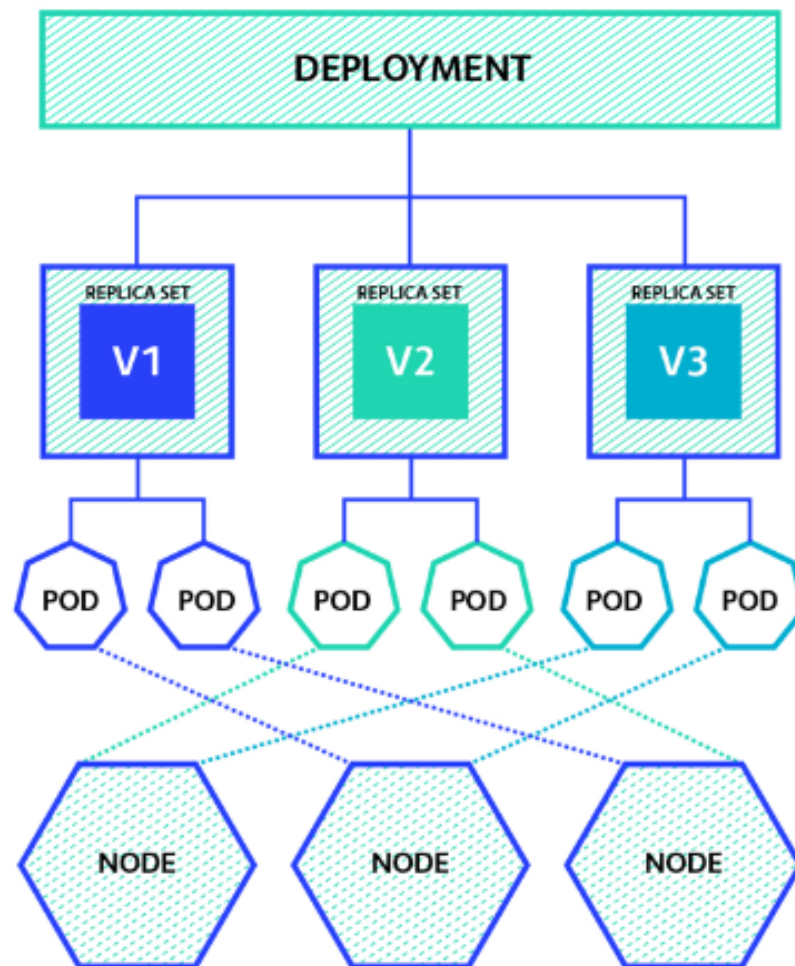


What are Kubernetes Deployments?

Deployment in Kubernetes is a declarative resource object that defines the **desired state for a set of replica Pods**. It provides a way to **manage the deployment and scaling of containerized applications** within a Kubernetes cluster.

