**KUBERNETES**

**Headless service ?**

When you need direct access to individual pods we can use headless service.  
In general a headless service is created with cluster IP as None , it will not get a virtual IP inside the cluster. It will be communicated using pod IP   
  
Difference between statefulset and deployment ?

|  |  |  |
| --- | --- | --- |
| **Feature** | **Deployment** | **StatefulSet** |
| **Purpose** | Manages **stateless** applications | Manages **stateful** applications |
| **Pod Identity** | Pods are **interchangeable** | Each pod has a **unique, persistent identity** |
| **Pod Name Format** | Random or generic (e.g., nginx-8fd6b7d7) | Fixed: <name>-0, <name>-1, etc. |
| **Stable Hostnames** | ❌ No | ✅ Yes (e.g., mysql-0.mysql) |
| **Persistent Storage** | Shared or ephemeral | Each pod gets a **dedicated PersistentVolume** |
| **Startup/Termination Order** | ❌ Not guaranteed | ✅ Strictly ordered (0 before 1, etc.) |
| **Scaling Behavior** | Easy horizontal scaling | Slower due to ordered operations |
| **Use Cases** | Web apps, APIs, microservices | Databases, Kafka, Zookeeper, Elasticsearch, rabbitmq |

**How metrics server will get the metrics?**  
**How metrics generated for the pods on these nodes?**  
**Kubelet is an agent running on each node it also run a sub component knows as cAdvisor or Container Advisor.Container Advisor is responsible for retrieving performance metrics from the pods and exposing it through the kubelet API thus making it available to the cluster.**  
**- Exposing container statistics to monitoring tools like Prometheus and Grafana.**  
**- But this metics will be stored in its in memory we do not get historical metrics.**  
  
**Can you roll back control plane changes?**   
**Yes we can do it from the aws console itself.**   
  
**How many versions the node group can be behind the control plane?**

**Nodes can be up to 2 minor versions behind the control plane.**

**Objects in Kubernetes?**

Objects in Kubernetes are Pods, namespaces, replica sets , Daemon sets, services

**2. Namespace:**

Namespace is an isolated environment in Kubernetes, and we have 3 different namespaces, which are \*default, \*Kube-system, \*kube-public   
→ **default namespace** is created automatically by Kubernetes   
→ **kube-node-lease** This namespace holds Lease objects associated with each node. Node leases allow the kubelet to send heartbeats so that the control plane can detect node failure.   
→ **Kube-system:** when the cluster is first set up kubernetes creates a set of pods and services for its internal use such as those required by networking solution, the DNS service etc., to isolate these from the user and to prevent you from accidentally deleting or modifying these services Kubernetes creates them under the kube-system namespace at the cluster startup.   
→ **kube-public:** This is the namespace where resources that should be made available to all users are created.

\*Command to create a namespace:   
$ Kubectl create namespace   
\*Command to get pods in a specific namespace: $ Kubectl get pods -n   
Ex: $ kubectl get pods -n dev

**3. Services in Kubernetes:**

Services enable communication between various components within and outside of the application. There are 4 kinds of services available

* NodePort: Where the service makes an internal POD accessible on a Port on the Node. NodePorts can only be in a valid range which by default is from 30000 to 32767.
* cluster IP: In this case the service creates a virtual IP inside the cluster to enable communication between different services such as a set of front-end servers to a set of back-end servers.
* LoadBalancer: Where it provisions a load balancer for our application in supported cloud providers. A good example of that would be to distribute load across the different web servers in your front-end tier.
* Services of type ExternalName map a Service to a DNS name, not to a typical selector such as my-service or cassandra. You specify these Services with the spec.externalName parameter.

**4. Node-Affinity:**

Node-affinity ensures the pods are hosted on a particular node with advance capabilities types

* Required during scheduling ignored during execution

Required --> It is mandatory for the scheduler to place the pod on the node with the given affinity rules, if it cannot find the pod will not be scheduled.

* Preferred during scheduling ignored during execution

Preferred --> if the matching node is not found the scheduler will ignore the node affinity rules and places the pod on the available nodes.

In both cases ---> Ignore ===> pod will continue to run and any change in node affinity will not impact them once they are scheduled

Uses operators like IN, NOTIN, EXISTS

* IN---to pace the pod on a given value
* NOTIN---to not to place on a given key and value
* Exists---it simply checks the labels and does not req values as it does not compare values

**5. I want to place the particular pd on a particular node how can we do it ?**   
**NODE SELECTORS:**

To place the pod on a particular node first we need to label the node

Kubectl label node node01 key=value

Then mention this key and value in the pod definition file

Under --> spec:

nodeSelector:

key: value

**6. How can we see the logs of pods and if pod contains container 1, 2, 3 how can we see logs of 2 container?**

$ Kubectl logs <pod-name>

$ Kubectl logs –f <pod-name>

$ kubectl logs <pod-name> -c <init-container2>…............. (To see logs of a specific container on a pod)

**7. Persistent volumes:**  
Lets say I have 20 micro services running in one namespace and I want all the 20 Microservices to access 1 file. How can you achieve it?

