**------------------**

**EKS BASICS**

**------------------**

**ABOUT THIS LAB**

Elastic Kubernetes Service (EKS) is a fully managed Kubernetes service from AWS. In this lab, you will work with the AWS command line interface and console, using command line utilities like **eksctl** and **kubectl** to launch an EKS cluster, provision a Kubernetes deployment and pod running instances of nginx, and create a LoadBalancer service to expose your application over the internet.

**Course files can be found here:**

<https://github.com/insighttechworld/Kubernetes_yaml_File.git>

**LEARNING OBJECTIVES**

Create an IAM User with Admin Permissions

Launch an EC2 Instance and Configure the Command Line Tools

Provision an EKS Cluster

Create a Deployment on Your EKS Cluster

Test the High Availability Features of Your EKS Cluster

**--------------Additional Resources**

Your task is to launch an EKS cluster with three worker nodes in the us-east-1 region. Then, create a Deployment, and a Pod running container instances using the public nginx image. Then, expose your application to the internet by adding a LoadBalancer service.

**--------------Additional Info**

* Note that all AWS resources should be created in the us-east-1 region.
* You will also need to make sure you are working in the us-east-1 region when configuring and using the AWS CLI.
* Use the AWS CLI v2.x or later.
* You can find information on the official public image repository on Docker Hub.
* Here's info on installing the AWS CLI.
* Here's how to get started with EKS.

Course files can be found here: https://github.com/insighttechworld/Kubernete-Basics.git

**--------------Learning Objectives**

**Create an IAM User with Admin Permissions**

* Create an IAM user with programmatic access and administrator-level privileges.
* Take note of the access key and secret access key of your user because we will use them in the next step.

**Launch an EC2 Instance and Configure the Command Line Tools**

* Create an EC2 instance in us-east-1region.
* If necessary, upgrade the AWS CLI on your EC2 instance to **CLI v.2x** or later.
* Configure the AWS CLI using the credentials of the user you just created.
* Install **eksctl** on your EC2 instance.
* Install **kubectl** on your EC2 instance.

**Notes:**

**eksctl** is a simple CLI tool for creating clusters on EKS - Amazon's new managed Kubernetes service for EC2. It is written in Go, uses CloudFormation, was created by Weaveworks and it welcomes contributions from the community. <https://eksctl.io/>

**kubectl** is a command line interface for running commands against Kubernetes clusters. It is the command line tool for **Kubernetes**, then enables the interaction with the cluster: to create pods, services and other components.

**Provision an EKS Cluster**

* Use eksctl to provision an EKS cluster with three worker nodes in us-east-1.
* Use Kubernetes version 1.16 or later.

**Create a Deployment on Your EKS Cluster**

* Course files can be found here: <https://github.com/insighttechworld/Kubernetes_yaml_File.git>
* Use kubectl to create a LoadBalancer service.
* Check the status of your LoadBalancer service using kubectl.
* Use kubectl to create a Deployment on your EKS cluster, using the standard nginx image EKS has available in the default Docker Hub registry.
* Check the status of your cluster, deployment, and pods using kubectl.
* When the Deployment is up and running, check that you can access your application using the DNS name of the LoadBalancer.

**Test the High Availability Features of Your EKS Cluster**

* In the AWS Console, shut down all the worker nodes.
* Check the status of your cluster, deployment, and pods using kubectl.
* After a few minutes, you should see EKS launching new instances to keep your service running.
* When the cluster is back to a steady state, check that your application is up and running.

-------------------------------------------

**LAB GUIDES: Commands Used**

------------------------------------------

**Launching an EKS Cluster**

**Introduction**

Elastic Kubernetes Service (EKS) is a fully managed Kubernetes service from AWS. In this lab, you will work with the AWS command line interface and console, using command line utilities like **eksctl** and **kubectl** to launch an EKS cluster, provision a **Kubernetes deployment** and **pod** running instances of **nginx**, and create a **LoadBalancer service** to expose your application over the internet.

