cluster states

What is Kubernetes

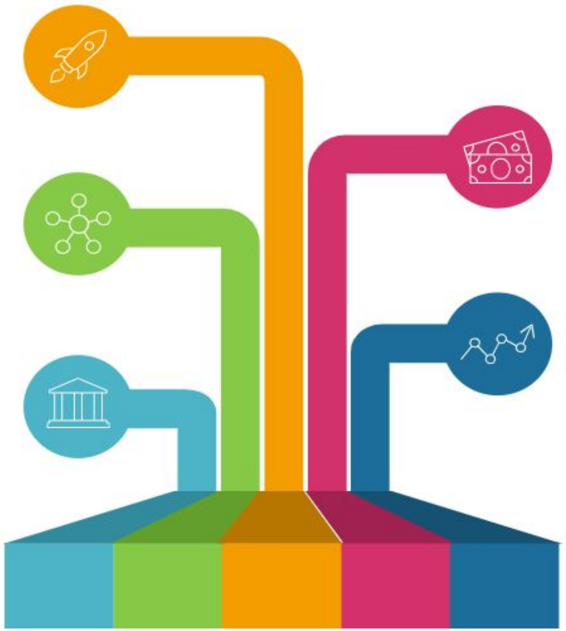
Kubernetes is an open source orchestration system which is used for:

- Deployment of containerized application
- Scaling of containerized application
- Management of containerized application

Kubernetes enables to:

- Run multiple containers on a single machine
- Schedule containers on cluster of machines
- Run log running services such as web applications

Features of Kubernetes



Automatic bin-packing

Automatically places containers based on their resource requirements and other constraints, while not sacrificing the availability.

Horizontal Scaling

Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.

Self-Healing

Automatically restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond.

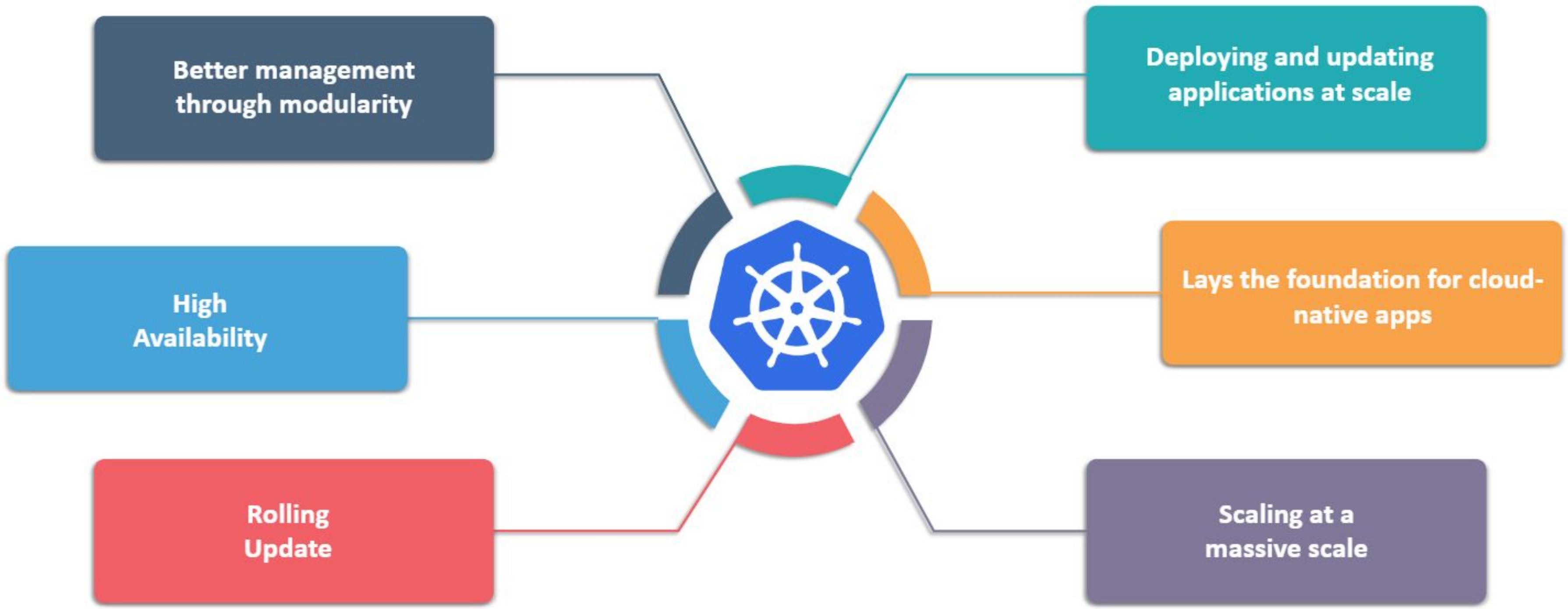
Batch Execution

In addition to services, Kubernetes can manage your batch and CI workloads as well as replacing containers that fail, if needed.