Persistent volume is a cluster of storage volume that which is used by users deploying applications on the cluster. The users can select storage from this cluster using persistent volume claims.  
PERSISTENT VOLUME

(ReadOnlyMany)

(ReadWriteOnce)

(ReadWriteMany)

|  |  |  |
| --- | --- | --- |
| **Access Mode** | **Description** | **Common Use Case** |

|  |  |  |
| --- | --- | --- |
| ReadWriteOnce | Volume can be **mounted as read-write by a single node**. | Databases, single-node apps |

|  |  |  |
| --- | --- | --- |
| ReadOnlyMany | Volume can be **mounted read-only by many nodes**. | Shared config/data, logs, etc. |

|  |  |  |
| --- | --- | --- |
| ReadWriteMany | Volume can be **mounted as read-write by many nodes at the same time**. | Shared file systems like NFS/EFS |

pv-definition.yaml

apiVersion: v1

kind: PersistentVolume metadata:

name: pv-vol1 specs: accessModes: - ReadWriteOnce capacity:

storage: 1Gi awsElasticBlockStore: volumeID: <volume-id> fstype: ext4

**8. Persistent Volume Claims:**

Users create persistent volume claims to use the storage. For each pvc user get a single persistent volume. The user specifies access modes and resources that he want to claim. Here access modes are ReadOnlyMany, ReadWriteOnce, ReadWriteMany.

pvc-definition.yaml

apiVersion: v1

kind: PersistentVolumeClaim metadata:

name: myclaim spec: accessModes: - ReadWriteOnce: resources: requests:

storage: 500Mi

**9. Kubernetes Architecture**

There are two kinds of nodes: master node and worker node. The master node is responsible for managing the Kubernetes cluster. The master node does all of these using a set of control plane components.

They are ETCD, Scheduler, Node controller, Replication Controller, Kube-api server, Kubelet, and Kube-proxy

→ *ETCD* is a database that stores information about nodes, pods, configs, secrets, accounts etc.

→ *Scheduler* is responsible for deciding which pod goes on which node

→ *Node Controller* is responsible for onboarding new nodes to the cluster when nodes become unavailable or are destroyed.

→ *Replication Controller* ensures the desired number of pods are running at all times.

→ The *Kube*-*Api server* is responsible for managing all the operations in the Kubernetes cluster. When you run the kubectl utility, in fact, kubectl reaches the Kube Api server. Then, the Kube Api server first authenticates and validates the request, retrieves the data from the ETCD cluster, and responds back with the requested information.

→ *Kubelet* is an agent that runs on each node, and it is responsible for creating pods on nodes. and also registers the worker nodes with the Kubernetes cluster.

→When it receives instructions to load a container or a POD on the node, it requests the container run time engine, which may be Docker, to pull the required image and run an instance.

→ *Kube-proxy* is a process that runs on each node in the Kubernetes cluster. It looks for new services, and every time a new service is created, Kube-proxy creates appropriate rules to forward traffic to those services to the backend pods. One way it does this is by using IP tables.

→ Different ways it does this is USER SPACE, where Kube proxy listens on a port for each service. Proxy’s connections to the pods by IPvs rules, and the default option is iptables.

**10. Master node taint effect?**

NoSchedule

**11. Autoscaling in Kubernetes.**

Another popular feature of Kubernetes is the ability to both vertically and horizontally autoscale your pods and nodes. Autoscaling in Kubernetes has three dimensions:

**Horizontal Pod Autoscaler (HPA):** adjusts the number of replicas of an application.

**Cluster Autoscaler:** adjusts the number of nodes of a cluster.

**Vertical Pod Autoscaler (VPA):** adjusts the resource requests and limits of a container.

The different auto-scalers work at one of two Kubernetes layers

* **Pod level:** The HPA and VPA methods take place at the pod level. Both HPA and VPA will scale the available resources or instances of the container.
* **Cluster level:** The Cluster Autoscaler falls under the Cluster level, where it scales up or down the number of nodes inside your cluster.

**apiVersion**: autoscaling/v2  
**kind**: HorizontalPodAutoscaler  
**metadata**:  
 **name**: php-apache  
**spec**:  
 **scaleTargetRef**:  
 **apiVersion**: apps/v1  
 **kind**: Deployment  
 **name**: php-apache  
 **minReplicas**: 1  
 **maxReplicas**: 10  
 **metrics**:  
 - **type**: Resource  
 **resource**:  
 **name**: cpu  
 **target**:  
 **type**: Utilization  
 **averageUtilization**: 50

**View logs for the last hour for a pod**  
kubectl logs --since=1h <pod>

**12. .How to debug CrashLoopBackOff of a pod in Kubernetes**

CrashLoopBackOff tells that a pod crashes right after the start. Kubernetes tries to start pod again, but again pod crashes and this goes in loop.

Sometimes the issue can be because of the Less Memory or CPU provided to application.

* Insufficient resources —lack of resources prevents the container from loading
* Config loading error- a server cannot load the configuration file.
* Misconfigurations—a general file system misconfiguration.
* Connection issues—DNS or Kube-DNS is not able to connect to a third-party service.

**13. Difference between CORE DNS and KUBE DNS**

Core DNS is deployed as a pod in kube-system namespace.