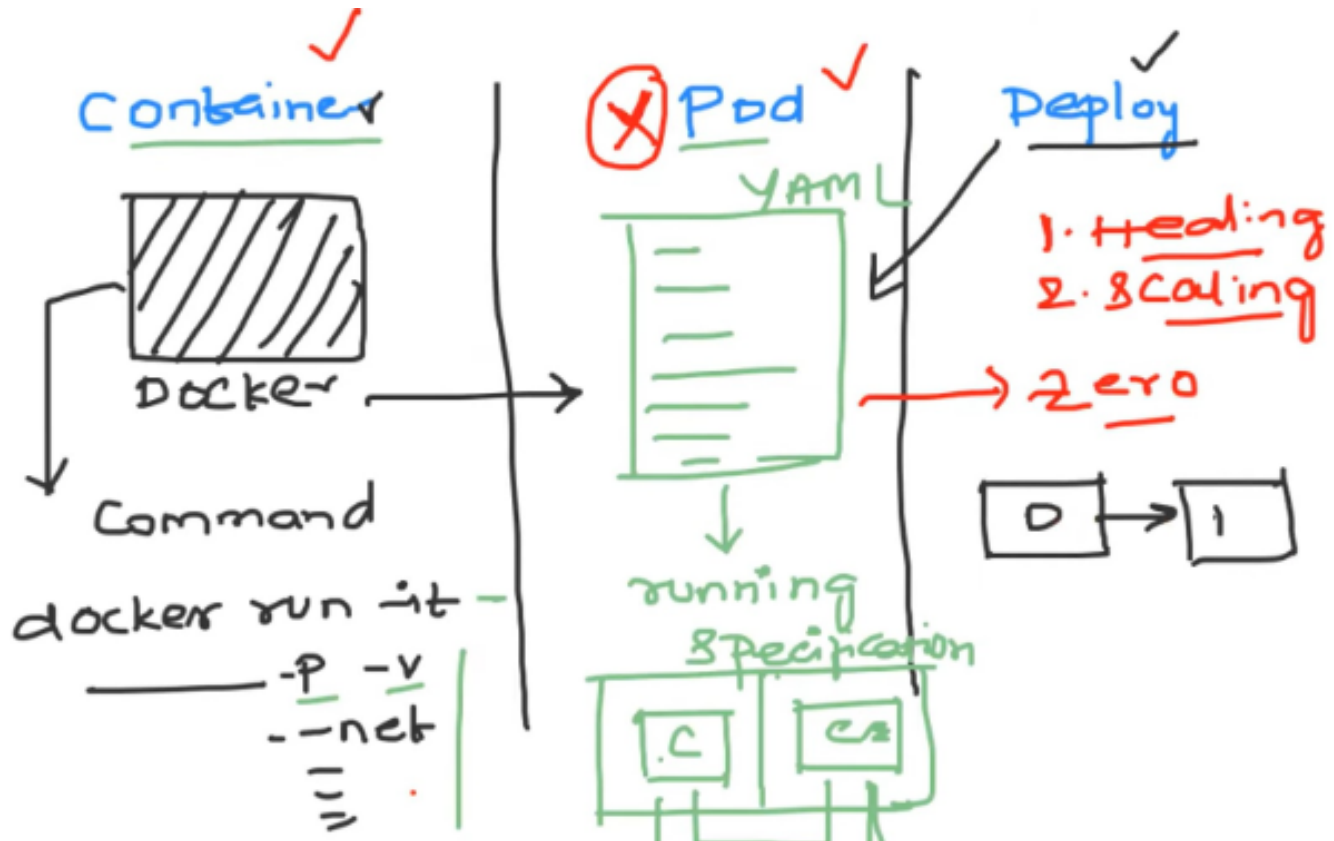
- In **simpler terms**, a Deployment defines how many copies of an application should run, what each copy should look like, and how to handle updates and scaling, ensuring that the desired state is maintained. It abstracts away the underlying infrastructure and automates the management of application instances, making it easier to ensure availability, scalability, and reliability of applications in a Kubernetes environment.



What are the differences between a container, a pod, and a deployment?

Containers are the basic building blocks of applications and Pods provide a way to group one or more containers together, Deployments are higher-level constructs used to manage the lifecycle and desired state of Pods. Deployments simplify tasks like scaling, updating, and rolling back

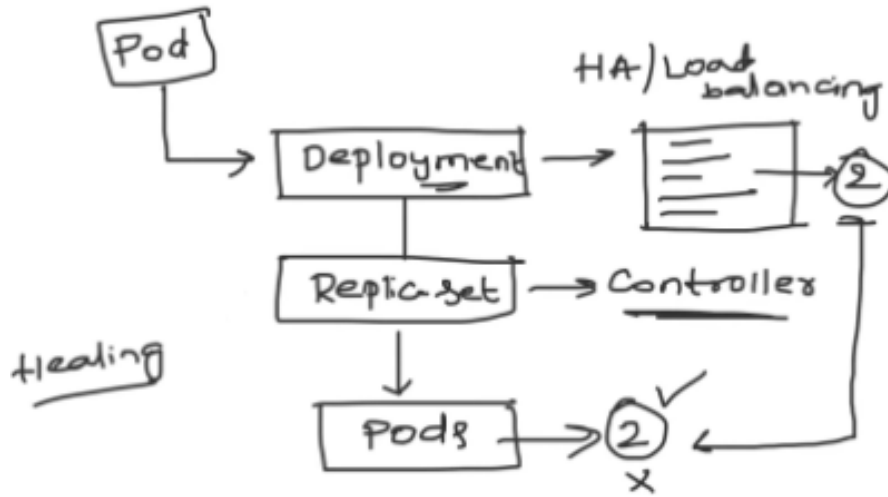
applications in a Kubernetes cluster, making them a crucial resource for managing containerized



