**KUBERNETES**

**Master-Node communication**

1.We need to have the master servers and node servres

2.Add the docker packages to the master server ass well as the node server

3.Add the kunerntes packages to the master and node server in the cluster

4.we need to install the kubeadm,kubectl,kubelet,docker Engine

5.Enable net.bridge.bridge-nf-call-iptables on all three nodes.

**Initializing the cluster**

1.On only the Kube Master server, initialize the cluster and configure kubectl.

2. Install the flannel networking plugin in the cluster by running this command on the Kube Master server.

3. The kubeadm init command that you ran on the master should output a kubeadm join command containing a token and hash. You will need to copy that command from the master and run it on both worker nodes with sudo.

4. Now you are ready to verify that the cluster is up and running. On the Kube Master server, check the list of nodes.

5.Then we can able check the cluster by using the kubectl get command

**Kubernetes API Primitives & Cluster architecture**

1. Kubernetes Objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of the cluster.
2. What containerized applications are running (and on which nodes)
3. The resources available to those applications
4. The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance
5. A Kubernetes object is a “record of intent”–once you create the object, the Kubernetes system will constantly work to ensure that object exists.

**Kubernetes Services**

1.A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.

2. The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

3. Applications that run to completion inside a pod are called "jobs."  This is useful for doing batch processing.

4.Most Kubernetes objects are created using yaml. Here is some sample yaml for a job which uses perl to calculate pi to 2000 digits and then stops.

5.We need to write the kubernetes service file and apply to the kubapi server by using the kubectl service apply –f filename

**Network Primitives**

1.Create this yaml file on your master node and call it "pi-job.yaml". Run the job with the command:

kubectl create -f pi-job.yaml

2. Check the status of the job using the kubectl describe command.

3. When the job is complete, view the results by using the kubectl logs command on the appropriate pod.

4. Write yaml for a new job.  Use the image "busybox" and have it sleep for 10 seconds and then complete.  Run your job to be sure it works.

5.Pods usually represent running applications in a Kubernetes cluster.

When we are using the yaml files we will use the kind like pod,service,Replication

Controller and node and files.

**Secure kunberntes cluster**

1.You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster.

2.As Kubernetes is entirely API driven, controlling and limiting who can access the cluster and what actions they are allowed to perform is the first line of defense.

3.Kubernetes expects that all API communication in the cluster is encrypted by default with TLS

4.TLS means Transport layer security used to give the certificates to the network api request’s

5. All API clients must be authenticated, even those that are part of the infrastructure like nodes, proxies, the scheduler, and volume plugins.

**Managing kubernetes Highly available.**

1.Creating Highly Available Clusters with kubeadm

2. There are two types of ways to make the kubernetes HA

1. With stacked control plane nodes.
2. With an external etcd cluster.

For both methods you need this infrastructure:

* Three machines that meet [kubeadm’s minimum requirements](https://kubernetes.io/docs/setup/independent/install-kubeadm/#before-you-begin) for the masters
* Three machines that meet [kubeadm’s minimum requirements](https://kubernetes.io/docs/setup/independent/install-kubeadm/#before-you-begin) for the workers
* Full network connectivity between all machines in the cluster (public or private network)
* sudo privileges on all machines
* SSH access from one device to all nodes in the system
* **kubeadm** and **kubelet** installed on all machines. **kubectl** is optional.

3.With stacked control plane nodes. This approach requires less infrastructure. The etcd members and control plane nodes are co-located.

4.With an external etcd cluster. This approach requires more infrastructure. The control plane nodes and etcd members are separated.

5.Your clusters must run Kubernetes version 1.12 or later. You should also be aware that setting up HA clusters with kubeadm is still experimental and will be further simplified in future versions.

**Create a kube-apiserver load balancer with a name that resolves to DNS.**

1. In a cloud environment you should place your control plane nodes behind a TCP forwarding load balancer.
2. This load balancer distributes traffic to all healthy control plane nodes in its target list. The health check for an apiserver is a TCP check on the port the kube-apiserver listens on (default value :6443).
3. It is not recommended to use an IP address directly in a cloud environment.
4. The load balancer must be able to communicate with all control plane nodes on the apiserver port.
5. It must also allow incoming traffic on its listening port.