/etc/coredns

Kubernetes plugin is the one that makes core dns run

When you deploy core DNS solution. It also creates a service to make it available to the cluster. This service is called

KUBE-DNS by default. The ip address of this kube-dns service is configured as name server on the pod

(/etc/resolv.conf)

**14. Kubernetes Security.**

**kube-config file:** This config file is under home directory/.kube/config location

The kube-config file has 3 sections : **clusters, context, users**

**--> Clusters** are the type of k8s cluster that we need access to ex: dev-cluster, uat-cluster

**--> contexts:** which users account will be used to access which cluster **--> users:** user account with which you can access the cluster

**RBAC :----** We create a role to provide managing access with in the k8s cluster.

To provide permissions to the users to manage resources like pods, configMaps.

We need to create the role and role binding rules:

- apiGroups:

resources: pods, configmaps

verbs: list, get create, update, delete

**Cluster Role and Cluster role bindings**: These are similar to role and role binding but they work at cluster level that means if we want to give access to cluster scoped resources we can use cluster role and cluster role binding.

**Service accounts**:

To authenticate against machines. Is an account used by an application to interact with Kubernetes cluster

Ex: Jenkins uses service account to deploy applications on the Kubernetes cluster

Prometheus uses service account to pull the Kubernetes API for performance metrics.

**Image security:**  
This refers to strong images in a private registry everytime when a pod is deployed it will pull the required image from the registry.  
In deployment.yaml we provide   
imagePullSecrets:

* Name: gitlab-secret  
    
  **Security Context:**  
   with the help of security context we can set security standards such as the ID of the user to run a cotainner. If we specify at pod level It will carry over to all the containers with in the pod.
* 

Container security

**15. Readiness Probe:** To know when a container is start accepting traffic

**16. Startup Probe:**  To know when a container application has started.

**17. Liveness Probe:** To know when to restart a container.

**18. What exactly happens when you create a pod:**

1)The API server creates a POD object without assigning it to a node, updates the information in the ETCD server updates the user that the POD has been created.

2)The scheduler continuously monitors the API server and realizes that there is a new pod with no node assigned the scheduler identifies the right node to place the new POD on and communicates that back to the kube-apiserver.

3)The API server then updates the information in the ETCD cluster. The API server then passes that information to the kubelet in appropriate worker node.

4)The kubelet then creates the POD on the node and instructs the container runtime engine to deploy the application image.

5)Once done, the kubelet updates the status back to the API server and the API server then updates the data back in the ETCD cluster.

A similar pattern is followed every time a change is requested.

**19. What is a pod in kubernetes ?**

Ultimate aim is to deploy our application in the form of containers on worker nodes in a cluster. The containers are encapsulated into a Kubernetes object known as pods. A pod is a single instance of an application.

**20. .How a pod in one namespace can reach another pod in another namespace.**

**Q) I have 1 cluster with 2 different namespaces and I want to communicate one of our services to another namespace few services.**   
Q) Lets say we have 2 diff namespaces ns A and ns B . There is an application in ns A which needs to connect to App in B.How do we point to another pod running a diff ns

The resources within a namespace can refer to each other simply by their names. The Web app Pod can reach a service in another namespace as well.For this you must append the name of the namespace to the name of the service. For example for the web pod in the default namespace to connect to the database in the dev namespace

ex: mysql.connect("db-serive.dev.svc.cluster.local")

- you're able to do this because when the service is created a DNS entry is added automatically in this format looking closely at the DNS name of the service.

The last part Cluster.local is the default domain name of the Kubernetes cluster SVC is the subdomain for

service followed by the namespace and then the name of the service itself.   
  
how does the pods know which pod should I target to communicate. how do you point a service to reach specific set of pods

We use labels and selectors in svc yaml

Now I did not create svc and I have 2 podsd in same namespace how can anble communication with each other   
Every Pod gets a unique cluster-internal IP address.  
With the help of that we can enable communication

How does kubernetes resolve DNS? Like when I hit a service it is reaching to a specific pod.  
Kubernetes runs an internal DNS server, typically CoreDNS (used to be kube-dns earlier).   
This DNS service runs as a set of pods in the cluster and watches the Kubernetes API for changes to services and pods.  
When ever a service is created this DNS service creates a record for the service.it maps the service name to the IP address  
**DNS resolves Service name → Service ClusterIP** (by CoreDNS)  
  
Once the pod is created , kubernetes API will invoke the network plugin that may be CNI , Flannel. Then it will assign a pod Ip from the cluster of Ip pool. In this way pod gets an IP.  
  
- On worker nodes there is a service ices call kube-proxy , it is the one which is responsible for creating appropriate rules between the set of front end and backend applications.It looks for new services like whenever a new service is created it create appropriate rules to forward traffic to those backend pods.  
  
**DNS resolves Service name → Service ClusterIP** (by CoreDNS)  
  
------> **CoreDNS** is the **default DNS server** in Kubernetes. It provides **DNS-based service discovery**, allowing **pods and services to find and communicate with each other** using names instead of IPs.

**21.Issus and the troubleshooting methods**  
**\*** Pod stuck in **Pending** state à The pod is stuck in pending state may be the issue is with resource availability or sometimes if the volume that referenced in the deployment is not available in such cases pod will stuck in pending state.

**\*** Pod status **ImagePullBackOff** à ImagePullBackOff happens when Kubernetes try to pull image from the repository but the imagepullSecret mentioned or created is incorrect therefore authentication fails with the repository and it can’t pull image.

**\*** Pod status **ErrImagePull** à this happens when the image it is trying to pull is not present in the respective repository.

