**Kubernetes**

Architecture:

Node is a machine physical/virtual where Kubernetes is installed.

Node is a worker machine that’s where a container is launched.

Cluster : set of nodes grouped together

having multiple node shares loads aswell

Master: is responsible for managing cluster

With kubernetes installed init.

When we installed Kubernetes on the system we are actually installing following componenetd

* **API Serve**r - acts as frontend for kubernetes. The users, management devices, command interfaces all talk to API server tointeract with kubernetes cluster.
* **Etcd** - is a distributed , realiable key value stored used by kubernetes to store all data manage used to manage the cluster. Responsible for implementing logs within the cluster
* **Controller**. - brain behind orchestration. Responsible for noticing and responding when nodes, containers or a end points goes down. Controller make decision to bring new containers in such cases.
* **Container Runtime** - Underlying s/w used to run containers. Eg: docker, rkt, CRI-O
* **Kubelet.** - is the agent that runs on each node in the cluster. The agent is responsible for making sure the containers are running on the nodes as expected.
* **Scheduler** - responsible for distributing work or container across multiple nodes it looks for newly created contaienr and assign them nodes.

Scheduler will only decide which pod should go on which node.

Kubeproxy. Will handle network components within the pods

Master vs Worker Nodes

Worker Nodes : will have Container Runtime, kubelet agent,

Master Server : wiil have </> Kube-api server, etcd, contraller, scheduler

Command utiltiy :

**Kubectl** : is a tool used to deploy and manage application on kubernetes cluster, to get cluster information, status of other nodes in the cluster, manage many other thing

kubectl run : command is used to deploy application on cluster

kubectl cluster-info : To get the cluster information

kubectl get nodes : to list all the nodes part of cluster

Kubectl describe nodes. -

Kubernetes pods:

Containers are encapsulated into a Kubernetes object known as pod. Pod is a single instance of an application. Pod is the smallest object that we create in kubernetes

Pods usually have 1 to 1 relationship with container running our application

Don't add additional container to an existing pod to scale our application

Ex: Sinlge node Kubernetes cluster with a single instance of application running in single docker container encapsulated in a pod.

What if the number of user accessing application increases, we need to add additional instancec

of the web application to share the load, so do we spin up addition new instance do we bring up new containerinstance with in the same pod ? NO

We create new pod altogether with a new instances with same application. Running on two separate pod on the same kubernetes system or node.

Multi-container PODs:

A single pod can have multiple container expect for the fact that usually not multiple container of the same kind.

Eg: Helper container that will doing some supporitng task to the web application which requires to run of the same pod in that case we will have on the same pod. So two container can slo communicate directly by referring each other as localhost since they share the same network space plus they can easily share the same storage space as well.

**How to deploy pods ?**

**Kubectl run nginx -** it deployes a docker container by creating a pod., It first crreates a pod automatically and deployes an instance of nginx

**Where does this gets the application image from ?**

**- For that we need specify the image name with the image paramater**

**eg: kubectl run nginx --image nginx(**docker image to be used**)**

i.e application image (i.e nginx) is downloaded from the docker registry/hub

**How to see list of pods available ?**

**- kubectl get pods**

**kubectl run nginx**(name of the pod) **--image nginx(**docker image to be used**)**

while the pod name can be anything, but image name as to be the name of an image avilable

in docker hub or in any other container registry. We can additionally specify the tag for the image name .

To get more information on the pod:

**kubectl describe pod nginx**

To get additional status of pod

**kubectl get pods -o wide**

**YAML in Kubernetes :**

Kubernetes uses YAML files as inputs for the creation of objects such as pods, replicas, deployements etc . All of this follow similar structure.

Kubernetes definition file contains 4 top level fields and also these are required fields:

1. apiVersion - This is the version of the kubernetes api we are using to create the object.
2. kind. - Type of object that we create here refers to the "pod"
3. metadata. - Data about the objects ( name, labels etc ) [Name is a string, labels is a dictionary with the metadata it will have key value pair (app,type)]
4. Spec - Specification section(provide additional info pertaining to the object)

|  |  |
| --- | --- |
| Kind | Version |
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |

Eg: pod-defination.yml

apiVersion: v1

kind: Pod

metadata:

name: myapp-pod

labels:

app: myapp

type: front-end

spec:

containers:

- name: nginx-container

image: nginx

To Create a pod from the file:

kubectl create -f <filename9pod-defination.yml)>

or

kubectl apply -f <filename9pod-defination.yml)>

To see the created pod

kubectl get pods or kubectl describe pod <podname(my-pod)>

**Kubernetes Controller:**

Controller are the brain behind kubernetes. They are the process that monitor kubernetes object and respond accordingly.