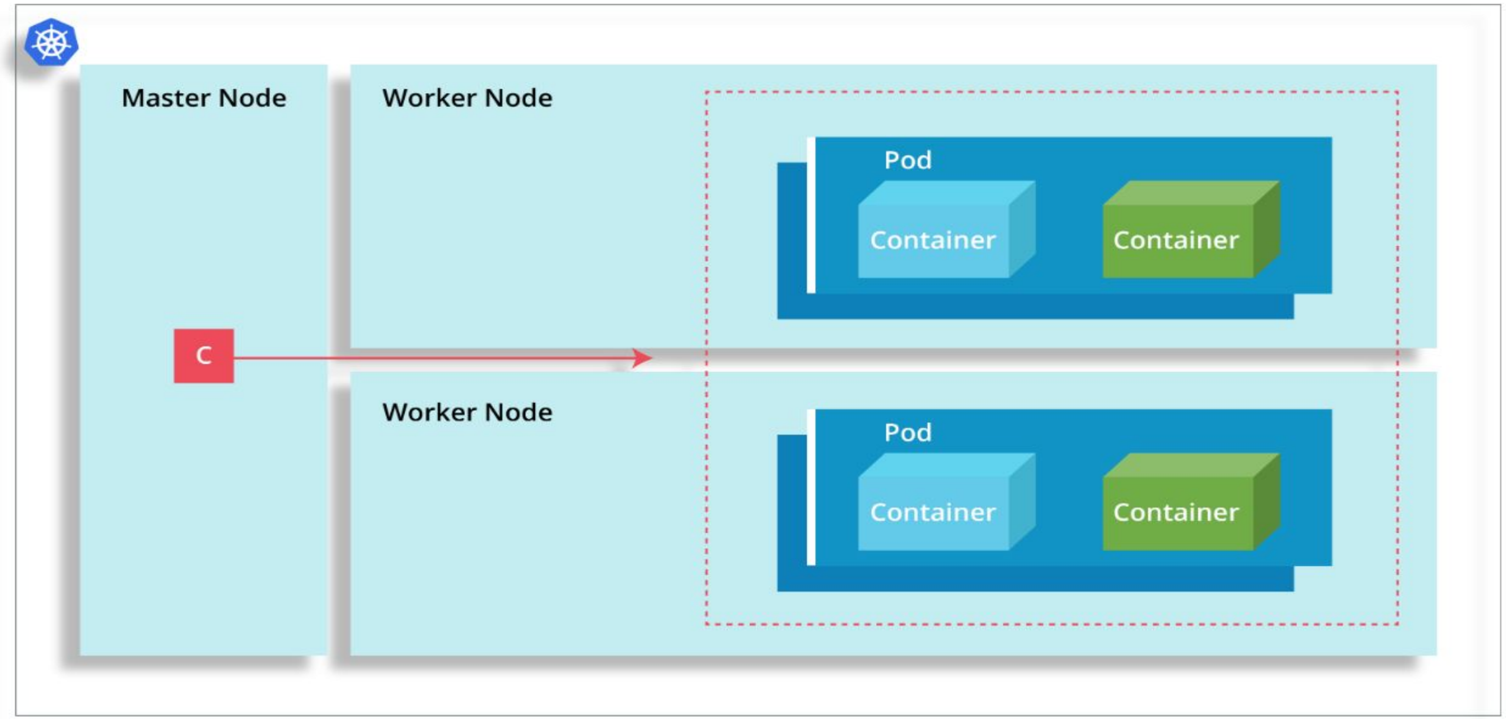
Storage-Orchestration

Automatically mount the storage system of your choice, whether from local storage or a public cloud provider such as GCP, AWS.

