|  |  |
| --- | --- |
| Kubernetes (@kubernetesio) / Twitter  KUBERNETES | Abstract  [Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]  Sourabh Gulhane |

**Kubernetes**

**What is Kubernetes?**

**Kubernetes Architecture**

Kubernetes Architecture consist of two components

* Master (Control Plane): its primary operation is to schedule a pod on node, monitor pod and nodes and implement changes on nodes.
* Slave (Worker Node): worker nodes report the status of nodes and pods to master node and continuously watches for new work assignments.

**Master components have 4 major components. They are**

* **API server**: API server is the most critical component in master sever where all of the components communicate with it including the external users as well, it primarily acts as a gatekeeper to the entire k8s cluster. For example, if a user wants to create, update, delete or even display any k8s objects, it has to go through the API server. So, whenever a user submits a YAML file to the API server, the API server will check for the authentication of that user and whether the user is authorized to perform the requested actions or not. So once that check is passed, API will validate the YAML configuration file. So, if everything looks good, then the API server will store all the data in persistent key value storage called ETCD.
* **ETCD:** It is a key value database where all the k8s stores, all its cluster configuration and app configuration secrets and other sensitive data, so it is crucial to secure a database from all different types of possible attacks.
* **Controller Manger:** It is called as the controllers of controllers, there are primarily 4 controllers behind this controller manager which are node controller, replication control and end point controller and service token and token controls. So at a high level, control manager is responsible for the overall health of entire k8s cluster.it ensure that nodes are up and running and the correct number of parts are running.
* **Schedule**r: The scheduler continuously watches the API server to check if any of the pods need to be scheduled onto the worker node. In case if there are any, then it finds the appropriate healthy worker node that matches the requirement of the Pod and then assign pod to respective worker node. If the scheduler does not find a suitable worker node that matches the pod requirement, then the pod will not be scheduled and marked as pending. So to put this in simple, the job of the scheduler is to find the appropriate worker node and assign it to a pod.

**Slave or worker node have below major Component:**

* **Pod**: it is the basic unit of scheduling in k8s cluster. It holds the container, and it has its own Ip address.
* **Kublet**: It is like an Kubernetes agent which runs on every node inside the k8s cluster. The primary job of the kublet is to watch the API server of master node for new work assignments to execute on that specific node. Also, the Kublet is responsible for reporting the status of worker nodes and pod running on that node to the API sever.
* **Kubeproxy**: It is responsible for maintaining entire network configuration. This network agent runs on every node inside the cluster, it plays a significant role in service configuration, routing and load distribution within the cluster.
* **Container Runtime Engines:** it is responsible for downloading and running container images on that node.
* **Kubectl**: It is a command line tool use to connect to k8s cluster.

**Basic Commands:**

1. Kubectl get nodes
2. Kubectl get nodes -o wide
3. kubeadm token create --print-join-command

**PODS**

Pods is basic unit of Scheduling in Kubernetes. Each pod can have one or more containers.

Container inside pod share same IP address, same volumes all pods can communicate with each other. Generally, we have one container per pod.

**Pod life cycle:**

we write the manifest file and submit it to API server. It will create POD using Manifest file. It will be in pending state until all containers is created. Once all containers inside Pod are up and running it will go to running state. when container task is completed Pod go to succeededstate. If any container inside Pod fails it will go to failed state.

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: nginx-pod1  labels:  app: nginx-label  tier: dev  spec:  containers:  - name: nginx-container  image: nginx  ports:  - containerPort: 80 |

**Commands:**

1. kubectl create -f pod.yaml
2. kubectl get pod
3. kubectl get pod nginx-pod -o wide
4. kubectl get pods -o yaml
5. kubectl describe pod nginx-pod
6. kubectl edit pod nginx-pod
7. kubectl exec -it nginx-pod -- /bin/bash
8. kubectl delete pod nginx-pod
9. kubectl delete pod.yaml
10. kubectl delete pod –all
11. kubectl delete –help
12. kubectl logs -nginx-pod
13. kubectl logs -l app=nginx --all-containers=true
14. 16)kubectl top pods
15. kubectl top pods --sort-by cpu
16. kubectl top pods --sort-by memory
17. kubectl top pods -n [namespace] sort-by cpu >cpu.txt
18. kubectl top pods -n [namespace] sort-by memory
19. kubectl top pods -all-namespace sort-by memory