**Replication Controller:**

What is replica and y do we need replica?

**High Availablity:**

- when we have single pod running the application and application crashes the user will not be able to access. To prevent users from loosing access from application. To have more than 1 instances at the same time that way if 1 fails we still have application running on the other one. Replication controller helps us run multiple instances of a single pod in the kubernetes cluster. Thus providing **High availablity** .

Even if have a single pod the Replication controller can help by automatically bringing up a new pod when the exisiting one fails. Thus the Replication controller ensure that the specified number of pods are running all the time even if its just one or hundred.

**Load balancing and scaling:**

Scenario : When the number of users increases we deploy a additonal pod to balance the load across the 2 pods. If the demand further increases and if we have to run out of resources on the first node, we could deploy additional pods across the other node in the cluster.

Replication controller spans across multiple pods in the cluster.

**Diff b/w Replication Controller and Replica set:**

1. Both have the same purpose but they are not the same.

2. Replication controller is the older technology that’s being replaced by Replica set

1. Replica set is the new recommended way to set up replication.

Text

Description automatically generated

kubectl create -f <filename(rc-defination.yml)

kubectl get replicationcontroller

kubectl get pods

**Replica set:**

There is One Major difference b/w Replica set and Replica Controller. Replica set requires a selector definition. The Selector section helps the replica set identify what pods fall under it.

And its has to be written in the form of "matchLabels". The matchLabel selector simply matches the label specified under it to the lables on the pods.

Text

Description automatically generated

kubectl create -f replicaset-defination.yml

kubectl get replicaset

kubectl delete replicaset myapp-replicaset

kubectl get pods

**Labels and Selectors:**

Role of Replica set is to monitor the pod and if any of them fail then deploy new one. In fact it is process which monitors the pod.

How does replicaset knows what pods to moniotor. ?

* There can be hundreds/thousands of other pods in the cluster running different applications this is where labeling out pods during the creation comes in handy. We could now provide this label as a filter for replica set. Under the selector section we use the "matchLables" filter and provide the same label that we use while creating the pod. This way replicaset knows which pods to monitor.

**Scale:**

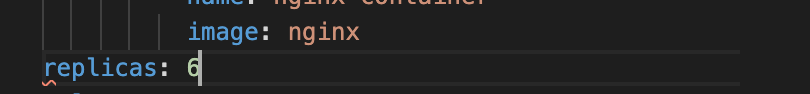
How we scale replica set ?

- Say we started with 3 replicas and futture we decided to scale to 6 howdo we update replicateset to 6 replicas

There are multiple ways:

**To update replaceset and run**

**1.** To update in the definition file to 6.



kubectl replace -f replicaset-defination.yml

2. kubectl scale --replicas=6 -f replicaset-defination.yml. ( in this it will not update on the file)

**Replicaset** Commands:

* Kubectl create -f replicaset-defination.yml ( To create replicaset)
* Kubectl get replicaset (to get the list of replicaset
* Kubectl delete replicaset <filename> (to delete the replicaset)
* Kubectl replace -f <filename>. ( To update the replicaset)
* Klubectl scale --replicas=6 -f <filename>. (to update the replicaset without modifing the file)

**Deployment:**

To upgrade application one after the other is known as Rolling upgrade.

To Rollback the changes made

The deployment provides us the capability to upgrade underlying instances seemlessly using Rolling updates, undo changes and pause and resume changes as required.

Text

Description automatically generated

kubectl create -f deployment-definiation.yml

kubectl get deployments

kubectl get replicaset

kubectl get pods

kubectl get all

kubectl describe deployment <deploymentname>

**Rollout and Versioning:**

when we first create deployment it triggers a rollout . A new rollout creates a new deployment reversion lets call it revision1.