Advantages Of Kubernetes


Better management through modularity
High Availability
Rolling Update
Deploying and updating applications at scale
Lays the foundation for cloud-native apps
Scaling at a massive scale

Kubernetes Cluster Architecture



Kubernetes Cluster Components

Master Node : Cluster master manages the Kubernetes API server, resource controller and scheduler. It's lifecycle is managed by Kubernetes engine when starting

Worker Node : Worker node previously known as minions may be physical machine or a virtual machine based on the cluster

Pods : Pods are group of containers which are tightly coupled together. This is done, when the containers are dependent on each other.

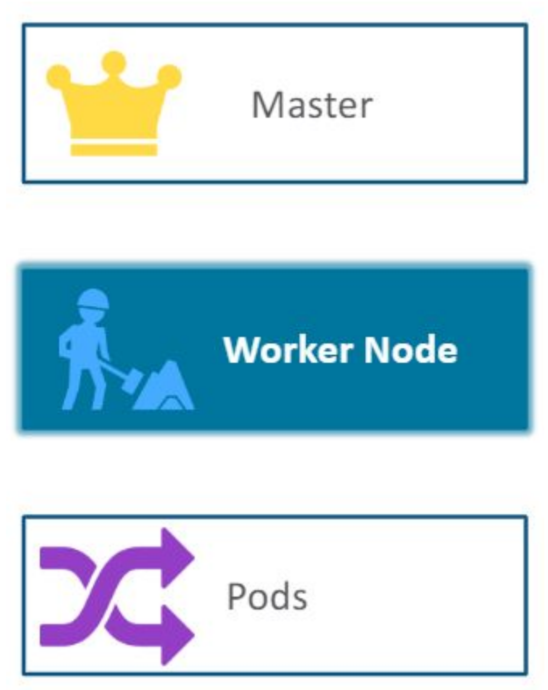
Let's talk about each of these in detail:

Kubernetes Master

Kubernetes master is a collection of four processes:

- **Kube API Server**: It validates and configures all the data for the API objects which include pods, servies, replica controllers and others
- **Kube controller manager**: It's a daemon that includes the non terminal loops (that regulates the state of the system) shipped with Kubernetes
- **Etcd**: It is a distributed key-value store designed to reliably and quickly preserve and provide access to critical data
- **Kube scheduler**: The Kubernetes Scheduler is a workload-specific function that significantly impacts availability, performance, and capacity. Workload-specific requirements will be exposed through the API as required

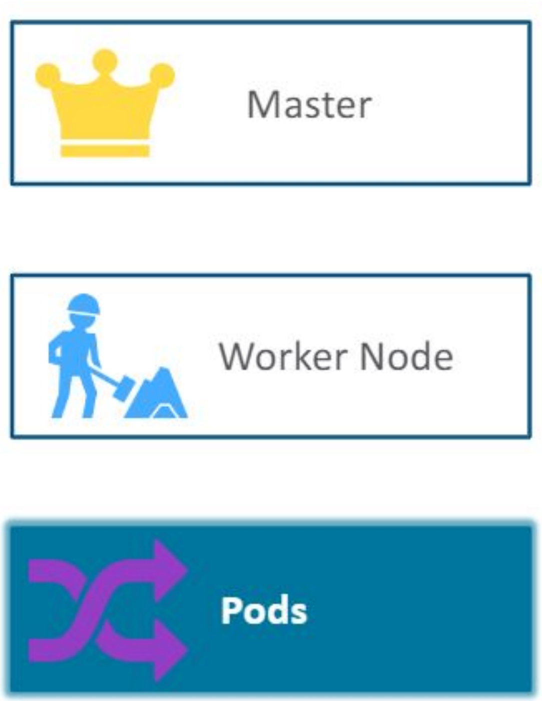
Kubernetes Worker Node



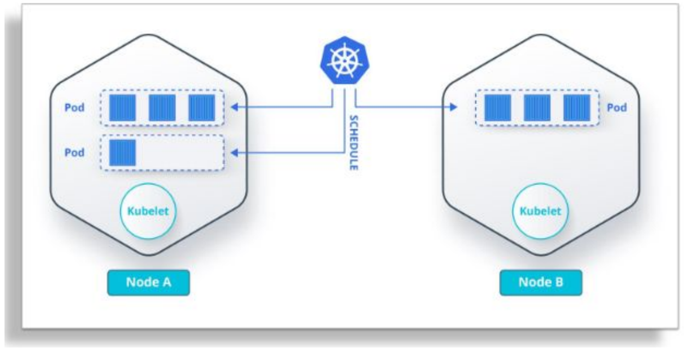
Worker node in the cluster runs three processes:

- **Kubelet**: It's a foremost node agent running on each node works under the terms of PodSpec. A PodSpec is a YAML or JSON object that describes a pod
- **Kube Proxy**: Kubernetes network proxy runs on each node
- **Container Runtime**: The container runtime is the software that is responsible for running containers. Kubernetes supports several runtimes: Docker, rkt, runc and OCI runtime spec implementation

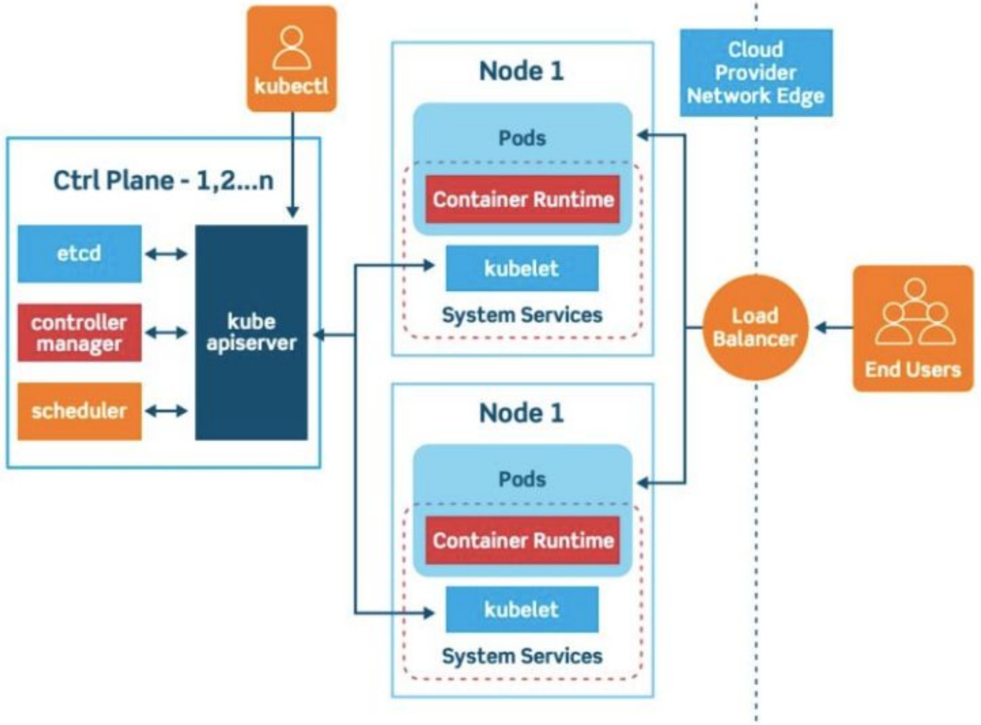
Kubernetes Pods



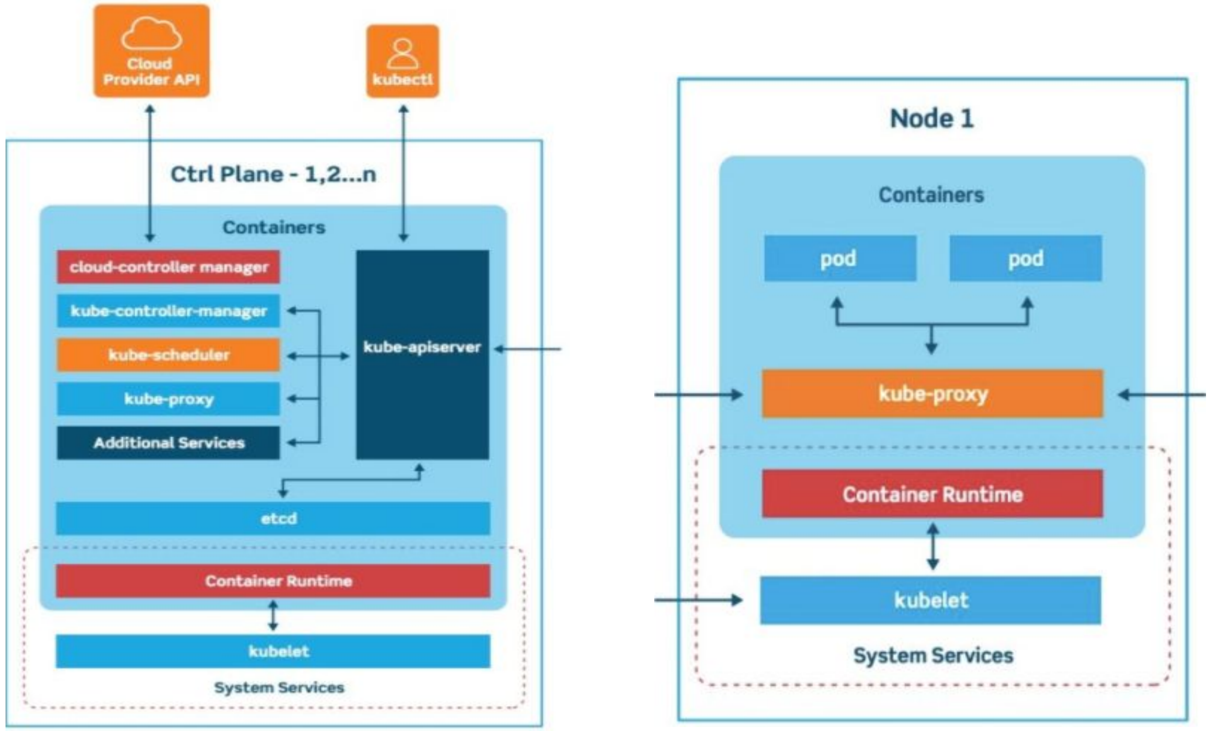
- Containers are deployed and scheduled through Kubernetes in a group called pods
- These are tightly coupled containers i.e the application running on them are dependent on each other
- 1 to 5 tightly coupled containers can be stored in a pod that collaborate to provide a service. This is called micro service



Inside a Kubernetes Cluster



Control Plane and Cluster Node

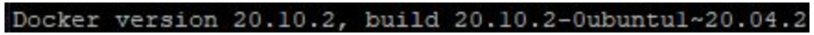


Setting Up a Cluster

- First need to spin up 3 VMs using any cloud service such as AWS or using VirtualBox
- Need to install docker on all 3 machines which provides the container runtime environment
- Install Kubeadm which bootstrap a new cluster
- Once Kubeadm is installed initialize the master
- Install a Pod network configuration, for this demo we will install flannel