Course files can be found here: https://github.com/insighttechworld/Kubernetes\_yaml\_File.git

**Solution**

Log in to the live AWS environment using the credentials provided. Make sure you're in the N. Virginia (us-east-1) region throughout the lab.

**Steps:**

* Create an IAM User with Admin Permissions
* Navigate to IAM > Users.
* Click Add user.
* Set the following values:
* Username: k8-admin
* Access type: Programmatic access
* Click Next: Permissions.
* Select Attach existing policies directly.
* Select AdministratorAccess.
* Click Next: Tags > Next: Review.
* Click Create user.
* Copy the access key ID and secret access key, and paste them into a text file, as we'll need them in the next step.
* Launch an EC2 Instance and Configure the Command Line Tools
* Navigate to EC2 > Instances.
* Click Launch Instance.
* On the AMI page, select the Amazon Linux 2 AMI.
* Leave t2.micro selected, and click Next: Configure Instance Details.
* On the Configure Instance Details page:
* Network: Leave default
* Subnet: Leave default
* Auto-assign Public IP: Enable
* Click Next: Add Storage > Next: Add Tags > Next: Configure Security Group.
* Click Review and Launch, and then Launch.
* In the key pair dialog, select Create a new key pair.
* Give it a Key pair name of "mynvkp".
* Click Download Key Pair, and then Launch Instances.
* Click View Instances, and give it a few minutes to enter the running state.
* Once the instance is fully created, check the checkbox next to it and click Connect at the top of the window.
* In the Connect to your instance dialog, select EC2 Instance Connect (browser-based SSH connection).
* Click Connect.

**In the command line window, check the AWS CLI version:**

aws --version

**Download v2 and Unzip it:**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

which aws

**Update it:**

sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update

**It should now be updated.**

aws --version

**Configure the CLI:**

aws configure

AWS Access Key ID: paste in the access key ID you copied earlier.

AWS Secret Access Key: paste in the secret access key you copied earlier.

Default region name: enter us-east-1.

Default output format: enter json.

**Confirm configuration:**

cd .aws

ls -l

cat credentials

cat config

cd

**Download kubectl:**

curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.15/2020-11-02/bin/linux/amd64/kubectl

chmod +x ./kubectl

**Copy the binary to a directory in your path:**

mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

**Ensure kubectl is installed:**

kubectl version --short --client

**Download eksctl:**

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

**Move the extracted binary to /usr/bin:**

sudo mv /tmp/eksctl /usr/bin

**Get the version of eksctl:**

eksctl version

**See the options with eksctl:**

eksctl help

**Provision an EKS Cluster**

**Provision an EKS cluster with three worker nodes in us-east-1:**

eksctl create cluster --name dev-cluster --version 1.18 --region us-east-1 --nodegroup-name standard-workers --node-type t3.micro --nodes 2 --nodes-min 1 --nodes-max 4 --managed

**OR**

eksctl create cluster --name eksdemo1 --version 1.19 --region us-east-1 --nodegroup-name eksdemo1-ng-public1 --node-type t3.micro --nodes 2 --nodes-min 2 --nodes-max 4 --node-volume-size=20 --ssh-access --ssh-public-key=kube-demo --managed

**In the AWS Management Console, navigate to CloudFormation and take a look at what’s going on there.**

* *Select the eksctl-dev-cluster stack (this is our control plane).*
* *Click Events, so you can see all the resources that are being created.*
* *We should then see another new stack being created — this one is our node group.*
* *Once both stacks are complete, navigate to Elastic Kubernetes Service > Clusters.*
* *Click the listed cluster.*
* *Click the Compute tab, and then click the listed node group. There, we'll see the Kubernetes version, instance type, status, etc.*
* *Click dev in the breadcrumb navigation link at the top of the screen.*
* *Click the Networking tab, where we'll see the VPC, subnets, etc.*
* *Click the Logging tab, where we'll see the control plane logging info.*
* *The control plane is abstracted — we can only interact with it using the command line utilities or the console. It’s not an EC2 instance we can log into and start running Linux commands on.*
* *Navigate to EC2 > Instances, where you should see the instances have been launched.*
* *Close out of the existing CLI window, if you still have it open.*
* *Select the original t2.micro instance, and click Connect at the top of the window.*
* *In the Connect to your instance dialog, select EC2 Instance Connect (browser-based SSH connection).*
* *Click Connect.*