- A **container** is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.
- A **pod** is a group of one or more containers that are scheduled and run together on the same node. Pods are the smallest deployable unit in Kubernetes.
- A **deployment** is a Kubernetes object that manages the deployment and scaling of a set of pods. Deployments ensure that a specified number of pods are running at all times, and they can be used to update pods automatically.

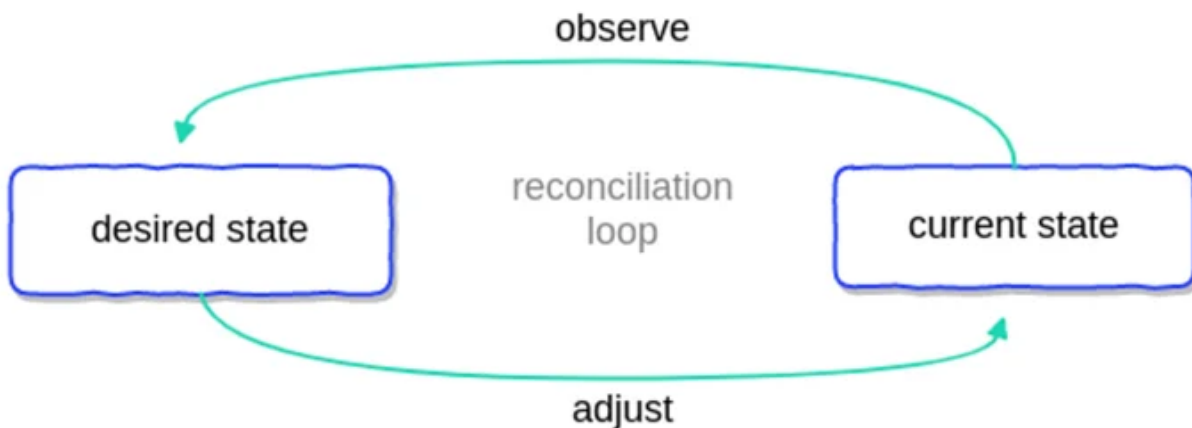
Here is a table that summarizes the key differences between containers, pods, and deployments:

Feature	Container	Pod	Deployment
Definition	A lightweight, standalone, executable package of software	A group of one or more containers that are scheduled and run together on the same node	A Kubernetes object that manages the deployment and scaling of a set of pods
Use cases	Running individual applications or services	Running a single application or service that consists of multiple containers	Running a set of pods that represent a single application or service
Management	Managed by the container runtime	Managed by Kubernetes	Managed by Kubernetes
Scalability	Can be scaled up or down manually	Can be scaled up or down automatically	Can be scaled up or down automatically



What is controller in K8's ?