**HAProxy can be used as a load balancer.**

1. Make sure the address of the load balancer always matches the address of kubeadm’s ControlPlaneEndpoint.
2. Add the first control plane nodes to the load balancer and test the connection:
3. nc -v LOAD\_BALANCER\_IP PORT
4. A connection refused error is expected because the apiserver is not yet running. A timeout, however, means the load balancer cannot communicate with the control plane node. If a timeout occurs, reconfigure the load balancer to communicate with the control plane node.
5. Add the remaining control plane nodes to the load balancer target group.

**When you SSH to any node, make sure to add the -A flag:**

STEP1:Master node should be reliable.

STEP2:Storage layer

STEP3:Replicated API services

STEP4:Controllers and scheduled Daemons

STEP5: SSH between nodes to check that the connection is working correctly.

**End-to-End Testing and Validations**

1. End-to-end (e2e) tests for Kubernetes provide a mechanism to test end-to-end behavior of the system, and is the last signal to ensure end user operations match developer specifications.
2. unit and integration tests provide a good signal, in a distributed system like Kubernetes it is not uncommon that a minor change may pass all unit and integration tests, but cause unforeseen changes at the system level.
3. The primary objectives of the e2e tests are to ensure a consistent and reliable behavior of the kubernetes code base, and to catch hard-to-test bugs before users do, when unit and integration tests are insufficient.
4. End-to-end tests will pass on a properly running CDK cluster outside of bugs in the tests.To deploy the end-to-end test suite, you need to relate the **kubernetes-e2e** charm to your existing kubernetes-master nodes
5. Command for the deployment is

juju deploy cs:~containers/kubernetes-e2e

juju run-action kubernetes-e2e/0 test

**juju actions kubernetes-e2e --format=yaml –schema//Tuning the testing**

**Validating Nodes & the Cluster**

**1.**We need to login into the master machine and enter the kubectl get nodes.

Command –

2. kubectl get nodes –o wide

3.kubernetes describe node node-name.com

4.By using the above command we can see the description consist in the node server.

5.We can also see the which kube is running in the node server by  ps aux | grep kube

**Application Lifecycle Management**

**Deployment, Roll updates and Rollbacks**

1.Deployment files can be used to the  deploy the files of an object to the kubeapi server

Command—     kubectl create deployment –f filename

        Kubectl get deployments

2.Roll updates

1. 1.kubectl set image deployment/nginx-deployment nginx:nginx1.9
2. 2.kubectl rolling-update frontend-v1 frontend-v2 –image=image:v2
3. 3.You can see the status of the rolling updates in the deployment file using the  following command
4. 4.Kubectl rollout status deployment/nginx-deployment
5. 5.Kubectl describe deployment/nginx-deployment

**How Kubernetes Configures Applications**

1. specify the latest stable API version.
2. When defining configurations, specify the latest stable API version.
3. Write your configuration files using YAML rather than JSON. Though these formats can be used interchangeably in almost all scenarios, YAML tends to be more user-friendly.
4. Group related objects into a single file whenever it makes sense. One file is often easier to manage than several. See the guestbook-all-in-one.yaml file as an example of this syntax.
5. Note also that many kubectl commands can be called on a directory. For example, you can call kubectl create on a directory of config files.

Config maps mainly used to define the environmental variables by using the following commands

Kubectl create configmap my-mp –from-literal=school=LinuxAcademy

You can verify the config maps created by using the following command

Kubectl get configmaps

Kubectl describe configmaps my-map

**Scaling Applications**

1. In kubernetes scaling  we have two different types one is horizontal autosccaling and vertical autoscaling
2. In the HPA the pods  will be created to handle the requests from the users.
3. In the VPA the pods will additional recourses to make the cluster healthy.
4. In the above example we have two replicas so we need to change the replicas by using the cli
5. Command for changing the replicas is