**In the CLI, check the cluster:**

eksctl get cluster

**Enable it to connect to our cluster:**

aws eks update-kubeconfig --name dev-cluster --region us-east-1

**View clusters and nodes:**

eksctl get cluster

kubectl get nodes

**Create a Deployment on Your EKS Cluster**

**Install Git:**

sudo yum install -y git

**Download the course files:**

git clone https://github.com/ACloudGuru-Resources/Course\_EKS-Basics

**Change directory:**

cd Course\_EKS-Basics

**Take a look at the deployment file:**

cat nginx-deployment.yaml

**Take a look at the service file:**

cat nginx-svc.yaml

**Check its status:**

kubectl get service

**Create the service:**

kubectl apply -f ./nginx-svc.yaml

kubectl get service

**Create the deployment:**

kubectl apply -f ./nginx-deployment.yaml

kubectl get deployment

*You will get number of nodes provisioned and all should be in ready state*

curl "a9281fa3ccafd4c6192bd4cb0ff32626-1115102941.us-east-1.elb.amazonaws.com"

***Copy the external IP of the load balancer, and paste it into a text file, as we'll need it in a minute.***

**View the pods: (to see the list of pods provisioned)**

kubectl get pod

**View the ReplicaSets: (you will all replica set running)**

kubectl get rs

**View the nodes: (all three nodes running)**

kubectl get node

**How do we access the application? We can access our application using the load balancer service which expose our application to the internet, from the command output:**

kubectl get service

**Access the application using the load balancer, replacing** <LOAD\_BALANCER\_EXTERNAL\_IP> with the IP you copied earlier:

curl "<LOAD\_BALANCER\_EXTERNAL\_IP>"

* The output should be the HTML for a default Nginx web page.
* In a new browser tab, navigate to the same IP, where we should then see the same Nginx web page.

**After all the configurations, we need to test our EKS cluster for scalability and high-availability by shutting down our worker-nodes.**

* Test the High Availability Features of Your EKS Cluster
* In the AWS console, on the EC2 instances page, select the three t3.micro instances.
* Click Actions > Instance State > Stop.
* In the dialog, click Yes, Stop.
* After a few minutes, we should see EKS launching new instances to keep our service running.
* In the CLI, check the status of our admin cluster node:

kubectl get node

**All the nodes should be down (i.e., display a NotReady status).**

**Check the pods:**

kubectl get pod

*We'll see a few different statuses — Terminating, Running, and Pending — because, as the instances shut down, EKS is trying to restart the pods.*

**Check the nodes again:**

kubectl get node

*We should see a new node, which we can identify by its age.*

*Wait a few minutes, and then check the nodes again:*

**kubectl get node**

We should have one in a Ready state.

**Check the pods again:**

kubectl get pod

**We should see a couple pods are now running as well.**

**Check the service status:**

kubectl get service

***Copy the external IP listed in the output.***

**Access the application using the load balancer, replacing** <LOAD\_BALANCER\_EXTERNAL\_IP> with the IP you just copied:

curl "<LOAD\_BALANCER\_EXTERNAL\_IP>"

We should see the Nginx web page HTML again. (If you don't, wait a few more minutes.)

In a new browser tab, navigate to the same IP, where we should again see the Nginx web page.