In the future when the application is upgraded meaning the container version is updated to a new one a new rollout is triggered and a new deployment revision is created name revision 2 . This helps us keep track of the changes made to our deployment and enable us to rollback to previous versions of deployement if necessary

**To See the Status of rollout:**

kubectl rollout status deployment/myapp-deployment

**To see the Revision and history of the deployment**

kubectl rollout history deployment/myapp-deployment

**Deployment Strategy:**

1. One to destroy all the applicationa and then create newer version of application instances.

Meaning first destroy the 5 running instances and then deploy 5 new instances of new application versions. The problem is application will be down and inaccesiable to user . This strategy is known as **Recreate Strategy. And this is not the default strategy.**

1. The second startegy is where we don't destroy all of them at once instead we take down the older version and bring up the newer version one by one. This way the application will never goes down and upgrade is seemless. This is known as "**Rolling Update".** If we don’t specify the strategy while creating the deployment it will assume it to be "Rolling Update". i.e Rolling Update is the default deployment strategy

Text

Description automatically generated

**To apply the changes , New Rollout to trigger and new revision of deployement is created**

kubectl apply -f <deployementfilename>

Otherway: To update image of the application. But doing this way will resolve the deployment defination file having different configuration so much be careful when using the same defination file to make changes in the future

kubectl set image deployement/myapp-deployment nginx=nginx:1.9.1

To see the details:

kubectl describe myapp-deployment

**Upgrades:**

When a new deployemnt is created to deploy 5 replicas it first creates a replicaset automatically which in turn creates a number of pods required to meet the number of replicas

When we upgrade the application , the Kubernetes deployment object creates a new replica sets and starts creating containers there. At the same time taking down the pods in the old replica set following a rolling strategy. This can be seen when we try to list the replica set using the "kubectl get replicasets".

Once we upgrades the application later we realize something is wrong with the new version of the build used to upgrade so like to roll back the upgrade ,kubernets allow us to rollback to previous revision.

**To undo a Change (i.e Rollback)**

kubectl rollout undo deployement/myapp-deployment

The deployment will then destroy the pod in the new replica set and bring the older onces up in the old replica set. And application will be back to older format.

**Deployment Commands:**

* kubectl create -f deployement-defiantion.yml (To create the deployment)
* kubectl get deployments (To list the deployments)
* kubectl apply -f deployment-defination.yml (apply and set image command to update the deployments)
* kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1
* kubectl rollout status deployement/myapp-deployement (to see the status of rollouts)
* kubectl rollout history deployment/myapp-deployment
* kubectl rollout undo deployement/myapp-deployment (To rollback the deployment)

**Kubernetes Networking:**

In kubernetes IP address is assigend to a pod . Each pod in kubernetes gets its own internal IP address.

**Cluster Networking:**

**Criteria:**

1. All containers/PODS can communicate to once another without NAT

2. All nodes can communicate with all containers and vice-versa without NAT

Solutions are:

1. cisco

1. Vmvare
2. Vnet
3. Fannel
4. Cilium

**Kubernetes Servcies**

* Enable communication between varies components with in and outside of the application.
* Helps us to connect application with other application or the users

**Eg:**

**Node port :**

**Terms:**

**Target port is Pod port.**

**PORT is simply known as Service port (service infact a virtual server inside the node, inside the cluster it as its own IP address and that IP address is called Cluster Ip of the service)**

**The Port on the node itself which is used to access the webserver externally that is known as "Node Port"**

**Default range of node port: 30000 - 32767**

Text

Description automatically generated

**To Create a service:**

kubectl create -f service-defination.yml

**To see the created the services:**

kubectl get services

eg:

curl <http://192.168.1.2:30008> (IP of node and port of node)

Type of Services

1. NodePort : Service makes an internal port accessible on a port on the node

2. ClusterIP : Service creates a virtual ip inside the cluster to enable communication between different services such as set of front servers to set of backend servers

3. LoadBalancer: Provisions a load balancer for our application in supported cloud providers

When there are multiple pod in the same Node and have the same label , when the servcie is created it look for matching label and finds 3 of them , the service will automatically select all the 3 pods as end points to forward the external request. Coming form the user. We don’t have to do any additional configuartion to make this happen.

It uses a "Random" Algorithm , thus service acts a s built in load balancer to distributev load across different pods.

When there are multiple pods are distributed across multiple nodes, when we create w service which any additional configuration, kubernetes automatically creates a service that spans across all the nodes in a cluster and maps the target port to the same Node port on all the Node in the cluster. This way we can access the application using the IP of any node in the cluster and using the same port number.

Eg: curl <http://192.168.1.2:30008>

curl <http://192.168.1.3:30008>

curl <http://192.168.1.4:30008>

**Cluster IP: To communicate to internal application**

**Eg: from frontend to backend applications**

Text

Description automatically generated

Load balancer:

Eg: haproxy , nginx etc…

To host the application as (<http://example-vote.com>, <http://example-result.com>)

Text

Description automatically generated

**Microservices Architecture :**

**Important:**

* **kubeadm tool is used to bootstrap the cluster**

* **kubelet is the process that is responsible to manage pods and container on the node**

* **kubectl utility is the kuberneted command line**