**Replication Controller**

It ensure specific number ( mentioned in yaml file) of pods are always running. If there excess pods it gets killed and vice versa**.** Replication Controller and Pods are assosciated with Labels

Advantages of replication controller is High availability and load balancing

Replication controller is old and is replaced by replica set.

|  |
| --- |
| # nginxRc.yaml  apiVersion: v1  kind: ReplicationController  metadata:  name: rs-nginx-pod  labels:  app: frontend-rs  spec:  replicas: 2  selector:  app: rs-pod  template:  metadata:  name: rs-pod  labels:  app: rs-pod  spec:  containers:  - name: nginx-cont  image: nginx  ports:  - containerPort: 80 |

Commands

1. kubectl create -f rc.yaml
2. kubectl get rc
3. kubectl get pods
4. kubectl get rs -pod-nginx -o wide
5. kubectl describe rc rs-pod-niginx
6. kubectl logs rc/rs-nginx-pod -c nginx-cont (more than one container)
7. kubectl delete rc rs-nginx-pod
8. kubectl scale rc rs-nginx-pod --replicas=3
9. kubectl scale rc rs-nginx-pod --replicas=5

**Kubernetes Flags**

|  |  |
| --- | --- |
| **-f** | **force** |
| **-o** | **output** |
| **-l** | **label** |
| **--record** |  |
| **--all** | **All resources** |
| **--help** | **help** |
| **--port** |  |

|  |  |
| --- | --- |
| Object | Api version |
| Pod | v1 |
| Replication Controller | v1 |
| Replica Set | apps/v1 |

**Replica Set**

Difference between ReplicaSet and ReplicationController is that replicaset uses Set based selector where as replication controller select equity based selector.

**Set based Selector vs Equality based selector**

|  |  |
| --- | --- |
| **Set based Selector** | **Equality based selector** |
| Used in Replica set | **Used in Replication Controller** |
| **It has 3 operators**   1. **in** 2. **not in** 3. **exist** | **It has 3 operators**   1. **=** 2. **==** 3. **!=** |
| **Example**  selector:  environemnt in (production,qa)  tier notin (frontend,backend) | **Example**  selector:  environemnt=production  tier!=frontend |
| kubectl get pods -l environment in ( production ) | kubectl get pods -l environemnt=production |
| in set based we can select more than one value. |  |
| New Resources like RelicaSet, Jobs,Dameon set | Old resources  Replication Controller , Services |

|  |
| --- |
| **#Replicaset.yaml**  apiVersion: apps/v1  kind: ReplicaSet  metadata:  name: rspod  spec:  replicas: 3  selector:  matchLabels:  app: frontend  template:  metadata:  name: pod1  labels:  app: frontend  spec:  containers:  - name: cont1  image: nginx  ports:  - containerPort: 80 |

**Commands:**

1. Kubectl create -f replicaset.yaml
2. Kubectl get pods
3. Kubectl get rs
4. Kubectl describe rs <rs-name>
5. Kubectl edit rs <rs-name>
6. Kubectl scale rs <rs-name> --replicas=4
7. Kubectl delete rs <rs-name>
8. Kubectl get -f replicaset.yaml

**Deployments**

**Why we use deployment contorller.**

- scaling ( replicaset)

- rollout an rollback

- version 1 to version 2

- pause

- resume

- we can go back to version 1 from version 2

**Scaling** : application intsance to increase or decrease ( scalein or scale out)

rollout: ver1 to ver 2

rollback : ver2 to ver 1

#**Deployment Feature**

1) rollout

2) pause / resume

3) rollback

4) replica set self-healing and high availability

5) scale up and scale down

**Deployment strategy**

There are 4 type of deployment strategy are there

1)R**ecreate:**

it is dummy deployment using this we will completely shut down version 1 and then deploy version 2.