**In the CLI, delete everything:**

eksctl delete cluster <cluster\_name>

eksctl delete cluster dev-cluster

**Conclusion**

**This is the basic of Kubernetes and AWS EKS . you have been able to do the followings:**

* Build a highly available EKS cluster using command line tool
* You created deployment and a pod and simple application
* Exposed your application to the internet
* You practiced how easy it is to get started with Kubernetes using EKS

----------------------------

**Configuration Files**

----------------------------

----------------------------

***nginx-deployment.yaml***

----------------------------

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

env: dev

spec:

replicas: 3

selector:

matchLabels:

env: dev

template:

metadata:

labels:

env: dev

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

-------------------

***nginx-svc.yaml***

-------------------

apiVersion: v1

kind: Service

metadata:

name: nginx-svc

labels:

env: dev

spec:

type: LoadBalancer

ports:

- port: 80

selector:

env: dev

# Create EKS Cluster & Node Groups

## Step-00: Introduction

* Understand about EKS Core Objects
  + Control Plane
  + Worker Nodes & Node Groups
  + Fargate Profiles
  + VPC
* Create EKS Cluster
* Associate EKS Cluster to IAM OIDC Provider
* Create EKS Node Groups
* Verify Cluster, Node Groups, EC2 Instances, IAM Policies and Node Groups

## Step-01: Create EKS Cluster using eksctl

* It will take 15 to 20 minutes to create the Cluster Control Plane

# Create Cluster

eksctl create cluster --name=eksdemo1 \

--region=us-east-1 \

--zones=us-east-1a,us-east-1b \

--without-nodegroup

# Get List of clusters

eksctl get clusters

## Step-02: Create & Associate IAM OIDC Provider for our EKS Cluster

* To enable and use AWS IAM roles for Kubernetes service accounts on our EKS cluster, we must create & associate OIDC identity provider.
* To do so using eksctl we can use the below command.
* Use latest eksctl version (as on today the latest version is 0.21.0)

# Template

eksctl utils associate-iam-oidc-provider \

--region region-code \

--cluster <cluter-name> \

--approve

# Replace with region & cluster name

eksctl utils associate-iam-oidc-provider \

--region us-east-1 \

--cluster eksdemo1 \

--approve

## Step-03: Create EC2 Keypair

* Create a new EC2 Keypair with name as kube-demo
* This keypair we will use it when creating the EKS NodeGroup.
* This will help us to login to the EKS Worker Nodes using Terminal.

## Step-04: Create Node Group with additional Add-Ons in Public Subnets

* These add-ons will create the respective IAM policies for us automatically within our Node Group role.

# Create Public Node Group

eksctl create nodegroup --cluster=eksdemo1 \

--region=us-east-1 \

--name=eksdemo1-ng-public1 \

--node-type=t3.medium \

--nodes=3 \

--nodes-min=2 \

--nodes-max=4 \

--node-volume-size=20 \

--ssh-access \

--ssh-public-key=kube-demo \

--managed \

--asg-access \

--external-dns-access \

--full-ecr-access \

--appmesh-access \

--alb-ingress-access

## Step-05: Verify Cluster & Nodes

### Verify NodeGroup subnets to confirm EC2 Instances are in Public Subnet

* Verify the node group subnet to ensure it created in public subnets
  + Go to Services -> EKS -> eksdemo -> eksdemo1-ng1-public
  + Click on Associated subnet in **Details** tab
  + Click on **Route Table** Tab.
  + We should see that internet route via Internet Gateway (0.0.0.0/0 -> igw-xxxxxxxx)

### Verify Cluster, NodeGroup in EKS Management Console

* Go to Services -> Elastic Kubernetes Service -> eksdemo1

### List Worker Nodes

# List EKS clusters

eksctl get cluster

# List NodeGroups in a cluster

eksctl get nodegroup --cluster=<clusterName>

eksctl get nodegroup --cluster=eksdemo1

# List Nodes in current kubernetes cluster

kubectl get nodes -o wide

# Our kubectl context should be automatically changed to new cluster

kubectl config view --minify

### Verify Worker Node IAM Role and list of Policies

* Go to Services -> EC2 -> Worker Nodes
* Click on **IAM Role associated to EC2 Worker Nodes**

### Verify Security Group Associated to Worker Nodes

* Go to Services -> EC2 -> Worker Nodes
* Click on **Security Group** associated to EC2 Instance which contains remote in the name.