In Kubernetes, a Controller is a key part of the control plane responsible for managing and ensuring the desired state of various resources within the cluster. Controllers are control loops that continuously work to reconcile the **current state of resources with the desired state** specified in the Kubernetes API objects. They help in automating various tasks such as scaling, healing, and updating application



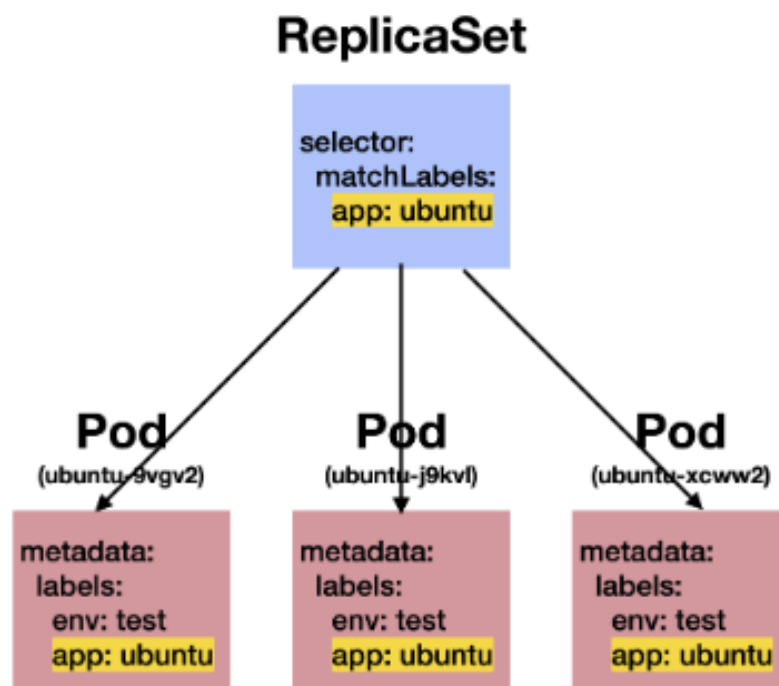
Various controllers in Kubernetes:

1. **ReplicaSet Controller:** Ensures a specified number of replica Pods are running, maintaining desired scale.
2. **Deployment Controller:** Manages app deployments, automating updates and rollbacks.
3. **StatefulSet Controller:** Orchestrates stateful app deployments with stable network identities.
4. **DaemonSet Controller:** Ensures a specific Pod runs on every node, ideal for system-level services.

5. **Job Controller:** Handles one-off batch tasks, parallelizable workloads.
6. **CronJob Controller:** Schedules and automates recurring Jobs based on a cron-like schedule.
7. **Service Controller:** Manages network access to Pods, exposing services internally or externally.
8. **Ingress Controller:** Routes external traffic to services, often used for HTTP and HTTPS traffic.
9. **HPA Controller:** Dynamically adjusts replica counts based on resource utilization or custom metrics.
10. **Custom Controllers:** Enables custom resource management by developing your own controller logic.

Explain ReplicaSet Controller ?

A ReplicaSet in Kubernetes is a resource object used to ensure that a **specified number of replica Pods are running at all times**. It is part of Kubernetes' built-in mechanisms for maintaining the desired state of applications in a cluster



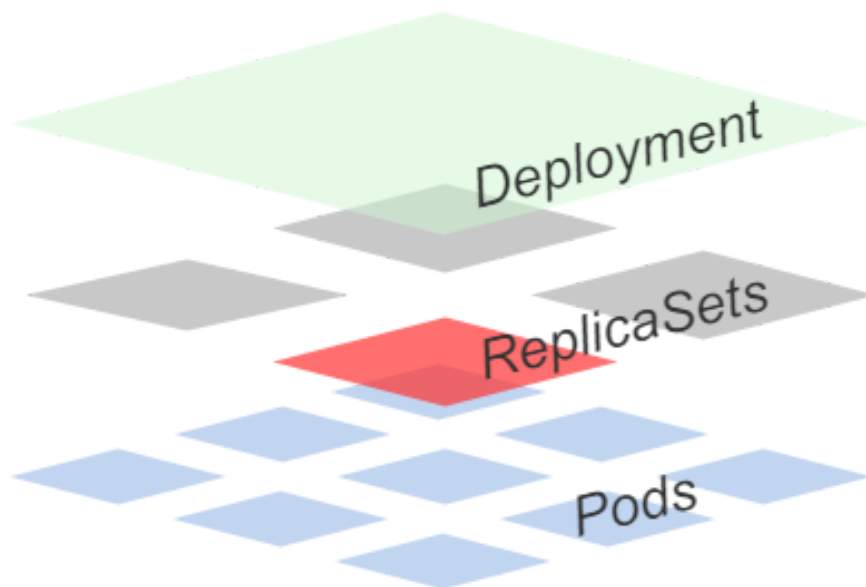
Here are some key points about ReplicaSets:

1. **Desired Replicas:** ReplicaSets allow you to specify the desired number of identical replica Pods that should be running. This number is defined in the ReplicaSet's configuration.
2. **Automatic Scaling:** If the actual number of Pods deviates from the desired number (e.g., due to node failures or manual deletions), the ReplicaSet controller takes corrective actions to reconcile the difference, either by creating new Pods or terminating excess Pods.

3. **Immutable Pods:** Once a Pod is created by a ReplicaSet, it should not be manually modified. Instead, if changes are needed, you should update the template in the ReplicaSet specification, and the controller will manage the rollout of the updated Pods.
4. **Selector:** ReplicaSets use a selector to identify which Pods they should manage. The selector is based on labels, and it matches Pods with labels that match the selector criteria.
5. **Use Cases:** ReplicaSets are often used when you need to ensure a certain level of availability and fault tolerance for your applications. They are replaced by the more advanced Deployment resource in most modern Kubernetes setups, which provides additional features like rolling updates and rollbacks.

Explain difference between Deployment vs replicaset in k8's ?

Both Deployments and ReplicaSets are used for maintaining a desired number of replica Pods, Deployments provide a higher-level and more feature-rich abstraction that is particularly well-suited for managing applications, handling updates, and ensuring smooth rollbacks. ReplicaSets are a lower-level building block that may be used in more specific scenarios where fine-grained control over replicas is necessary without the need for advanced application management features.



Deployment:

- Use it when you want to easily manage your application's lifecycle, including updates and rollbacks.
- It's like a manager for your app, making sure the right number of copies run and helping with smooth updates.

ReplicaSet:

- Use it when you only need to ensure a fixed number of copies of your app are running.
- It's like a basic worker that keeps a specific number of app copies, but it doesn't handle updates or rollbacks.

In most cases, you'd use Deployments because they offer more control and features for managing applications. ReplicaSets are used for simpler cases where you only need to maintain a fixed number of replicas.

What does 'kubectl get all -A' command in Kubernetes do, and what information does it provide?"

The `kubectl get all` command and `kubectl get all -A` command are used to retrieve information about resources within a Kubernetes cluster. However, they have different behaviors:

```

k8s-demo git:(master) * kubectl get all
NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP     10.96.0.1    <none>        443/TCP    84m

k8s-demo git:(master) * kubectl get all -A
NAMESPACE   NAME                                     READY   STATUS    RESTARTS   AGE
kube-system pod/coredns-64897985d-46vt5            1/1     Running   0           84m
kube-system pod/etcd-minikube              1/1     Running   0           84m
kube-system pod/kube-apiserver-minikube   1/1     Running   0           84m
kube-system pod/kube-controller-manager-minikube 1/1     Running   0           84m
kube-system pod/kube-proxy-4d5m7          1/1     Running   0           84m
kube-system pod/kube-scheduler-minikube    1/1     Running   0           84m
kube-system pod/storage-provisioner        1/1     Running   1 (83m ago) 84m

NAMESPACE   NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
default      service/kubernetes  ClusterIP     10.96.0.1    <none>        443/TCP    84m
kube-system  service/kube-dns     ClusterIP     10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP 84m

NAMESPACE   NAME                                     DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
kube-system daemonset.apps/kube-proxy            1         1         1         1             1           kubernetes.io/os=linux 84m

NAMESPACE   NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
kube-system deployment.apps/coredns        1/1      1             1           84m

```

1. kubectl get all:

- When you run `kubectl get all`, it attempts to retrieve resources with the type "all" which doesn't exist in Kubernetes. As a result, you will typically get an error message indicating that there is no resource type called "all."