Kubectl scale deployment nginx-deployment –replicas:4

**Self-Healing Applications**

1. 1.In this lesson they will  show you how applications in Kubernetes are self-healing because of the way Kubernetes constantly monitors the cluster and compares it with the specifications.
2. 2.In general, Pods do not disappear until someone destroys them. This might be a human or a controller.
3. 3.The only exception to this rule is that Pods with a **phase** of Succeeded or Failed for more than some duration (determined by terminated-pod-gc-threshold in the master) will expire and be automatically destroyed.
4. Use a [Job](https://kubernetes.io/docs/concepts/jobs/run-to-completion-finite-workloads/) for Pods that are expected to terminate, for example, batch computations. Jobs are appropriate only for Pods with restartPolicy equal to OnFailure or Never.
5. Use a [ReplicationController](https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/), [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/), or [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) for Pods that are not expected to terminate, for example, web servers. ReplicationControllers are appropriate only for Pods with a restartPolicy of Always.

Use a [DaemonSet](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/) for Pods that need to run one per machine, because they provide a machine-specific system service.

**Labels & Selectors**

1.Labels are mainly used to configure the certain nodes and as well as the pods

2.By using the labels we can able to schedule the application to run on which pods and to run the pods on which nodes .

3.command to create the labels for pods and nodes inside the cluster

Kubectl get pods –l app=nginx

Kubectl label pod  pod-name sravan=kumar

Kubectl describe –l sravan=kumar

4. The set of pods that a service targets is defined with a label selector. Similarly, the population of pods that a replicationcontroller should manage is also defined with a label selector.

5. Labels selectors for both objects are defined in json or yaml files using maps, and only equality-based requirement selectors are supported

**DaemonSets**

1.A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.

2. Deleting a DaemonSet will clean up the Pods it created.

3.Some typical uses of a DaemonSet are:

4.running a cluster storage daemon, such as glusterd, ceph, on each node.

running a logs collection daemon on every node, such as fluentd or logstash.

5.running a node monitoring daemon on every node, such as Prometheus Node Exporter, collectd, Dynatrace OneAgent, Datadog agent, New Relic agent, Ganglia gmond or Instana agent.

* Kubectl get daemonsets –n kube-system
* Kubectl describe daemonsets kube-flannel-ds –n kube-system

**Resource Limits & Pod Scheduling**

1.In this lesson, we'll discuss how to set limits and how pods get scheduled based on their needs.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes.

2.One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

You add a taint to a node using kubectl taint. For example,

3.kubectl taint nodes node\_name.com node.role.kubernetes.io/master

4.when you entering the command in the cli it will untainted from the cluster

5.kubectl taint nodes node\_name.com node.role.kubernetes.io=master:NoSchedule

**Manually Scheduling Pods**

1.Manually scheduling pods can be possible by using the labes and selectors

Kubectl node node\_name –l app=nginx(key,value pairs)

2.Generally such constraints are unnecessary, as the scheduler will automatically do a reasonable placement (e.g. spread your pods across nodes, not place the pod on a node with insufficient free resources, etc.) but there are some circumstances where you may want more control on a node where a pod lands.

3.We need  to give the key=value pair to the node by using the following commands

Kubectl label node node\_name.com key=value

4.We can see the label is attached to the node or not by using the following command

Kubectl describe node ncode\_name.com

5.wen you writing the manifest file and apply that file to the kubeapi server in that time if you mentioned any labels to certain nodes and pods that will be effected by using the slectors inside the manifest files.

**Monitoring Cluster and Application Components**

1.To scale an application and provide a reliable service, you need to understand how the application behaves when it is deployed.

2.You can examine application performance in a Kubernetes cluster by examining the containers, pods, services, and the characteristics of the overall cluster.

3Kubernetes provides detailed information about an application’s resource usage at each of these levels.

4 This information allows you to evaluate your application’s performance and where bottlenecks can be removed to improve overall performance.

5.Heapster is the monitoring tool supports cluster wide aggregator of monitoring and event data.

[**Upgrading Kubernetes Components**](https://linuxacademy.com/cp/courses/lesson/course/1437/lesson/1/module/155)

1.You need to have a kubeadm Kubernetes cluster running version 1.12.0 or later. Swap must be disabled. The cluster should use a static control plane and etcd pods.

2.Make sure you read the release notes carefully.