### Verify CloudFormation Stacks

* Verify Control Plane Stack & Events
* Verify NodeGroup Stack & Events

### Login to Worker Node using Keypai kube-demo

* Login to worker node

# For MAC or Linux or Windows10

ssh -i kube-demo.pem ec2-user@<Public-IP-of-Worker-Node>

# For Windows 7

Use putty

## Step-06: Update Worker Nodes Security Group to allow all traffic

* We need to allow All Traffic on worker node security group

## Additional References

* <https://docs.aws.amazon.com/eks/latest/userguide/enable-iam-roles-for-service-accounts.html>
* <https://docs.aws.amazon.com/eks/latest/userguide/create-service-account-iam-policy-and-role.html>

# Kubernetes - PODs

## Step-01: PODs Introduction

* What is a POD ?
* What is a Multi-Container POD?

## Step-02: PODs Demo

### Get Worker Nodes Status

* Verify if kubernetes worker nodes are ready.

# Get Worker Node Status

kubectl get nodes

# Get Worker Node Status with wide option

kubectl get nodes -o wide

### Create a Pod

* Create a Pod

# Template

kubectl run <desired-pod-name> --image <Container-Image> --generator=run-pod/v1

# Replace Pod Name, Container Image

kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0 --generator=run-pod/v1

* **Important Note:** Without **--generator=run-pod/v1** it will create a pod with a deployment which is another core kubernetes concept which we will learn in next few minutes.
* **Important Note:**
  + With **Kubernetes 1.18 version**, there is lot clean-up to **kubectl run** command.
  + The below will suffice to create a Pod as a pod without creating deployment. We dont need to add **--generator=run-pod/v1**

kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0

### List Pods

* Get the list of pods

# List Pods

kubectl get pods

# Alias name for pods is po

kubectl get po

### List Pods with wide option

* List pods with wide option which also provide Node information on which Pod is running

kubectl get pods -o wide

### What happened in the backgroup when above command is run?

1. Kubernetes created a pod
2. Pulled the docker image from docker hub
3. Created the container in the pod
4. Started the container present in the pod

### Describe Pod

* Describe the POD, primarily required during troubleshooting.
* Events shown will be of a great help during troubleshooting.

# To get list of pod names

kubectl get pods

# Describe the Pod

kubectl describe pod <Pod-Name>

kubectl describe pod my-first-pod

### Access Application

* Currently we can access this application only inside worker nodes.
* To access it externally, we need to create a **NodePort Service**.
* **Services** is one very very important concept in Kubernetes.

### Delete Pod

# To get list of pod names

kubectl get pods

# Delete Pod

kubectl delete pod <Pod-Name>

kubectl delete pod my-first-pod

## Step-03: NodePort Service Introduction

* What are Services in k8s?
* What is a NodePort Service?
* How it works?

## Step-04: Demo - Expose Pod with a Service

* Expose pod with a service (NodePort Service) to access the application externally (from internet)
* **Ports**
  + **port:** Port on which node port service listens in Kubernetes cluster internally
  + **targetPort:** We define container port here on which our application is running.
  + **NodePort:** Worker Node port on which we can access our application.

# Create a Pod

kubectl run <desired-pod-name> --image <Container-Image> --generator=run-pod/v1

kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0 --generator=run-pod/v1

# Expose Pod as a Service

kubectl expose pod <Pod-Name> --type=NodePort --port=80 --name=<Service-Name>

kubectl expose pod my-first-pod --type=NodePort --port=80 --name=my-first-service

# Get Service Info

kubectl get service

kubectl get svc

# Get Public IP of Worker Nodes

kubectl get nodes -o wide

* **Access the Application using Public IP**

http://<node1-public-ip>:<Node-Port>

* **Important Note about: target-port**
  + If target-port is not defined, by default and for convenience, the **targetPort** is set to the same value as the **port** field.