2. kubectl get all -A:

- When you run `kubectl get all -A`, it retrieves information about all resources in all namespaces within the Kubernetes cluster. This is a handy way to get a comprehensive overview of the resources across all namespaces, including Pods, Services, Deployments, ConfigMaps, and more.

The `-A` flag is a shorthand for specifying all namespaces. It tells `kubectl` to look for resources in every namespace, not just the default one. This is useful for cluster-wide resource exploration and monitoring.

1. Created Pod and Executed it

```

k8s-demo git:(master) x ls
deployment.yml pod.yml
k8s-demo git:(master) x vim pod.yml
k8s-demo git:(master) x kubectl apply -f pod.yml
pod/nginx created
k8s-demo git:(master) x kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           6s
k8s-demo git:(master) x kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP           NODE      NOMINATED NODE   READINESS GATES
nginx     1/1     Running   0           12s   172.17.0.3   minikube   <none>           <none>
k8s-demo git:(master) x minikube ssh
k8s-demo git:(master) x ssh -i <> sfgg
k8s-demo git:(master) x minikube ssh
Last login: Mon Feb 27 13:02:58 2023 from 192.168.49.1
docker@minikube:~$ curl

```

2. Created deployment file with Replicaset

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3 # Specify the desired number of NGINX Pods
  selector:
    matchLabels:
      app: nginx # Label selector to identify NGINX Pods
  template:
    metadata:
      labels:
        app: nginx # Label applied to NGINX Pods
    spec:
      containers:
        - name: nginx-container
          image: nginx:latest # NGINX container image
          ports:
            - containerPort: 80 # Port the NGINX container listens on

```

3. Deployments trigger replicaset and create pods

Here's a step-by-step breakdown:

- You create or update a Deployment resource with a desired number of replicas and a Pod template (specifying container images, labels, etc.).
- The Deployment controller (part of the Kubernetes control plane) analyzes the desired state and compares it to the current state of the cluster.
- If there are fewer Pods than desired, the Deployment controller creates new Pods based on the template.
- If there are more Pods than desired, the Deployment controller scales down by terminating excess Pods.
- If you update the Deployment (e.g., change the container image), the Deployment controller orchestrates a rolling update, gradually replacing old Pods with new ones to minimize downtime.

So, Deployments serve as a higher-level abstraction that simplifies the management of ReplicaSets and Pods, making it easier to handle application scaling, updates, and rollbacks in a declarative and automated manner.