drawback: there is down time ver 1 to ver 2

2)**Rolling update**:

using this deployment strategy k8s will slowly rollout newer version by slowly updating the pods one after the other until all instances are rolled out. this is default deployment strategy in k8s.

but is slow process.

|  |
| --- |
| # nginx-deploy.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-deploy  labels:  app: nginx-app  spec:  replicas: 3  strategy:  type: RollingUpdate  RollingUpdate:  maxSurge: 2  maxUnavailable: 0  template:  metadata:  labels:  app: nginx-app  spec:  containers:  - name: nginx-container  image: nginx:1.7.9  ports:  - containerPort: 80  selector:  matchLabels:  app: nginx-app |

maxSurge: it will create 2 pod instances with newer version during upgrade

maxUnavailable: no of pod that can be unavailable during upgrade process

3) **Canary:**

we use canary deployment to test newer version of application before it is fully rolled out.

it is good for testing user can rule out if there is any issue before they fully use it,

4)**Blue green**

using this we deploy same no of newer app instance along with older version.

So the goal is to switch the user traffic from older version to your app instances, if there are any issues with the new version. It is very easy to switch back to the older version. we can remove the older version with the newer version is working as it should be.

It is a great option, but it is a very expensive option as well, because it requires a double resources such as CPU RAM.

**Commands:**

1. kubectl create -f nginx-deploy.yaml
2. kubectl create deployment imperativedeploy --image=nginx:1.9.1 --replicas=3
3. kubectl create deployment redis-deploy --image=redis --replicas=3 --dry-run=client -o yaml > redis-deploy.yaml (For dry-run: It tests to ensure were there any issues. Will NOT create the Object)
4. kubectl get deploy -l app=nginx-app
5. kubectl get rs -l app=nginx-app
6. kubectl get pods -l app=nginx-app
7. kubectl describe deploy nginx-deployment
8. kubectl set image deploy nginx-deploy nginx-container=nginx:1.91 --record
9. kubectl rollout status deployment/nginx-deploy
10. kubectl rollout history deployment/nginx-deploy
11. kubectl rollout undo deployment/nginx-deploy
12. kubectl rollout status deployment/nginx-deploy
13. kubectl describe deploy nginx-deploy | grep -i image

Update Version of "nginx:1.7.9" to "nginx:1.9.1"

1. kubectl set image deploy nginx-deploy nginx-container=nginx:1.9.1
2. kubectl edit deploy nginx-deploy
3. kubectl rollout status deployment/nginx-deploy
4. kubectl get deploy

Testing: Scale UP

1. kubectl scale deployment nginx-deploy --replicas=5
2. kubectl get deploy
3. kubectl get po -o wide

Testing: Scale DOWN

1. kubectl scale deployment nginx-deploy --replicas=3
2. kubectl get deploy
3. kubectl get po -o wide

Cleanup

1. kubectl delete -f nginx-deploy.yaml
2. kubectl get deploy
3. kubectl get rs
4. kubectl get po

**Namespace**

it is way of partitioning the Kubernetes cluster into multiple virtual partition.

u can partition it by Teams, application, environments etc.

by default, we have two namespaces

1) kube-system : it consist of Kubernetes system related pods

2) default

it allows us to create two identical resources with two identical name in namespaces

|  |
| --- |
| apiVersion: v1  kind: Namespace  metadata:  name: dev |
| apiVersion: v1  kind: Pod  metadata:  name: nginx-pods  namespace: dev  spec:  containers:  - image: nginx  name: nginx-pod |