- Once Pod is running join the node to the master

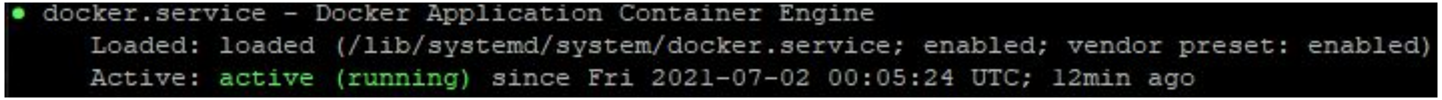
Installing Docker on all 3 Machines

- First need to update the package list in all 3 machines
  - `sudo apt-get update`
- Next install Docker on all 3 machines
  - `sudo apt-get install docker.io`
- Next check the version of Docker on all 3 machines
  - `docker --version` →



```
Docker version 20.10.2, build 20.10.2-0ubuntu1~20.04.2
```

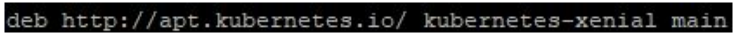
- Next set Docker to launch at boot in all 3 machines
  - `sudo systemctl enable docker`
- Next verify if Docker is running on all 3 machines
  - `sudo systemctl status docker` →



```
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2021-07-02 00:05:24 UTC; 12min ago
```

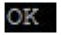
Install Kubernetes in all 3 Machines

- Since you are downloading Kubernetes from a non-standard repository, it is essential to ensure that the software is authentic. This is done by adding a signing key
  - `echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list`



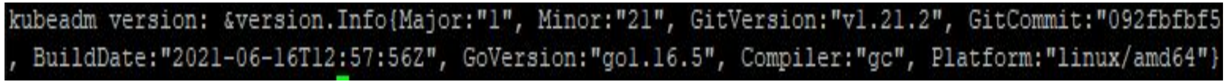
```
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

  - `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add`