```
→ k8s-demo git:(master) x vim deployment.yml
→ k8s-demo git:(master) x kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
→ k8s-demo git:(master) x kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    1/1     1            1           7s
→ k8s-demo git:(master) x kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-9456bbbf9-hkkmw    1/1     Running   0           10s
→ k8s-demo git:(master) x Deploy -> ReplicaSet -> pod
```

What is 'kubectl get pods -w ' mean ?

The `kubectl get pods -w` command is used to **watch and continuously monitor the status of Pods** in your Kubernetes cluster. When you run this command, it provides a real-time stream of information about the Pods, and it updates the output whenever there are changes to the Pod status.

Here's what each part of the command does:

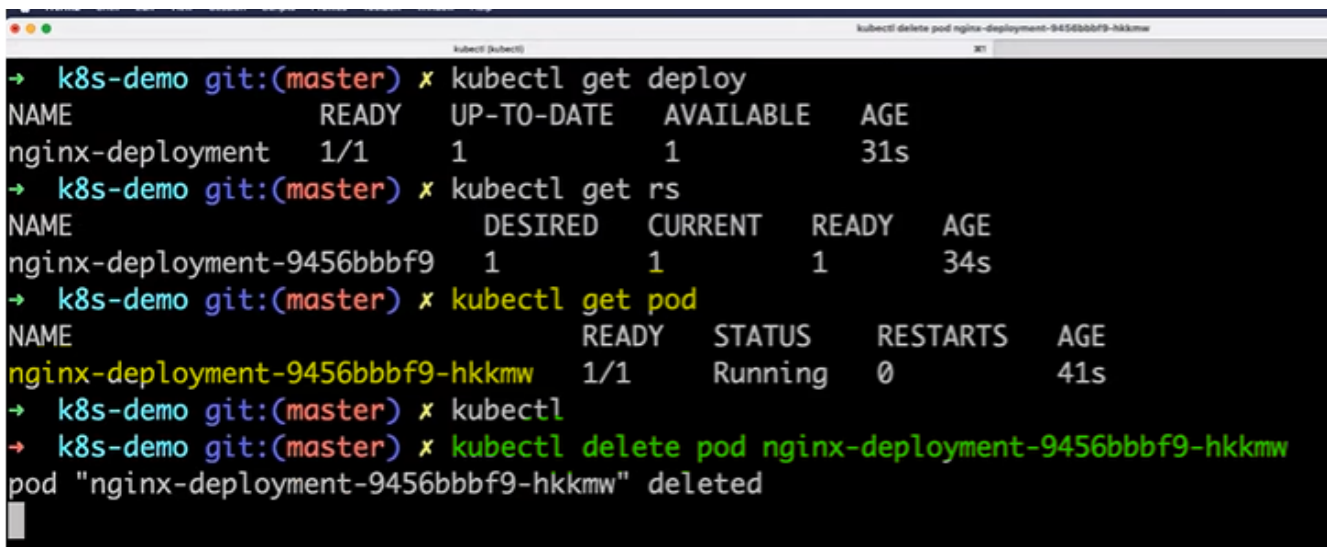
- `kubectl` : The command-line tool for interacting with Kubernetes clusters.

- `get pods` : This part of the command tells Kubernetes to retrieve information about Pods in the cluster.
- `-w` or `--watch` : The `-w` flag indicates that you want to watch for changes in real-time.

4. "How can you monitor the real-time status of a specific Pod's deletion in Kubernetes, and what command would you use for this purpose?"

When you delete a Pod using `kubectl delete pod <pod-name>`, the Pod is immediately removed from the cluster. If you want to watch the status of the Pod deletion in real-time, you can combine the deletion command with `kubectl get pods -w`. Here's how you can do it:

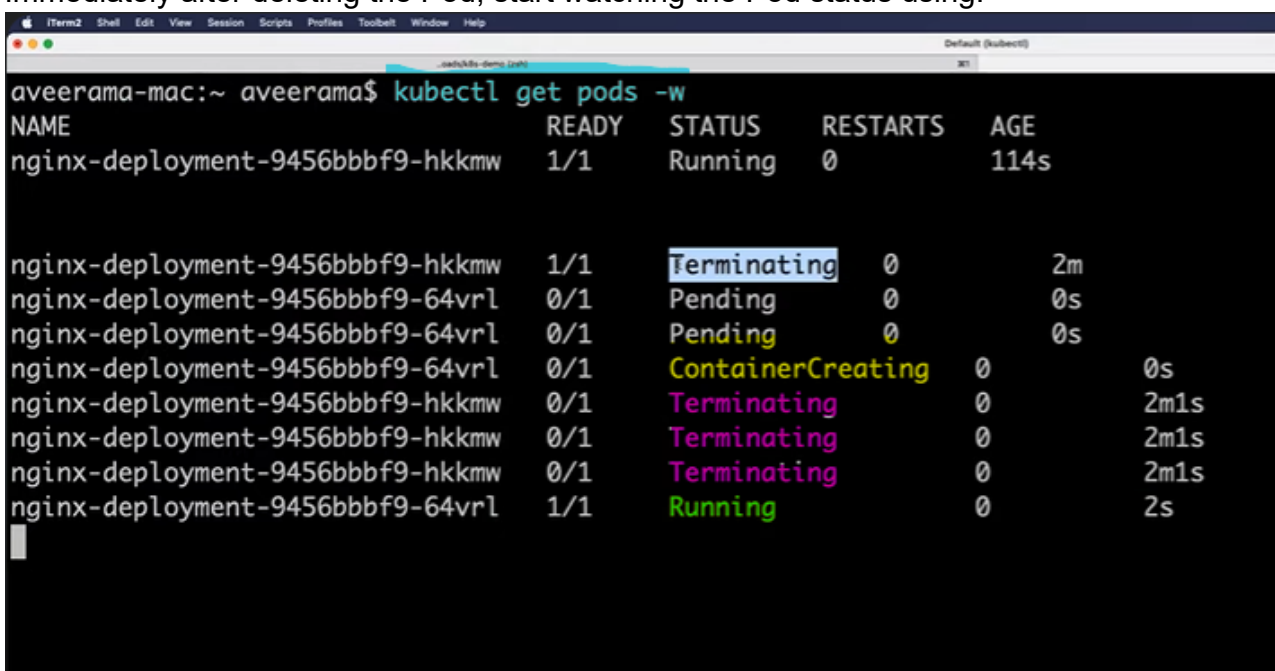
1. Open your terminal.
2. Delete the Pod you want to monitor (replace `<pod-name>` with the actual name of the Pod):