**\*** Pod status **ConfigError** à This issue occurs when there is issue with pod specific config map  
\* we use to run mutilple jobs in our jobs namespace so everytime a a job is created it creates a serviuce object and though we delete the job pod the service object remains in the cluster holding the IP and after all the IP got hold to service objects if we spawnb a new pod then it will stuck in pending state.  
\*

**22. How pod get assigned with an IP address in Kubernetes**

- A pod gets assigned an IP address by the network plugin (such as Calico, Flannel, or CNI) when the pod is created.  
- The Kubernetes control plane interacts with the network plugin to allocate an IP address from the cluster's IP address pool, ensuring each pod has a unique IP within the cluster.   
- This IP address allows the pod to communicate with other pods and services within the Kubernetes network.

**23. Difference between CORE DNS and KUBE DNS**

\* Core DNS is deployed as a pod in kube-system namespace.

/etc/coredns

Kubernetes plugin is the one that makes core dns run

When you deploy core DNS solution. It also creates a service to make it available to the cluster. This service is called

KUBE-DNS by default. The ip address of this kube-dns service is configured as name server on the pod

(/etc/resolv.conf)

**24. Ingress Controller**

It is solution that is deployed in the Kubernetes cluster we deployed our ingress controller in ingress namespace. It has the ability to monitor the Kubernetes cluster for any new ingress resources are created then it will configure the nginx server accordingly. To do it’s the ingress controller requires a service account with correct roles and role-bindings.   
 **Ingress Resource:**   
 An ingress resource is a set of rules and configurations applies on the ingress controller  
 We can configure rules to forward traffic to an application based on the path defined.

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: minimal-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

ingressClassName: nginx-example

rules:

- http:

paths:

- path: /testpath

pathType: Prefix

backend:

service:

name: test

port:

number: 80

**Daemonsets:**  
Daemonset ensures that one copy of pod is always present on all the nodes in the cluster.  
Whenever a new node is added a replica of the pod is added automatically to the node and when a node is removed the pod is automatically removed.

**Ex;** In our cluster we depoy fluentd as a daemonset. So that a copy of fluentd is present on every node which helps to pull the required logs.  
$ kubectl get ds –n namespace-name

**What exactly happens in the backend when we delete a pod ?**

When you delete a Kubernetes pod (using kubectl delete pod <pod-name> or when it's deleted via some controller), the following sequence of events happens in the background:

### **1. API Server Marks Pod for Deletion**

* The Kubernetes API server receives the delete request and sets a **deletion timestamp** on the pod object.
* This timestamp marks the pod as *“terminating.”*

### **2. Graceful Termination Period Begins**

* The pod gets a **grace period** (default: 30 seconds unless overridden via --grace-period).
* During this time, the **kubelet** sends a SIGTERM signal to the main process of each container in the pod.

### **3. PreStop Hook (if defined) Runs**

* If a preStop hook is configured in the pod spec, it executes before the container is stopped.
* Kubernetes waits for this to finish (or times out) before proceeding.

### **4. Container Process Handles SIGTERM**

* The application should handle SIGTERM to begin a graceful shutdown (e.g., finish in-flight requests, close DB connections).
* If the container doesn’t exit before the grace period ends, Kubernetes forcefully kills it with SIGKILL

