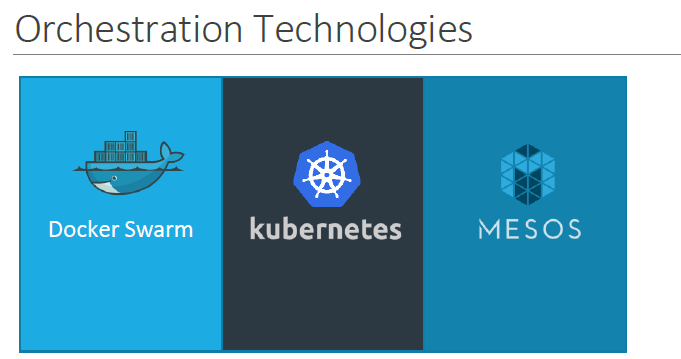


So we learned about containers and we now have our application packaged into a docker container. But what next? How do you run it in production? What if your application relies on other containers such as database or messaging services or other backend services? What if the number of users increase and you need to scale your application? You would also like to scale down when the load decreases.

To enable these functionalities you need an underlying platform with a set of resources. The platform needs to orchestrate the connectivity between the containers and automatically scale up or down based on the load. This whole process of automatically deploying and managing containers is known as Container Orchestration.

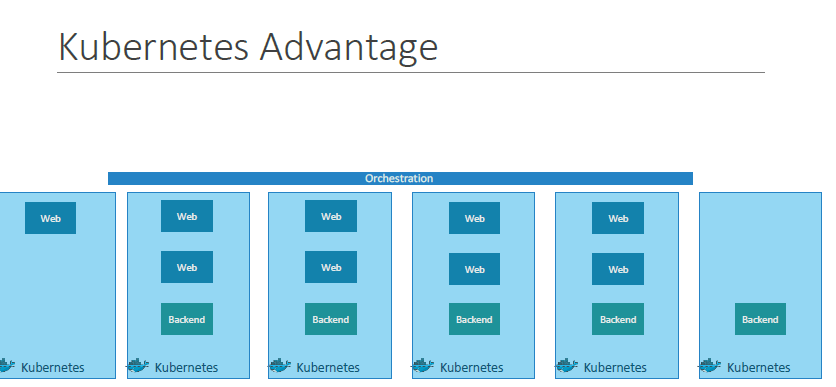


Kubernetes is thus a container orchestration technology. There are multiple such technologies available today – Docker has its own tool called Docker Swarm.

Kubernetes from Google and Mesos from Apache. While Docker Swarm is really easy to setup and get started, it lacks some of the advanced autoscaling features required for complex applications. Mesos on the other hand is quite difficult to setup and get started, but supports many advanced features. Kubernetes - arguably the most popular of it all – is a bit difficult to setup and get started but provides a lot of options to customize deployments and supports deployment of complex architectures.

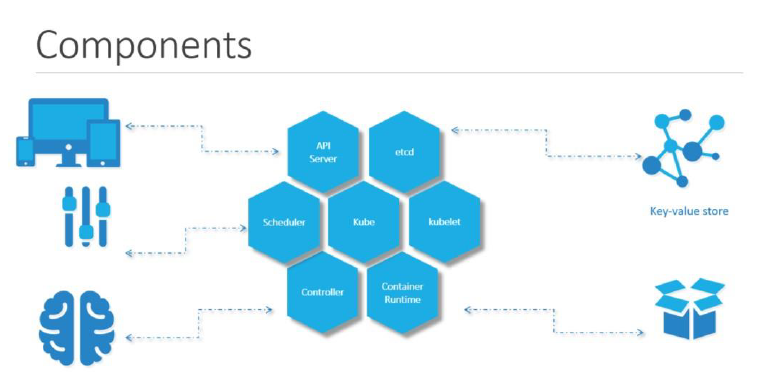
Kubernetes is now supported on all public cloud service providers like GCP, Azure and

AWS and the kubernetes project is one of the top ranked projects in Github.



There are various advantages of container orchestration. Your application is now highly available as hardware failures do not bring your application down because you have multiple instances of your application running on different nodes. The user traffic is load balanced across the various containers. When demand increases, deploy more instances of the application seamlessly and within a matter of second and we have the ability to do that at a service level. When we run out of hardware resources, scale the number of nodes up/down without having to take down the application. And do all of these easily with a set of declarative object configuration files.