3.Make sure to back up any important components, such as app-level state stored in a database. kubeadm upgrade does not touch your workloads, only components internal to Kubernetes, but backups are always a best practice

4.when we want to upgrade the nodes use the following commands to upgrade

**apt-mark unhold kubelet kubeadm && \**

**apt-get update && apt-get upgrade -y kubelet kubeadm && \**

**apt-mark hold kubelet kubeadm**

5.we should unhold first why because when we initializing the all the services inside the master server

we will hold the services by using the hold command

**kubeadm upgrade plan     |||| kubeadm upgrade apply v1.13.0**

**Upgrading the Underlying Operating System**

1.In this we can add the node by generating the joining key with hash tag

When we generating any token by using the kubeadm init –pod-network=10.10.0.0/16

2.In this time the token will be available upto the 24 hrs then it will automatically disappears.

3.Suppose you want to add a node server to the cluster then we need to generate the token by using the following command

Kubeadm token generate

Sudo kubeadm token create created\_token\_name –ttl 3h –print-join-command

4.In this we have the option to upgrade the underlying operating system that can be possible to update the nodes to the cluster by using the

5.Upgrading a cluster is a two-part process which involve upgrading a cluster's master and/or its nodes. You cannot upgrade both the cluster's master and nodes at the same time.

**Node Network Configuration**

1.By defaultly we have the following networks

TCP     6443\*    Kubernetes API Server

TCP     2379-2380   etcd server client API

TCP     10250    Kubelet API

TCP     10251    kube-scheduler

TCP     10252    kube-controller-manager

TCP     10255    Read-Only Kubelet API

TCP     10250    Kubelet API

TCP     10255    Read-Only Kubelet API

TCP     30000-32767 NodePort Services

**Service Networking**

1.A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

2.As an example, consider an image-processing backend which is running with 3 replicas. Those replicas are fungible - frontends do not care which backend they use.

3.While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that or keep track of the list of backends themselves.

4 Kubernetes offers a simple Endpoints API that is updated whenever the set of Pods in a Service changes. For non-native applications, Kubernetes offers a virtual-IP-based bridge to Services which redirects to the backend Pods.

5.the deployment file should applied to the cluster when we run the deployment file it will creates some pods inside the cluster then we need to map the pods to give access to the outside of the world

Kubectl expose deployment nginx-deployment --type=”NodePort”  --port 80

**Ingress & Egress**

1.An API object that manages external access to the services in a cluster, typically HTTP.

2.Ingress can provide load balancing, SSL termination and name-based virtual hosting.

3.Ingress, added in Kubernetes v1.1, exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/)within the cluster. Traffic routing is controlled by rules defined on the ingress resource.

**internet**

**|**

**[ Ingress ]**

**--|-----|--**

**[ Services ]**

4.An ingress can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, and offer name based virtual hosting. An [ingress controller](https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-controllers) is responsible for fulfilling the ingress

5.An ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type [Service.Type=NodePort](https://kubernetes.io/docs/concepts/services-networking/service/#nodeport) or [Service.Type=LoadBalancer](https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer).

**Deploying a Load Balancer**

1. To create an external load balancer, add the following line to your [service configuration file](https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer):
2. In the configuararation file we must write the type and as well as the kind –service

**kubectl expose rc example --port=8765 --target-port=9376 \**

**--name=example-service --type=LoadBalancer**

1. The above command will create the loadbalancer inside the cluster
2. When you want to check the service you can check by using the
3. **kubectl describe services example-service**

**Configure & Use Cluster DNS**

1.In the kubernetes we have a great feature called cluster DNS

We can see the dns by typing the following command

Kubectl get pods –n kube-system

2.The DNS will automatically creates when we creating the service or applying the service object ot the kube-api server.

3.For checking the DNS of the service just use the following command

Kubectl exec –it service –nslookup webhead

4.We can check logs of dns server whether it accessing or not?

5.commands

Kubectl  logs –n kube-system $(kubectl get pods –n kube-system –l kube-app=kube-dns –o name) –c kubedns

Kubectl  logs –n kube-system $(kubectl get pods –n kube-system –l kube-app=kube-dns –o name) –c dnsmask

Kubectl  logs –n kube-system $(kubectl get pods –n kube-system –l kube-app=kube-dns –o name) –c sidecar

**Container Network Interface (CNI)**

1.By default, Kubernetes on DC/OS uses DC/OS overlay network to launch kubernetes pods.