### **5. Pod Removed from Endpoints**

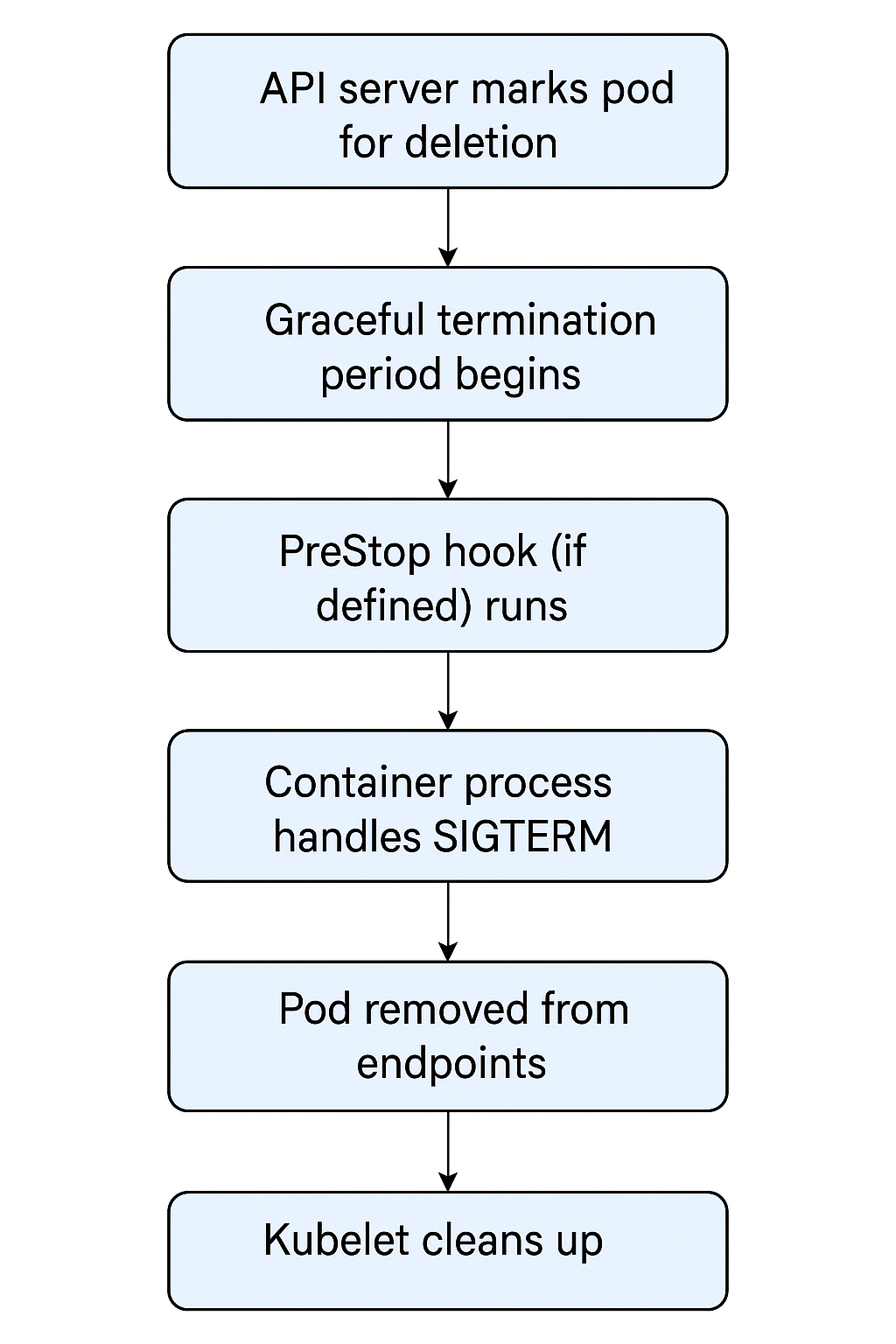
* The pod is removed from associated **Services**, meaning it no longer receives traffic.
* This helps in preventing new requests from hitting the terminating pod.

### **6. Kubelet Cleans Up**

* After the container processes exit, the kubelet:
  + Removes the pod from its internal state.
  + Deletes container resources (e.g., cgroups, network interfaces).
  + Releases volumes attached to the pod (if any).
  + Sends a status update to the API server.

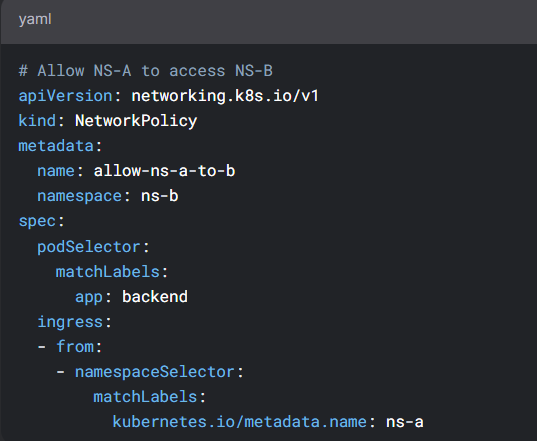
### **7. Pod Object is Deleted from etcd**

* Finally, once all cleanup is complete, the pod object is deleted from **etcd** (Kubernetes’ key-value store).

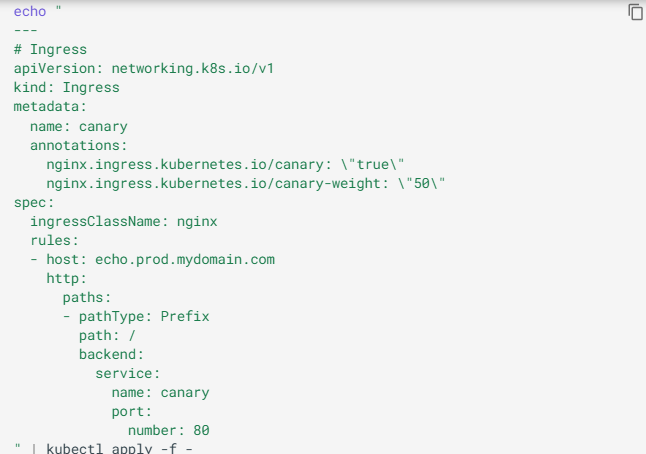
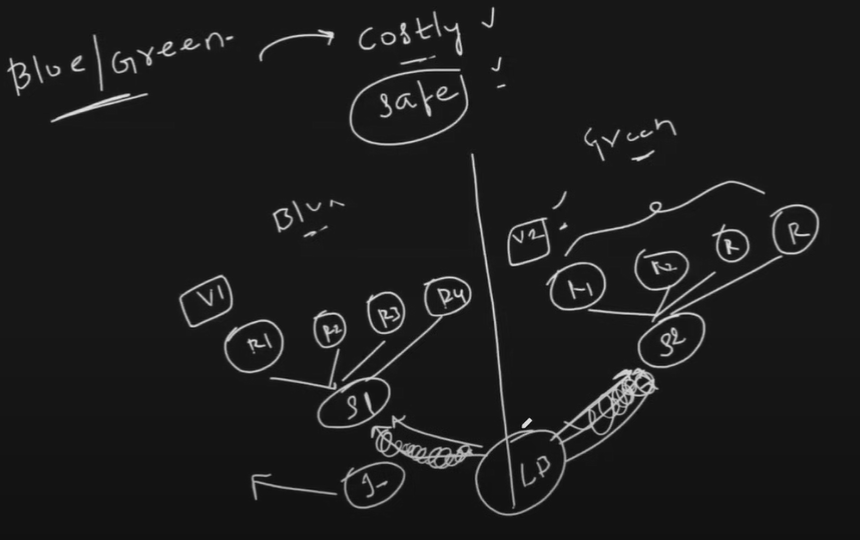