And THAT IS Kubernetes. It is a container Orchestration technology used to orchestrate the deployment and management of 100s and 1000s of containers in a clustered environment. Don’t worry if you didn’t get all of what was just said, in the upcoming lectures we will take a deeper look at the architecture and various concepts surrounding kubernetes. That is all for this lecture, thank you for listening and I will see you in the next lecture



When you install Kubernetes on a System, you are actually installing the following components. An API Server. An ETCD service. A kubelet service. A Container Runtime, Controllers and Schedulers.

The API server acts as the front-end for kubernetes. The users, management devices, Command line interfaces all talk to the API server to interact with the kubernetes cluster.

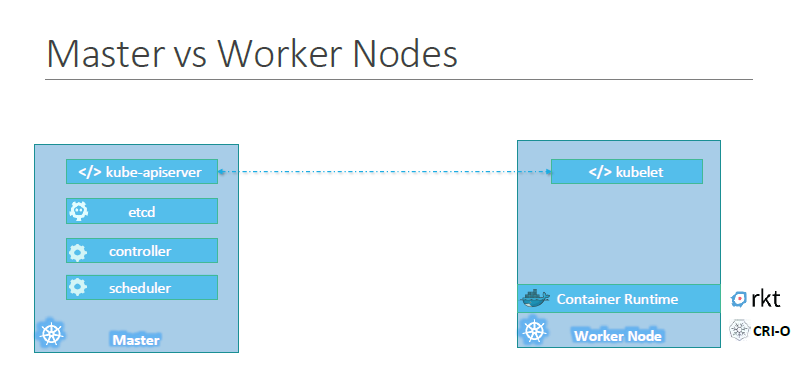
Next is the ETCD key store. ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. Think of it this way, when you have multiple nodes and multiple masters in your cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner. ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.

The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.

The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

The container runtime is the underlying software that is used to run containers. In our case it happens to be Docker.

And finally kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.



So far we saw two types of servers – Master and Worker and a set of components that make up Kubernetes. But how are these components distributed across different types of servers. In other words, how does one server become a master and the other slave?

The worker node (or minion) as it is also known, is were the containers are hosted. For example Docker containers, and to run docker containers on a system, we need a container runtime installed. And that’s were the container runtime falls. In this case it happens to be Docker. This doesn’t HAVE to be docker, there are other container runtime alternatives available such as Rocket or CRIO. But throughout this course we are going to use Docker as our container runtime.

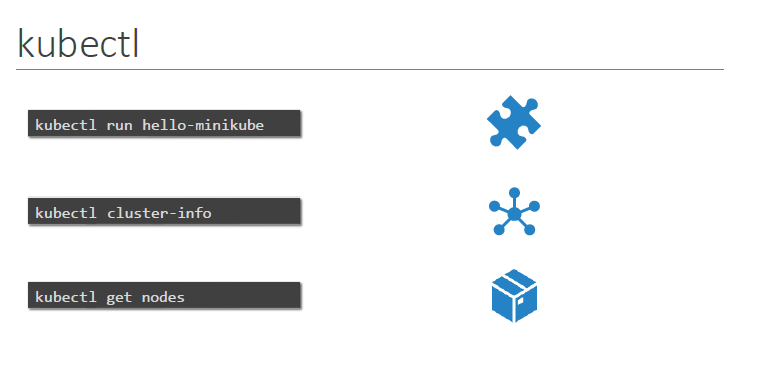
The master server has the kube-apiserver and that is what makes it a master.

Similarly the worker nodes have the kubelet agent that is responsible for interacting with the master to provide health information of the worker node and carry out actions requested by the master on the worker nodes.

All the information gathered are stored in a key-value store on the Master. The key value store is based on the popular etcd framework as we just discussed.

The master also has the controller manager and the scheduler.

There are other components as well, but we will stop there for now. The reason we went through this is to understand what components constitute the master and worker nodes. This will help us install and configure the right components on different systems when we setup our infrastructure.



And finally, we also need to learn a little bit about ONE of the command line utilities known as the kube command line tool or kubectl or kube control as it is also called. The kube control tool is used to deploy and manage applications on a kubernetes cluster, to get cluster information, get the status of nodes in the cluster and many other things.

The kubectl run command is used to deploy an application on the cluster. The kubectl cluster-info command is used to view information about the cluster and the kubectl get pod command is used to list all the nodes part of the cluster. That’s all we need to know for now and we will keep learning more commands throughout this course. We will explore more commands with kubectl when we learn the associated concepts.

For now just remember the run, cluster-info and get nodes commands and that will help us get through the first few labs.



There are lots of ways to setup Kuberentes. We can setup it up ourselves locally on our laptops or virtual machines using solutions like Minikube and Kubeadmin.