2.However, one can use some other network by selecting the network provider at the time of the kubernetes package installation.

3. The CNI plugin is selected by passing Kubelet the --network-plugin=cni command-line option. Kubelet reads a file from --cni-conf-dir (default /etc/cni/net.d) and uses the CNI configuration from that file to set up each pod’s network.

4. The CNI networking plugin supports hostPort. You can use the official [portmap](https://github.com/containernetworking/plugins/tree/master/plugins/meta/portmap) plugin offered by the CNI plugin team or use your own plugin with portMapping functionality.

5. If you want to enable hostPort support, you must specify portMappings capability in your cni-conf-dir

**Persistent Volumes**

1. Kubernetes has the capability to maintain the volumes and attached to the docker containers when any docker container get crashed
2. Managing storage is a distinct problem from managing compute. The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed. To do this we introduce two new API resources: PersistentVolume and PersistentVolumeClaim.

1. A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. We mostly use nfs server to configure.
2. A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a pod. Pods consume node resources and PVCs consume PV resources.
3. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

**Volumes & Their Access Modes**

1. Provisioning—In the provisioning we should do by two types one is static and dynamic
2. Static-- A cluster administrator creates a number of PVs. They carry the details of the real storage which is available for use by cluster users.
3. Dynamic-- When none of the static PVs the administrator created matches a user’s PersistentVolumeClaim, the cluster may try to dynamically provision a volume specially for the PVC.
4. Binding-- A user creates, or has already created in the case of dynamic provisioning, a PersistentVolumeClaim with a specific amount of storage requested and with certain access  modes.

         5.Using-- Pods use claims as volumes. The cluster inspects the claim to find the bound volume and mounts that volume for a pod. For volumes which support multiple access modes, the user specifies which mode is desired when using their claim as a volume in a pod.

**Authentication & Authorization**

A kubelet’s HTTPS endpoint exposes APIs which give access to data of varying sensitivity, and allow you to perform operations with varying levels of power on the node and within containers.

1. This document describes how to authenticate and authorize access to the kubelet’s HTTPS endpoint.
2. Transport layer security established.
3. Authentication (Authenticator modules)
4. Admission modules
5. Autherization

* ABAC
* RBAC
* Webhook

**Configure Network Policies**

1. A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
2. NetworkPolicy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.
3. By default, pods are non-isolated; they accept traffic from any source.
4. Pods become isolated by having a NetworkPolicy that selects them.
5. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy.

**TLS Certificates for Cluster Components**

1. Every Kubernetes cluster has a cluster root Certificate Authority (CA). The CA is generally used by cluster components to validate the API server’s certificate, by the API server to validate kubelet client certificates, etc. To support this, the CA certificate bundle is distributed to every node in the cluster and is distributed as a secret attached to default service accounts. Optionally, your workloads can use this CA to establish trust.

1. In this lesson we are going to initialize the tool is easy-rsa
2. Commands for initializing the tool is

1. Cd k8s/easy-rsa-master/easyrsa3/

./esayrsa init-pki

      5. We need to add the master into the environemtal vaibles before entering the foolowing command

./easyrsa –batch “—req-cn=${MASTER\_IP}@`date+%s`” build-ca nopass

**Securing Images**

In this lesson we learn about the securing the images of docker or rocket

When we deploying any containers to the cluster  in a private manner and update directly by using the sudo apt-get update

Key points:

* Regularly security updates to the environment
* Don’t run the apt-get update into the containers.
* Use rolling updates
* Ensure only authorized images used in the environment
* Use the private registry to store the images
* Ci pipeline should ensure that only veteed code is used for the building images.

**Troubleshooting**

1. In the troubleshooting session we mostly concentrate on the nodes and pods
2. For nodes kubectl get nodes –o wide
3. For pods kubectl get pods –o wide
4. In the troubleshooting we mainly focus on the nodes and pods if any error suspended during the troubleshooting

5.We should always give the private registries to the manifest file by using the docker private registries wa can able to give the security to the cluster.