**Network Policy in Kubernetes ?**  
**How do you enforce rules in kubernetes to control which pods can talk to each other**

Network policy helps to control network traffic between pods at the IP and port level enhancing security in the cluster.  
Here in this we have 2 policy types to control traffic they are ingress for incoming traffic and egress for outgoing traffic.  
  
  
The entities that a Pod can communicate with are identified through a combination of the following three identifiers:

1. Other pods that are allowed (exception: a pod cannot block access to itself)
2. Namespaces that are allowed
3. IP blocks (exception: traffic to and from the node where a Pod is running is always allowed, regardless of the IP address of the Pod or the node)  
   

**when ever you restart a pod a certain data is lost how can you handle this**

We can use pv , pvc to persists the data.  
Generally a pv is a cluster wide pool of storage volume to be used by users deploying applications in the cluster.  
The users can claim the storage using persistant volume claim.a pvc will make the storage available to the node  
Each pvc is bound to a single pv  
  
  
**Deployment Strategies in kubernetes?**  
--> Rolling update (default deployment strategy for kubernetes)  
--> Canary Deployment.  
--> Blue-green deployment  
We need deployment strategy to reduce the downtime during the deployment or version upgrade.  
  
a) Rolling update:   
Let's say we have a deployment with version 1 and 4 replicas,and after sometime our dev team will want to deploy v2. Here when we deploy version2 it will replcace the replicas one after the other that means it firstly it will bring a new replica of v2, once it is up and running it will bring dowm one replica of v1  
similarly it follows for other replicas.  
  
b) Canary Deployment:  
In this we test new version of the application to the production and only allowing this new version to only certain set of audience.  
For example, lets say we have 100 users and here we will load balance 90 % of traffic to the older version and 10 % of traffic to the newer version.  
  
In this we can control the traffic flow towards our application.  
  
  
c) Blue-Green Deployment:  
We acheived Blue-green deployment like when we deploy a new version of the application.  
simultaneously a BVT job will run with some test cases. if the EKS cluster returns 200 OK status then it will remove the older version and keep the newer version as the final version in this way we acheived blue green deployment strategy for our kubernetes deployments.  
  
In general if we face any issue like our green pod is not stable we just update the ingress resource to point to the blue pod in this way we can roll back  


How many types of EKS provisioning we can do ?  
Amazon EKS (Elastic Kubernetes Service) offers multiple provisioning types based on control plane management and worker node deployment strategies. Here are the main types:

**Control Plane Management Types**

1. **EKS Auto Mode** – Fully managed Kubernetes cluster by AWS, including worker nodes and auto-scaling.
2. **EKS Standard Mode** – AWS manages the control plane, but users manage worker nodes (EC2 instances).
3. **EKS Fargate Mode** – Serverless Kubernetes where AWS runs pods directly without user-managed nodes.
4. **EKS Anywhere** – Kubernetes clusters running on on-premises infrastructure using AWS-supported tooling.

**Worker Node Deployment Modes**

1. **EKS Managed Node Groups** – AWS provisions and manages EC2 worker nodes with auto-scaling and automated updates.
2. **EKS Self-Managed Nodes** – Users manually provision and manage EC2 instances for EKS worker nodes.
3. **EKS Fargate Mode** – No worker nodes; AWS runs Kubernetes pods directly in a serverless fashion.

**Difference between data plane and control plane ?**  
Kubernetes architecture is divided into two core logical components:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Control Plane** | **Data Plane** |
| **Definition** | Brain of the cluster. Manages and maintains the cluster state. | Worker part of the cluster. Runs actual workloads (containers). |
| **Main Role** | Makes decisions (e.g., scheduling, scaling, maintaining desired state). | Executes decisions made by the control plane (runs pods). |
| **Key Components** | - API Server - Scheduler - Controller Manager - etcd - Cloud Controller | - Kubelet - Kube Proxy - Container Runtime (e.g., containerd, Docker) |
| **Runs On** | Master nodes / control plane nodes | Worker nodes / node agents |
| **Stores Cluster State** | Yes (in etcd) | No – just executes workloads |
| **Handles Networking?** | No direct pod traffic routing | Yes – Kube Proxy routes pod-to-pod traffic |
| **Example Actions** | - Scheduling pods - Rolling updates - Scaling deployments - Approving certificate requests | - Running nginx, Java apps, databases in pods - Forwarding traffic |
| **Failure Impact** | If it fails, you can't make changes, but existing pods may still run. | If it fails, your apps stop running or responding. |

## IF my config map is in control plane what will happen ?

* All ConfigMaps are stored in **etcd**, the backing store of the Kubernetes control plane.
* This is **normal and expected** — you **create** and **manage** ConfigMaps using the API server.
* any pod referring to this ConfigMap pulls it via the API server.