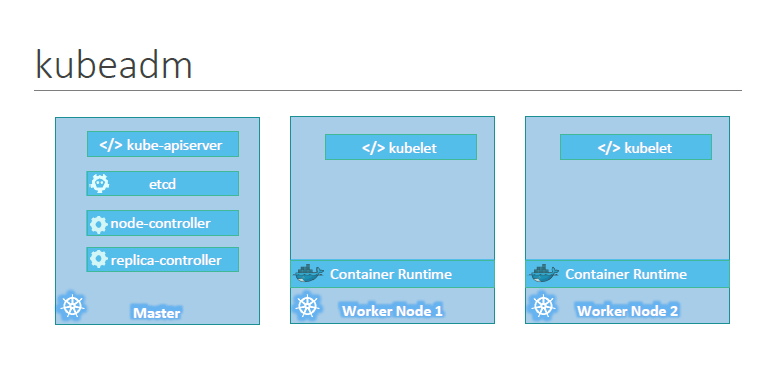
Minikube is a tool used to setup a single instance of Kubernetes in an All-in-one setup and kubeadmin is a tool used to configure kubernetes in a multi-node setup. We will look more into that in a bit.

There are also hosted solutions available for setting up kubernetes in a cloud

environment such as GCP and AWS. We will also have some demos around those.

And finally if you don’t have the resources or if you don’t want to go through the hassle of setting it all up yourself, and you simply want to get your hands on a kubernetes cluster instantly to play with, checkout play-with-k8s.com . I also have a demo on this.

So feel free to chose the one that is right for you. You need not go through all the demos, pick the ones that best suite your needs based on your time and resources.



Kubernetes Cluster Installation using Kubeadm tool:

Please follow the instructions below to proceed with Kubernetes cluster installation using kubeadm tool.

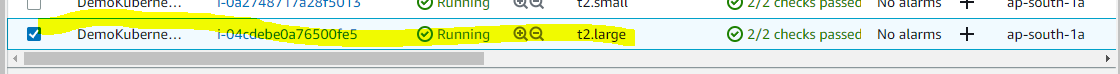
Prerequisites:

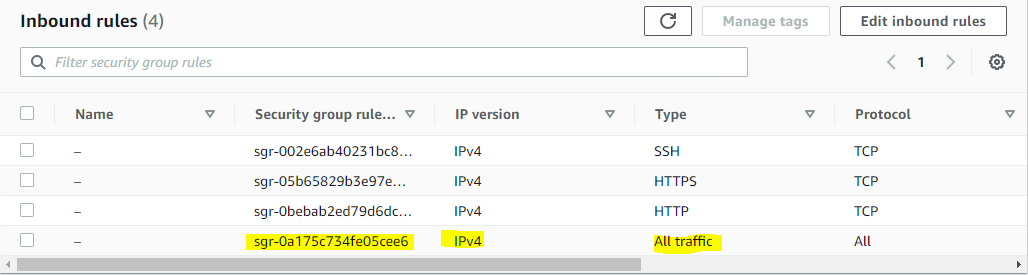
Two ec2 instance

MasterNode:

Centos Image, T2.large,

Security Group : All Traffic





Workernode:

Centos Image: t2.large

Security Group : All Traffic (As highlighted above)

Please follow the below steps to set up Kubernetes cluster both on Master and WorkerNode

Note: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

Note: All the steps must be followed both on master and worker node

Steps:

Command 1

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch

enabled=1

gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kubelet kubeadm kubectl

EOF

Command 2

sudo setenforce 0

Command 3

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

Command 4

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes

Command 5

sudo systemctl enable --now kubelet

Note: After the Kubernetes Version 1.23 any latest version of it doesn’t have the containerD(C-advisor utility comes with kubeadm tool we have to install it separately please follow the steps below both on master and workernode to install the same)

Command 6

sudo yum -y install yum-utils

Command 7

yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

Command 8

yum install -y containerd.io

Command 9

sudo mkdir -p /etc/containerd

Command 8

containerd config default | sudo tee /etc/containerd/config.toml

Command 9

systemctl restart containerd

Note: This will finish our Kubernetes cluster installation we have to initiate the cluster now. From now on all the command will be executed from master node

Initializing the Kubernetes cluster (Master Node)

Note: This command should only be executed on the master node

kubeadm init --apiserver-advertise-address=172.31.46.136(PrivateIPofOurec2Instance) --pod-network-cidr=192.168.0.0/16 ( We are using calico as a network utility as by default Kubernetes will not come with the networking. There are different tools available in the market which we can use as network utility for Kubernetes such as Calico, Weavenet, Fennel, Jaguar, Cisco etc.