**Imperial command:**

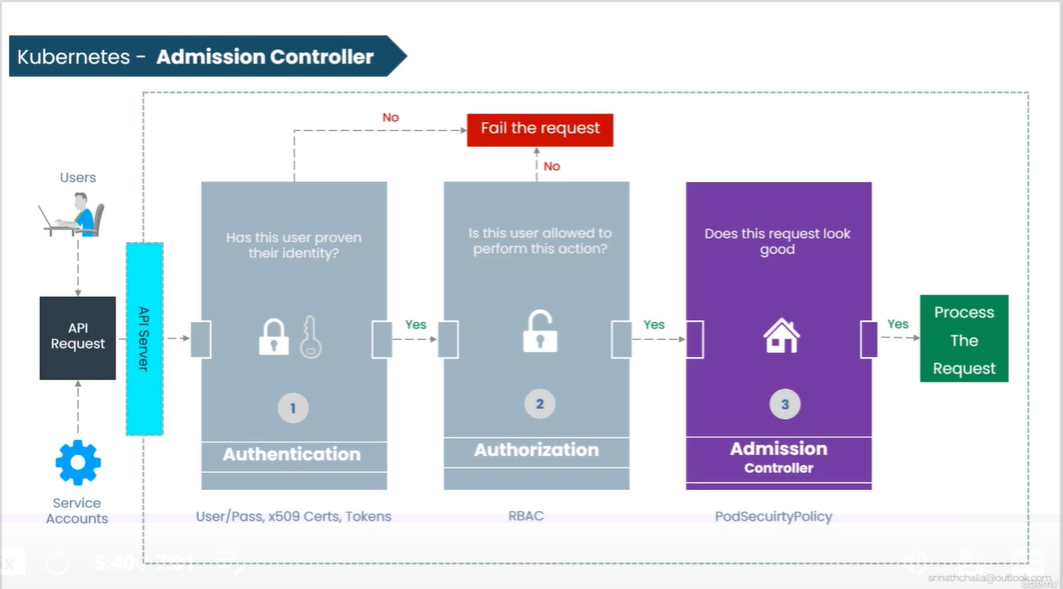
1. kubectl create namespace dev
2. kubectl apply -f namespace.yaml
3. kubectl get ns
4. kubectl get pod -n dev
5. kubectl get pod --namespace=dev
6. kubectl describe ns dev
7. kubectl config set-context -current --namespace=dev ( making dev as default namespace)
8. kubectl config view | grep namespace ( to check default namespace)
9. kubectl edit ns dev
10. kubectl delete ns dev
11. kubectl get pods -A ( pods in all namespace)
12. kubectl get [object-name] --all-namespaces

**Horizontal Scaling Vs vertical scaling**

Horizontal scaling means that you scale by adding more machines

into your pool of resources whereas Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.

**#3A of API server : Authorization, Authentication, Admission controller**



So before the app server, to process any API request, it needs to go through three main security challenges or stages

**1)Authentication** : API server will authenticate weather user sending API request is a from valid user or not. It will check identity if user.

If yes it passes to the next stage if not, request will be denied.

It has 3 types of authentication

1. basic authentication which contain username and password
2. x.509 certificates
3. tokens

**2)Authorization (RBAC):** it will check if the user has permission to use the specific object ( create , del).

**3)Admission controller**: here API server will check whether the request meets up the criteria to be consider as valid request.

It enables some advance feature like

We can limit compute resources per namespace

Pod security policy like pod run on specific user

**# ETCD backup and restore**

its a key value db.

it store the clustre configuration.

AKS ,GKE, EKS will take backup

we need to take backup if we use self managed cluster.

it has edcdctl command line tool.

**RBAC (Role Base Access Control)**

Set of action role can perform on resources.

Cluster role and cluster role binding ( cluster level)

role and role binding ( namespace space level)

#**role and role binding**

1) one role is can be link to one namespace

2) each role can have one or more rules.