## **When Pods Use a ConfigMap:**

You can mount it as:

* **Environment variables**
* **Volumes (as files)**

### **If the ConfigMap is changed:**

* **Mounted as env**: Pods need to be restarted to pick up changes.
* **Mounted as volume**: Changes propagate within a few seconds (Kubelet syncs it)

**In eks can we access the control plane?**  
In eks we cannot access the control plane as it is a managed kubernetes service   
we cannot ssh into control plane  
**But You Can Still:**

* **Control resources** via kubectl and the **API server endpoint**.
* Use **IAM Roles for Service Accounts (IRSA)** for fine-grained security.
* Enable **VPC endpoint access** for private API communication.

**How to upgrade the AMI of worker node ?**

a) Find or create a new AMI  
b) Create a New Launch template --> we need to make sure that instance type, IAM roles etc., match the worker nodes.  
C) Update the ASG EC2 --> ASG --> edit---> to use the new launch template   
d) set the desired capacity --> to trigger instance replacement  
e) then use   
 kubectl drain <old-node-name>   
this will safely evict the pods from the nodes

**Pod disruption Budget (PDB) ?**  
**Imagine your team needs to update your Kubernetes nodes but you don’t want your applications to suddenly stop working? --> This is where PDB comes in**  
**PDB –** *simply a mechanism in Kubernetes that defines how many replicas of your application can be removed or restarted simultaneously.*

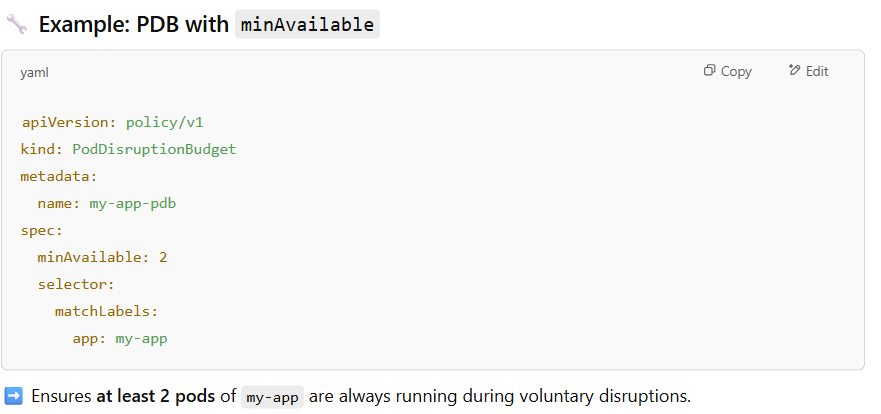
A PDB limits the number of Pods of a replicated application that are down simultaneously from voluntary disruptions.  
It helps ensure **high availability** during:

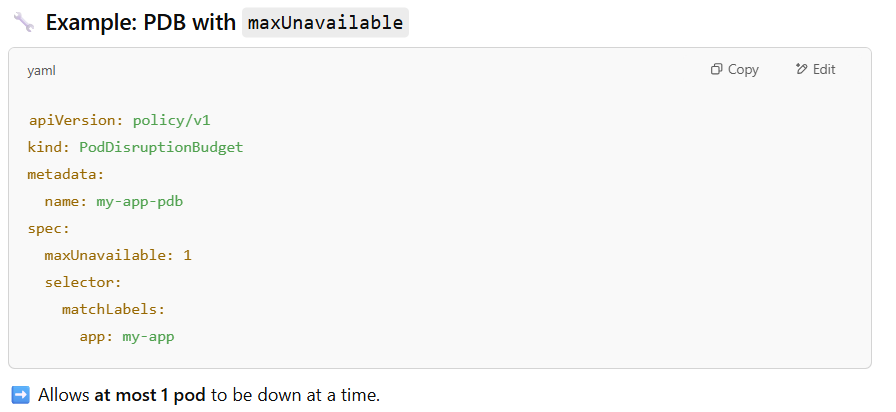
* Node drains (e.g., maintenance)
* Voluntary disruptions (e.g., cluster upgrades, eviction)
* Rolling updates

For example:  
 A Deployment which has a .spec.replicas: 5 is supposed to have 5 pods at any given time. If its PDB allows for there to be 4 at a time, then the Eviction API will allow voluntary disruption of one (but not two) pods at a time.  
  
**How It Works:**

You define a PDB with:

* **minAvailable**: Minimum number of pods that must be up
* **OR**
* **maxUnavailable**: Maximum number of pods allowed to be unavailable





|  |  |
| --- | --- |
| **Topic** | **Detail** |
| **Voluntary disruptions** | Evictions due to node maintenance, draining, rolling updates |
| **Involuntary disruptions** | Pod crashes, node failures – PDB **does not protect against these** |
| **Applies to** | Deployments, ReplicaSets, StatefulSets, etc. |
| **Works with** | Eviction API (kubectl drain, kubectl delete pod) |
| **Does not work with** | kubectl delete deployment or scale 0 — that removes PDB protection |

### **❗ Important:**

* A bad PDB (e.g., minAvailable: 5 with 3 pods) can block operations like draining nodes or upgrades.
* PDBs do **not ensure scaling or healing**, just availability during disruptions.