All these network utility will have the CIDR range i.e. Example 192.168.0.0 ( IP range for our pod creation using calico network utility). Depending on the network utility the IP CIDR range will be allocated to the pods.

**Note:** [**https://projectcalico.docs.tigera.io/getting-started/kubernetes/quickstart**](https://projectcalico.docs.tigera.io/getting-started/kubernetes/quickstart)

Once the above step completed you will have to run few commands on the master node to use our command line utility which as kubectl as a normal user

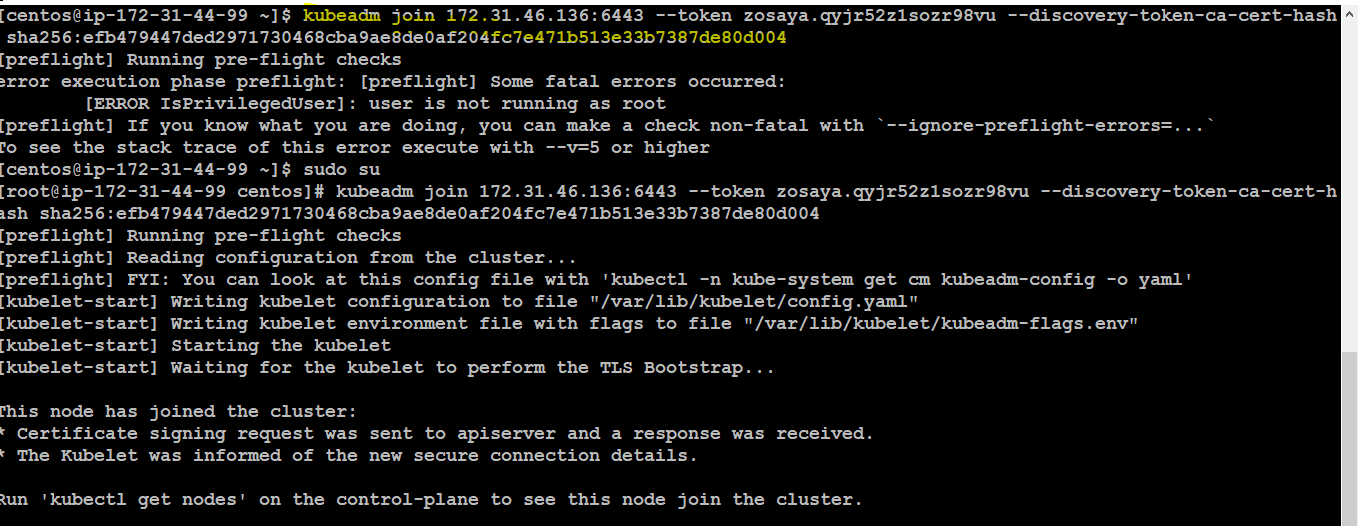
**Note all the below command has to be run as a normal user not a root user**

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

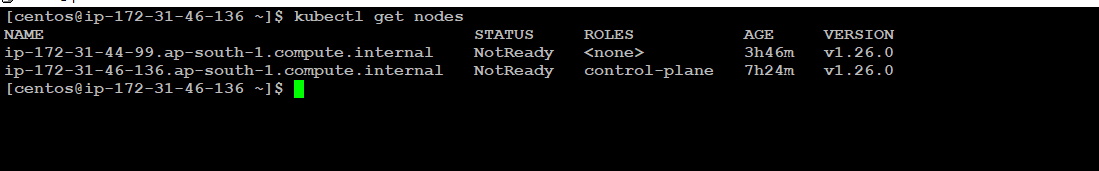
After you have completed running the above command you have to join or worker node to the cluster. There will be a command displayed on the screen of master node just copy it and paste it on the slave node



kubeadm join 172.31.46.136:6443 --token zosaya.qyjr52z1sozr98vu --discovery-token-ca-cert-hash sha256:efb479447ded2971730468cba9ae8de0af204fc7e471b513e33b7387de80d004

**Note: the IP address and the token will be different as per your master node and the above command has to be run on the worker node/Slave node**

Below is the command to check if the node has been added to the cluster or not. The status of the node will show as not ready as we have to configure the calico network utility . Below command must be run on the master node



Command: kubectl get nodes ( It will show all the nodes(VM) in the cluster. Under Roles column if you see control plane that means it is your master node and the one which as none that is your worker node(slave Node)