# Below command will fail when accessing the application, as service port (81) and container port (80) are different

kubectl expose pod my-first-pod --type=NodePort --port=81 --name=my-first-service2

# Expose Pod as a Service with Container Port (--taret-port)

kubectl expose pod my-first-pod --type=NodePort --port=81 --target-port=80 --name=my-first-service3

# Get Service Info

kubectl get service

kubectl get svc

# Get Public IP of Worker Nodes

kubectl get nodes -o wide

* **Access the Application using Public IP**

http://<node1-public-ip>:<Node-Port>

## Step-05: Interact with a Pod

### Verify Pod Logs

# Get Pod Name

kubectl get po

# Dump Pod logs

kubectl logs <pod-name>

kubectl logs my-first-pod

# Stream pod logs with -f option and access application to see logs

kubectl logs <pod-name>

kubectl logs -f my-first-pod

* **Important Notes**
  + Refer below link and search for **Interacting with running Pods** for additional log options
  + Troubleshooting skills are very important. So please go through all logging options available and master them.
  + **Reference:** <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

### Connect to Container in a POD

* **Connect to a Container in POD and execute commands**

# Connect to Nginx Container in a POD

kubectl exec -it <pod-name> -- /bin/bash

kubectl exec -it my-first-pod -- /bin/bash

# Execute some commands in Nginx container

ls

cd /usr/share/nginx/html

cat index.html

exit

* **Running individual commands in a Container**

kubectl exec -it <pod-name> env

# Sample Commands

kubectl exec -it my-first-pod env

kubectl exec -it my-first-pod ls

kubectl exec -it my-first-pod cat /usr/share/nginx/html/index.html

## Step-06: Get YAML Output of Pod & Service

### Get YAML Output

# Get pod definition YAML output

kubectl get pod my-first-pod -o yaml

# Get service definition YAML output

kubectl get service my-first-service -o yaml

## Step-07: Clean-Up

# Get all Objects in default namespace

kubectl get all

# Delete Services

kubectl delete svc my-first-service

kubectl delete svc my-first-service2

kubectl delete svc my-first-service3

# Delete Pod

kubectl delete pod my-first-pod

# Get all Objects in default namespace

kubectl get all

KUBERNETS ARCHITECTURE

Kubernetes have the master and nodes, the clients or workers are called as nodes in lubernetes.

Kubernets master contains many components, starting with the API Server, API server is the interface to the outside world, so anyone who wants to talk to the API in kubernets environments have to go through the API server. It could be the users, it could be the Kubernetes dashboard or any 3rd party applications, lets say Openshift sit on top of Kubernetes, so it has to go through the API server in order to leverage the Kubernetes functionalities.

API server coordinates with all the other components and then get things done

The second component which is also important is the Scheduler. When

The next component is the controller manager, controllers are the ones responsible for providing the application deployment the way the containers are launched, the way they scale and the way they are deployed the new versions of it and so on. And there are many different controllers which are available inside Kubernetes, so when you want to run those multiple replicas and run your applications with high availability, it is the controller like deployment which provides you with that. The replication scaling and the strategy to go from one version to another is all defined and controlled by the controllers, and then there are different controllers that Kubernetes has for different types of workloads. Let’s have a look at those.

When you want to run those stateless applications, typically your applications servers and so on with high availability, you would use a controller by name Deployment. When you want to run agents like monitoring agents, you want to run something on every nodes in your cluster, you will use something called the daemon set, daemon-set guarantees that you run one node, one instance of that application on every nodes.

When you want to run stateful applications like databases and so on, you where you need persistent storage and certain way of deploying those and so on, you would use stateful sets to run adhoc jobs, you would use the controller called Jobs which is run to completion and when you want to run cron jobs, there is a controller for it too.