3) each rolebinding have one or more user , service account or group but can refer to one role.

|  |
| --- |
| #role.yaml  apiVersion: rbac.authorization.k8s.io/v1  kind: Role  metadata:  name: deployment-manager  namespace: default  rules:  - apiGroups: ["apps" ,""]  resources: ["deployments", "pods", "replicasets"]  verbs: ["get","list","watch","create","update","patch","delete"] |
| #rolebinding.yaml  apiVersion: rbac.authorization.k8s.io/v1  kind: RoleBinding  metedata:  name: deployment-manager-binding  namespace: default  subject:  - kind: user  name: Jane  apiGroups: rbac.authorization.k8s.io  roleRef:  kind: Role  name: deployment-manager  apiGroups: rbac.authorization.k8s.io |

**# cluster role and cluster role binding**

this defines what action can be taken on which resources across all namespaces in the cluster.

|  |
| --- |
| # ClusterRole  kind: ClusterRole  apiVersion: rbac.authorization.k8s.io/v1  metadata:  name: clusterrole-monitoring  rules:  - apiGroups: [""]  resources: ["pods"]  verbs: ["get", "watch", "list"] |
| # ClusterRoleBinding  kind: ClusterRoleBinding  apiVersion: rbac.authorization.k8s.io/v1  metadata:  name: clusterrole-binding-monitoring  subjects:  - kind: User  name: appmonitor  apiGroup: rbac.authorization.k8s.io  roleRef:  kind: ClusterRole  name: clusterrole-monitoring  apiGroup: rbac.authorization.k8s.io |

ex - cluster admin , cluster monitor

**Node Selector:**

it used to select the node in which we want to run the pod.

for this first we need to assign Label to Node using below command

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: nginx-pod  namespace: dev  spec:  containers:  - image: nginx  name: nginx-pod  ports:  - containerPort: 80  nodeSelector:  disk: ssd |

**kubectl label node node-1 disk=ssd**

**kubectl get nodes –show-labels**

**kubectl create -f pod.yaml**

**Resource Limit**

CPU And Memory

**resource request** : it specify minimum amount of resource require to run the application

**resource limit:** it specify max amount of resource application can consume

CPU and RAM measurment

100m 0.1 core 1Mi 1024kb

1000m 1 core 100Mi 105mb

|  |
| --- |
| #Request limit Yaml  apiVersion: v1  kind: Pod  metadata:  name: fronted  spec:  containers:  - name: db  image: mysql  resource:  requests:  cpu: "250m"  memory: "100mi"  limits:  cpu: "500m"  memory: "256Mi" |

**Services**

Pod are ephemeral : they dies frequently , it has its own Ip address

services : services has stable Ip address , it provide load balancing as well

**Node port**

nodeport value : 30000- 32676

loadbalancer is an extension of nodeport service and node port service is an extension of cluster ip service

**NodePort Service YAML file:**

|  |
| --- |
| # nginx-svc-np.yaml  apiVersion: v1  kind: Service  metadata:  name: my-service  labels:  app: nginx-app  spec:  selector:  app: nginx-app  type: NodePort  ports:  - nodePort: 31111  port: 80  targetPort: 80 |

**Commands**

kubectl create –f nginx-deploy.yaml

kubectl create -f nginx-svc.yaml

kubectl get service -l app=nginx-app

kubectl get po -o wide

kubectl describe svc my-service

**Loadbalancer**

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: my-service  spec:  selector:  app: nginx-app  type: Loadbalancer  ports:  - nodePorts: 31000  port: 80  targetPort: 80 |

kubectl create -f nginx-service.yml

kubectl get ns -l app:nginx-app

kubectl describe ns my-service

**Cluster Ip**

it is a default service type in Kubernetes.

to access backend pod and connect them to frontend pod we use service type of cluster ip

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: my-service  spec:  selector:  app: nginx-app  type: ClusterIp  ports:  - nodePorts: 31000  port: 80  targetPort: 80 |

**Multiport:**

when we open multiple port in cluster Ip services.

|  |
| --- |
| ports:  - name : port1  protocol: tcp  port: 80  targetPort: 80  - name: port2  protocal: tcp  port:90  targetport: 90 |

**#Headless Service**

client want to communicate with 1 specefic pod

pods wants to talk directly with specific pod, not randomly selected

usecase: stateful application like database

|  |
| --- |
| spec:  clusterIP: None |

techworld with Nana 17:27

#**Demon set**

it will deploy one pod per node.i

it ensures all or some nodes have a copy of pods inside a cluster.

as nodes are added to the cluster pod are added to them.

as nodes are removed from the cluster pod are deleted

deliting a demon set delete the pod it created.