```

k8s-demo git:(master) x kubectl get deploy
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment 1/1     1            1           31s
k8s-demo git:(master) x kubectl get rs
NAME          DESIRED   CURRENT   READY   AGE
nginx-deployment-9456bbbf9 1          1         1       34s
k8s-demo git:(master) x kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
nginx-deployment-9456bbbf9-hkkmw 1/1     Running   0          41s
k8s-demo git:(master) x kubectl
k8s-demo git:(master) x kubectl delete pod nginx-deployment-9456bbbf9-hkkmw
pod "nginx-deployment-9456bbbf9-hkkmw" deleted
  
```

3. Immediately after deleting the Pod, start watching the Pod status using:



```

aveerama-mac:~ aveerama$ kubectl get pods -w
NAME          READY   STATUS    RESTARTS   AGE
nginx-deployment-9456bbbf9-hkkmw 1/1     Running   0          114s

nginx-deployment-9456bbbf9-hkkmw 1/1     Terminating 0          2m
nginx-deployment-9456bbbf9-64vrl 0/1     Pending      0          0s
nginx-deployment-9456bbbf9-64vrl 0/1     Pending      0          0s
nginx-deployment-9456bbbf9-64vrl 0/1     ContainerCreating 0          0s
nginx-deployment-9456bbbf9-hkkmw 0/1     Terminating 0          2m1s
nginx-deployment-9456bbbf9-hkkmw 0/1     Terminating 0          2m1s
nginx-deployment-9456bbbf9-hkkmw 0/1     Terminating 0          2m1s
nginx-deployment-9456bbbf9-64vrl 1/1     Running      0          2s
  
```

This will give you a real-time view of the Pod's status as it is terminated and eventually disappears from the list of Pods. The `-w` flag ensures that the `kubectl get pods` command continues to watch for updates until you manually stop it by pressing `Ctrl + C`.

Certainly, here's a concise conclusion about Deployments and ReplicaSets in Kubernetes:

- **Deployment:** Deployments are a higher-level Kubernetes resource used for managing application deployment, scaling, updates, and rollbacks. They provide a declarative and automated way to handle application lifecycle management and are the preferred choice for most scenarios due to their advanced features.
- **ReplicaSet:** ReplicaSets are a lower-level resource in Kubernetes focused solely on maintaining a specific number of replica Pods. They don't support application updates or rollbacks but are useful when you need fine-grained control over replicas in simpler scenarios.

In summary, Deployments are the go-to choice for managing applications in Kubernetes as they offer more control and features for application management, including seamless updates and rollbacks. ReplicaSets are used for specific cases where you only need to ensure a fixed number of replicas without the need for advanced application management.

--- Prudhvi Vardhan ([LinkedIn](#))