Even though there are different controllers, all of those are compiled into one single binary and that is the controller manager component that you see running on your Kubernetes master, so all of these are combined into the Controller Manager as mentioned in the architecture earlier.

We have talked about API Server, Scheduler, Controller Manager, now let’s look at one of the most important aspects and the component of Kubernetes, the brain of Kubernetes that is the etcd, it’s not really the brain because it’s more like a storage. The intelligence is provided by controllers, schedulers and so on, but the state of the cluster is stored in this database, it’s a key-value pair database called etcd. It’s a key-value pair and the entire state so any configuration changes that you make, any object that you create are first stored in the etcd, so when you talk to the API Server, it actually make an entry in the etcd server first and then when you want to set up your control plane with high availability, the Kubernetes master with high availability, first you go and set up a cluster etcd with minimum of three nodes and that is the one which is going to give you the high availability and then you can talk to any etcd node and these etcd nodes talk to each other with a protocol called RAFT. Raft is a distributed consensus protocol that is needed because any configuration that you store in any of the etcd nodes has to be propagated and stored in all of these nodes, that is how you get high availability.

Now, the fault tolerance formula is 2n + 1, that is why we need three or five and so on, because if you want to tolerate one node failure, you need two into one plus one that is three nodes of etcd, if you want to tolerate 2 nodes failure, you need 5 nodes of etcd and if you want to tolerate 3 node failure of etcd, you need 7. Typically, you would create maximum seven etcd because there is a lot of data which travels between etcd for every change, every commit, every updates which happens in Kubernetes cluster, so anything beyond 7 makes the cluster slower itself and for high availability etcd is the most important component here.

Lets look at what components run on the nodes, every nodes which participates in the cluster which run as a worker node or which is responsible for running containers runs process called kubelet. Kubelet is sort of the agent that Kubernetes cluster and that’s the one which talk to the API server and then talk to the runtime and get the job done. A runtime could be your typical container daemons like Docker for example. Docker is the most popular one but Kubernetes works with any these container tool, Kubernetes can work with rocket or cryo which is the newer closely integrated runtime for Kubernetes, and that’s how the containers are launched and managed by kubelet.

Kubelet is responsible to talk to the API server and then get the work done locally, also do the health checks and so on, and that’s what you run on every single nodes that participate in the cluster, every nodes that runs on inside the cluster has a kubelet, has the docker runtime, and then there is one more service which is called as kube proxy, but this will be discussed more when we are talking about Kubernetes services.

So, this is how it works, when you as a user connect to the API server on the master and run let’s say kubectl run or kubectl apply, it actually get scheduled, the scheduler decide which node its going to run on and that nodes kubelet is then informed about the new container to be run on it and then, the kubelet will talk to the local container runtime, whatever is configured with the kubelet, that is most likely, it is going to be docker or any other runtime that Kubernetes support, so kubelet talks to the runtime and ask it to run the actual container, and that’s how your containers are launched in the Kubernetes environment on those nodes, kubelet then keeps on doing probes. Probes are way to do the health checks and there are couple of probes that are available and that’s how kubelet decides whether the containers are available or not and it keeps on monitoring for it and then it inform back to the master if there are any issues with the containers.

Ans this is how the communication between the API server, the kubelet and the container runtime happens. Even on the master, you will typically see those client services running, that include kubelet, docker and kube proxy because most of the control plane services, controller manager, etcd, API server, scheduler are run as containers as well can be run as containers. In addition to this, there are certain add-ons which even though the are called as add-ons earlier have become an essential part of Kubernetes environment, that includes the DNS service, which you will see running inside your cluster, possibly multiple instances of DNS server, second is the CNI, that is for the networking, you will essentially need to network your containers running on different hosts and for that, you would have to set up those SNI plug-in components such as Flannel, Weave, Calico and so on.