use case: monitoring application

|  |
| --- |
| # fluentd-ds-allnodes.yaml  apiVersion: apps/v1  kind: DaemonSet  metadata:  name: fluentd-ds  spec:  template:  metadata:  labels:  name: fluentd  spec:  containers:  - name: fluentd  image: gcr.io/google-containers/fluentd-elasticsearch:1.20  selector:  matchLabels:  name: fluentd |

1b. Create | Display | Validate

kubectl create -f fluentd-ds-allnodes.yaml

kubectl get po -o wide

kubectl get ds

kubectl describe ds fluentd-ds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Deploy Pod on "Subset" of worker nodes inside k8s cluster using DaemonSet

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

2a. Attach label to the nodes

kubectl get nodes

kubectl label nodes worker1 worker2 disktype=ssd

kubectl get nodes --show-labels

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

2b. YAML

|  |
| --- |
| # nginx-ds-subsetnodes.yaml  apiVersion: apps/v1  kind: DaemonSet  metadata:  name: nginx-ds  spec:  template:  metadata:  labels:  name: nginx  spec:  containers:  - name: nginx-container  image: nginx  nodeSelector:  disktype: ssd  selector:  matchLabels:  name: nginx |

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

2c. Create | Display | Validate

kubectl create -f nginx-daemonset.yaml

kubectl get po -o wide

kubectl get ds

kubectl describe ds nginx-ds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Cleanup

kubectl delete ds fluentd-ds

kubectl delete ds nginx-ds

kubectl get po

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**JOBS**

it is a controller that supervises a pod to carry out certain task.

it of two types

1) run to completition - pods created as part of this will shutdown automaticaly once the task is complted.

we need to delete this pods manully

2) cron job ( scedule job)

|  |
| --- |
| apiVersion: batch/v1  kind: Job  metadata:  name: countdown  spec:  template:  metadata:  name: countdown  spec:  containers:  - name: counter  image: centos:7  command:  - "bin/bash"  - "-c"  - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"  restartPolicy: Never |

# Create & Display

kubectl create –f countdown-jobs.yaml

kubectl get jobs

kubectl get po

kubectl describe jobs countdown

#Test

kubectl logs [POD\_NAME]

#Cleanup

kubectl delete jobs countdown

kubectl get po

**#Storage class**

1) storage class are not namespaced, they are global so name of storage class is across all namespace.

2)short form is sc (kubectl get sc)

3) one of storage class is default storage class

PV And PVC

Static and dynamic pv

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#INGRESS**

it is an Internal service.

load balancing is done internally.

**Stateful set**

https://www.youtube.com/watch?v=pPQKAR1pA9U

cant be randonly created or deleted

cant be randonly addressed

replica pods are not identical

it maintain a identity for each pod

next pod will created only , when previos pod is ip and running.

each pod has individual Dns name

when pod restart pod name and dns name will be same

**DATree**

best practice and rules and policy assginmnet tool

https://www.youtube.com/watch?v=hgUfH9Ab258

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**HELM**

package manger for kuberntes

it package yaml file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Health Check**

https://www.youtube.com/watch?v=mxEvAPQRwhw

**1) readiness probe:** it is desgin to let kuberntes know when your app is ready to serve traffic.

kubernates will make sure rediness probes passes before service allow send traffic to pod. if readiness probe fails it will stop

sending trafic to pod.

**2) liveness probe:** it help kubernats know whether your app is live or dead. if it is live it will leave that pod alone , but if it is

dead, it will remove the pod and create another pod to replace it.

3 typesof probs :

1) https

2) command

3) tcp

**Sevice Discovery**

no 2 pod comunicate directly using IP address

\*\*ingress\*\*

\*\* PV and pVC\*\* static and synamic provising

**statefull and stateless**

A stateless application is one which depends on no persistent storage.

A stateful application, on the other hand, needs data to be stored.

\*\*emptydir:

\*\*hostpath\*\*

\*\*storage volume\*\*

**Topics**

Statefulset

Metric service

Imperative commands