**Can you explain what all components are present in control plane ?**

There are two kinds of nodes: master node and worker node. The master node is responsible for managing the Kubernetes cluster. The master node does all of these using a set of control plane components.

They are ETCD, Scheduler, Node controller, Replication Controller, Kube-api server, Kubelet, and Kube-proxy

→ *ETCD* is a database that stores information about nodes, pods, configs, secrets, accounts etc.

→ *Scheduler* is responsible for deciding which pod goes on which node

→ *Node Controller* is responsible for onboarding new nodes to the cluster when nodes become unavailable or are destroyed.

→ *Replication Controller* ensures the desired number of pods are running at all times.

→ The *Kube*-*Api server* is responsible for managing all the operations in the Kubernetes cluster. When you run the kubectl utility, in fact, kubectl reaches the Kube Api server. Then, the Kube Api server first authenticates and validates the request, retrieves the data from the ETCD cluster, and responds back with the requested information.

Types of application  
There are two types stateful and state less   
  
on what basis we differentiate this application stateless and this application is stateful.  
the **key to differentiating between a stateless and stateful application** lies in **how the application handles client data or session between requests**.  
**1. Does the application need to remember user data or sessions between requests?**

* **Yes → Stateful**
* **No → Stateless**

#### **2. Can any replica handle any request at any time without loss of context?**

* **Yes → Stateless**
* **No → Stateful** (some requests may depend on previous ones or specific pod memory)

#### **3. Does the application use a database or filesystem to store user data that must persist?**

* **Yes → Stateful**
* **No → Could be Stateless** (if it just computes and returns)

#### **4. Can you freely scale the application horizontally (add/remove replicas) without coordination or concern for data?**

* **Yes → Stateless**
* **No → Stateful**

Difference between config map and secret   
A) configmap is an object that has all the configuration required by the deployment in a key and value format under the data section in the configmap.yaml file   
basically used to pass configuration data in the form of key value pairs.  
  
We can make config map available to pod by referencing them under env section or as a configuration under volume   
  
Secrets ===> we have confidential data in the encrypted format like password, keys, tokens   
  
we can make secret available to pod by referencing them under env section or as a secret under volume  
  
  
**"Your pod is running. Your service is green. But users can't connect. What now?"**

Here's what I was looking for:  
- Check kubectl get endpoints - does the service point to any pods?  
- Validate the service selector was it changed during a rollout?  
- Think DNS nslookup? can the pod resolve other services via

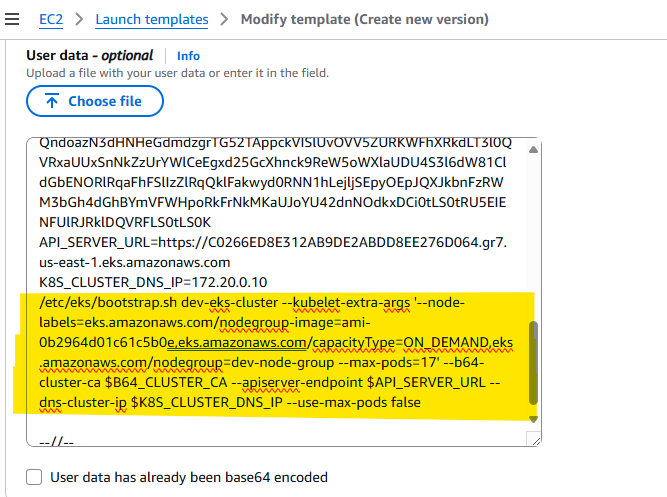
- Investigate RBAC - does the pod have permission to query services?  
- Check network policies are packets silently being dropped?

**Q) Taints and tolerations**  
Taints and tolerations are used to set restrictions on what pods can be scheduled on a node.  
Taints are set on nodes   
Tolerations are set on pods   
  
First we need to taint the node   
$ kubectl taint nodes node-name key=value:taint-effect  
here taint effect can be NoSchedule, PreferNoSchedule, NoExecute  
**NoSchedule** --- Pods will not be scheduled on the node  
**PreferNoSchedule** --- system will try to place the pod on the node but not guaranteed   
**NoExecute** ---- new pods will not be scheduled on the node existing pods will be evicted if they do not toleratee the taint

**Q) How deployment performs an upgrade under the hood?**  
When a new deployment is When a new deployment is created. Say to deploy five replicas. It creates a replica set automatically. Which in turn creates the number of ports required to meet the number of replicas.  
  
 When you upgrade your application. deployment object. Creates a new replica set under the hood. And starts deploying the containers thereafter. At the same time taking down the pods in the old replica set following a rolling update strategy.  
  
**Q) kubernetes components version classification?**  
No component should be at a higher version than the kube-api-server  
- controller manager & scheduler – can be one version lower

- kubelet & kube-proxy – can be 2 versions lower

-kubectl ----> + or – 1 or = to kube-apiserver version ex: if api server is 1.30 then kubectl can be at 1.31 or 1.30 or 129

**Q) How worker nodes are added to the eks cluster ?**  
When ASG launches an eks worker node. The ASG launch template has a bootstrap script that will be passed in the user data of the ASG launch template. This bootstrap script help the worker nodes to join the eks cluster.  
  
  
This script:

* Connects to the EKS control plane (via kubelet)
* Registers the new node with the cluster
* Uses the **IAM Role** to get sts:AssumeRole permissions