```
OK
```

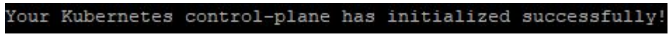
- Since Kubernetes is not included in the default repo, add software repo:
  - `sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"`
- Enter the following commands to install KUBEADM, KUBELET, KUBECTL:
  - `sudo apt-get install kubeadm kubelet kubectl`
  - `sudo apt-mark hold kubeadm kubelet kubectl`
- Verify the installation:
  - `kubeadm version`



```
kubeadm version: &version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.2", GitCommit:"092fbfbf5
, BuildDate:"2021-06-16T12:57:56Z", GoVersion:"go1.16.5", Compiler:"gc", Platform:"linux/amd64"}
```

Kubernetes Deployment

- In order to start Kubernetes deployment start by disabling swap memory for each server:
  - `sudo swapoff -a`
- Assign unique hostname for each server node or machine:
  - `sudo hostnamectl set-hostname master-node`
  - `sudo hostnamectl set-hostname worker01`
  - `sudo hostnamectl set-hostname worker02`
- Initialize Kubernetes on Master Node ONLY. Please note cidr 10.244.0.0/16 is designated for Flannel network. This network is required for Pods to communicate with each other.

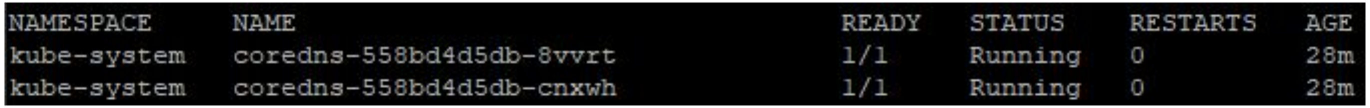


```
Your Kubernetes control-plane has initialized successfully!
```

  - `sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all`
- Please copy the printout kubeadm join command (at the end of the printout) for later use. This command is unique to each cluster and will be used for joining other nodes to the cluster. Need to run the following commands to complete deployment on Master node:
  - `mkdir -p $HOME/.kube`
  - `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
  - `sudo chown $(id -u):$(id -g) $HOME/.kube/config`

Deploy Pod Network to Cluster

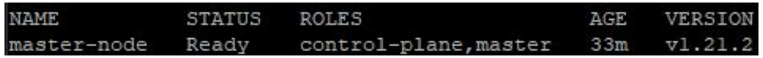
- A Pod Network is a way to allow communication between different nodes in the cluster. This tutorial uses the **flannel** virtual network. Enter the following command on Master node:
  - `kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml`
- Allow the process to complete and verify that everything is running and communicating:
  - `kubectl get pods --all-namespaces`
- The coredns should be up and running



```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE
kube-system   coredns-558bd4d5db-8vvrt      1/1     Running   0          28m
kube-system   coredns-558bd4d5db-cnxwh      1/1     Running   0          28m
```

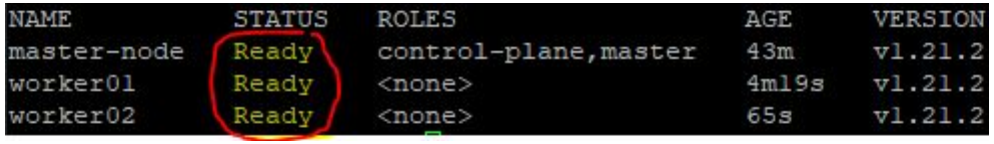
Join Worker Node to Cluster

- Check the nodes on the current cluster by running the following command in Master node:
    - `kubectl get nodes`



```
NAME          STATUS   ROLES                  AGE   VERSION
master-node   Ready    control-plane,master   33m   v1.21.2
```

- As mentioned in the earlier steps (see Kubernetes deployment), now kubeadm join command can be used in each worker node to connect them to the cluster. Run the command on each Worker node (the token part is excluded, since your token and IP should be unique):
    - `sudo kubeadm join 172.31.x.x:6443 --token`
- Now run the "kubectl get nodes" on the Master node to check node status
    - `kubectl get nodes`



```
NAME          STATUS   ROLES                  AGE     VERSION
master-node   Ready    control-plane,master   43m     v1.21.2
worker01      Ready    <none>                 4m19s   v1.21.2
worker02      Ready    <none>                 65s     v1.21.2
```

End of Part 1

- Practice creating a Kubernetes Cluster on your Linux Environment
- Go through the following link for a better understanding of concept

https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts/