😊 Namespaces

In Kubernetes, namespaces are a way to organize and isolate resources within a cluster. They provide a scope for names, allowing different teams, projects, or applications to share a Kubernetes cluster while maintaining logical separation. Here are some key aspects and use cases for namespaces in Kubernetes:

1. **Resource Isolation:** Namespaces provide a logical separation of resources within a Kubernetes cluster. Each resource, such as pods, services, deployments, and replica sets, belongs to a specific namespace. This isolation helps prevent naming conflicts and resource collisions between different teams or applications sharing the same cluster.
2. **Multi-tenancy:** Kubernetes namespaces support multi-tenancy by allowing multiple teams or users to coexist within the same cluster while maintaining isolation. Each team or user can have their namespace to deploy and manage their applications independently. This enables efficient resource utilization and resource quota enforcement for different tenants.
3. **Environment Segmentation:** Namespaces can be used to segment resources across different environments, such as development, staging, and production. By deploying applications into separate namespaces for each environment, teams can ensure that changes are tested in isolation before promoting them to production.
4. **Resource Quotas:** Kubernetes namespaces support resource quotas, which allow cluster administrators to limit the amount of CPU, memory, and other resources that can be consumed within a namespace. This helps prevent resource contention and ensures fair resource distribution among different namespaces.
5. **Access Control:** Kubernetes RBAC (Role-Based Access Control) policies can be applied at the namespace level, allowing administrators to grant fine-grained permissions to users and service accounts based on their namespace membership. This enables administrators to control who can create, modify, or delete resources within a namespace.
6. **Organization and Management:** Namespaces provide a way to organize and manage resources based on different criteria, such as teams, projects, or application tiers. Administrators can use namespaces to group related resources together, making it easier to monitor, troubleshoot, and maintain the cluster.
7. **Namespace-scoped Services:** By default, Kubernetes services are accessible cluster-wide. However, you can create services that are only accessible within a specific namespace. These namespace-scoped services are useful for exposing services internally without exposing them to the entire cluster.
8. **Namespace-level Networking Policies:** Kubernetes Network Policies allow you to define rules for how pods within a namespace can communicate with each other and with other resources in the cluster. This enables administrators to enforce network segmentation and security policies at the namespace level.

😊 Use cases of Namespaces:

Namespaces in Kubernetes offer a versatile way to organize and manage resources within a cluster. Here are some common use cases for namespaces:

1. **Multi-tenancy:** Namespaces allow different teams or tenants to share a single Kubernetes cluster while maintaining isolation. Each team can have its namespace to deploy and manage applications independently. This enables efficient resource utilization and isolation of resources and configurations.
2. **Environment Segmentation:** Namespaces can be used to separate resources across different environments, such as development, staging, and production. By deploying applications into separate namespaces for each environment, teams can ensure that changes are tested in isolation before being promoted to production.
3. **Resource Quotas:** Kubernetes namespaces support resource quotas, allowing cluster administrators to limit the amount of CPU, memory, and other resources that can be consumed within a namespace. This helps prevent resource contention and ensures fair resource distribution among different teams or projects.
4. **Access Control:** RBAC (Role-Based Access Control) policies can be applied at the namespace level, allowing administrators to grant fine-grained permissions to users and service accounts based on their namespace membership. This enables administrators to control who can create, modify, or delete resources within a namespace.
5. **Application Isolation:** Namespaces provide a way to isolate applications and their resources from each other within a cluster. This isolation helps prevent conflicts and interference between applications, allowing them to run independently without affecting each other's performance or stability.
6. **Namespace-scoped Services:** By default, Kubernetes services are accessible cluster-wide. However, you can create services that are only accessible within a specific namespace. These namespace-scoped services are useful for exposing services internally without exposing them to the entire cluster.
7. **Resource Labeling and Tagging:** Namespaces can be used as a way to label or tag resources within a Kubernetes cluster. By associating resources with specific namespaces, administrators can easily group, filter, and manage resources based on their namespace membership.
8. **Namespace-level Networking Policies:** Kubernetes Network Policies allow you to define rules for how pods within a namespace can communicate with each other and with other resources in the cluster. This enables administrators to enforce network segmentation and security policies at the namespace level.

In this guide, we're exploring Kubernetes namespaces, which provide a way to organize and isolate resources within a cluster. The end goal is to understand the purpose and use cases of namespaces and how to work with them in a Kubernetes environment.

Summary:

1. **Understanding namespaces: We discuss how namespaces provide resource isolation, multi-tenancy support, environment segmentation, resource quotas, access control, organization, and management within a Kubernetes cluster.**

2. **Use cases of namespaces: We explore common scenarios where namespaces are beneficial, such as multi-tenancy, environment segmentation, resource quotas, access control, application isolation, namespace-scoped services, resource labeling, and network policies.**
3. **Lab setup: We provide instructions for interacting with namespaces in a Kubernetes cluster, including checking existing namespaces, creating a new namespace, deploying Pods within a namespace, and deleting namespaces and associated resources.**
4. **Namespace deletion: We explain the consequences of deleting a namespace, including the deletion of all resources within the namespace and the namespace itself, and provide instructions for deleting namespaces safely.**

Overall, the guide aims to equip users with the knowledge and practical skills needed to effectively utilize namespaces for resource organization, isolation, and management in Kubernetes clusters.

😊 To begin with the Lab:

1. Now to move further first your cluster should be running. In case you have deleted your cluster then first start your cluster. Then move ahead.
2. So, the first command that we are going to execute is to check how many objects are there in our namespace.

   kubectl get ns

```
ubuntu@ip-172-31-25-114:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   20m
kube-node-lease   Active   20m
kube-public       Active   20m
kube-system       Active   20m
ubuntu@ip-172-31-25-114:~$
```

3. Now we will run this command. This command will show us all the object in our default namespace.

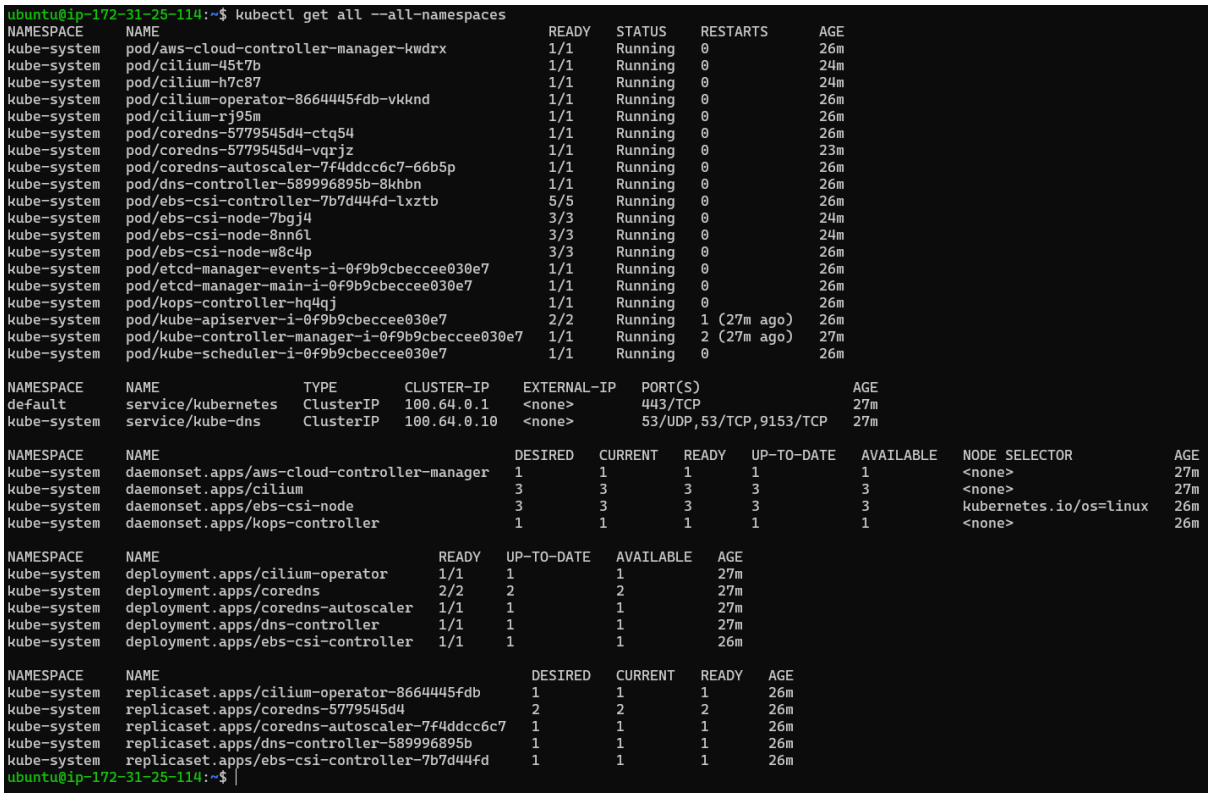
   kubectl get all

```
ubuntu@ip-172-31-25-114:~$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   100.64.0.1   <none>        443/TCP   24m
ubuntu@ip-172-31-25-114:~$
```

4. After that we will run this command and this will show us all the resources from all the namespaces.

kubectl get all --all-namespaces

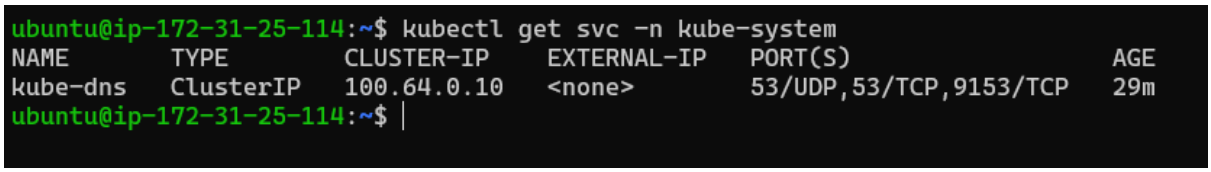
5. Below you can see that there is a lot of information.



```
ubuntu@ip-172-31-25-114:~$ kubectl get all --all-namespaces
NAMESPACE     NAME                                                READY   STATUS    RESTARTS      AGE
kube-system   pod/aws-cloud-controller-manager-kwdrx              1/1     Running   0             26m
kube-system   pod/cilium-45t7b                                    1/1     Running   0             24m
kube-system   pod/cilium-h7c87                                    1/1     Running   0             24m
kube-system   pod/cilium-operator-8664445fdb-vkknd                1/1     Running   0             26m
kube-system   pod/cilium-rj95m                                    1/1     Running   0             26m
kube-system   pod/coredns-5779545d4-ctq54                         1/1     Running   0             26m
kube-system   pod/coredns-5779545d4-vqrjz                         1/1     Running   0             23m
kube-system   pod/coredns-autoscaler-7f4ddcc6c7-66b5p             1/1     Running   0             26m
kube-system   pod/dns-controller-589996895b-8khbn                 1/1     Running   0             26m
kube-system   pod/ebs-csi-controller-7b7d44fd-lxztb               5/5     Running   0             26m
kube-system   pod/ebs-csi-node-7bgj4                              3/3     Running   0             24m
kube-system   pod/ebs-csi-node-8nn6l                              3/3     Running   0             24m
kube-system   pod/ebs-csi-node-w8c4p                              3/3     Running   0             26m
kube-system   pod/etcd-manager-events-i-0f9b9cbeccee030e7          1/1     Running   0             26m
kube-system   pod/etcd-manager-main-i-0f9b9cbeccee030e7           1/1     Running   0             26m
kube-system   pod/kops-controller-hq4qj                           1/1     Running   0             26m
kube-system   pod/kube-apiserver-i-0f9b9cbeccee030e7              2/2     Running   1 (27m ago)   26m
kube-system   pod/kube-controller-manager-i-0f9b9cbeccee030e7     1/1     Running   2 (27m ago)   27m
kube-system   pod/kube-scheduler-i-0f9b9cbeccee030e7              1/1     Running   0             26m

NAMESPACE     NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                  AGE
default       service/kubernetes   ClusterIP   100.64.0.1    <none>        443/TCP                  27m
kube-system   service/kube-dns     ClusterIP   100.64.0.10   <none>        53/UDP,53/TCP,9153/TCP   27m

NAMESPACE     NAME                                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
kube-system   daemonset.apps/aws-cloud-controller-manager   1         1         1       1            1           <none>                   27m
kube-system   daemonset.apps/cilium                         3         3         3       3            3           <none>                   27m
kube-system   daemonset.apps/ebs-csi-node                   3         3         3       3            3           kubernetes.io/os=linux   26m
kube-system   daemonset.apps/kops-controller                1         1         1       1            1           <none>                   26m

NAMESPACE     NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   deployment.apps/cilium-operator       1/1     1            1           27m
kube-system   deployment.apps/coredns               2/2     2            2           27m
kube-system   deployment.apps/coredns-autoscaler    1/1     1            1           27m
kube-system   deployment.apps/dns-controller        1/1     1            1           27m
kube-system   deployment.apps/ebs-csi-controller    1/1     1            1           26m

NAMESPACE     NAME                                             DESIRED   CURRENT   READY   AGE
kube-system   replicaset.apps/cilium-operator-8664445fdb       1         1         1       26m
kube-system   replicaset.apps/coredns-5779545d4                2         2         2       26m
kube-system   replicaset.apps/coredns-autoscaler-7f4ddcc6c7    1         1         1       26m
kube-system   replicaset.apps/dns-controller-589996895b        1         1         1       26m
kube-system   replicaset.apps/ebs-csi-controller-7b7d44fd      1         1         1       26m
ubuntu@ip-172-31-25-114:~$
```

6. Now if you want to look from a specific namespace then you execute this command.

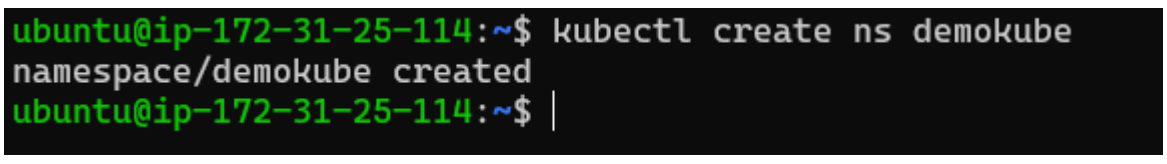
kubectl get svc -n kube-system



```
ubuntu@ip-172-31-25-114:~$ kubectl get svc -n kube-system
NAME       TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   100.64.0.10   <none>        53/UDP,53/TCP,9153/TCP   29m
ubuntu@ip-172-31-25-114:~$
```

7. Now we are going to create our namespace. Below you can see that we have used this command to create our first namespace.

kubectl create ns demokube



```
ubuntu@ip-172-31-25-114:~$ kubectl create ns demokube
namespace/demokube created
ubuntu@ip-172-31-25-114:~$
```

8. Here you can see that if you execute this command to check your namespaces then you can see your newly created namespace here.

```
ubuntu@ip-172-31-25-114:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   49m
demokube          Active   10m
kube-node-lease   Active   49m
kube-public       Active   49m
kube-system       Active   49m
ubuntu@ip-172-31-25-114:~$
```

9. Now we are going to run pod in this namespace. To run a pod in the demokube namespace, you can create a Pod manifest YAML file specifying the namespace and then apply it using kubectl.

   vim pod.yaml

10. This YAML file describes a simple Pod named my-pod with one container running the NGINX image. The namespace field is set to demokube, so the Pod will be created in the demokube namespace.

    apiVersion: v1
    kind: Pod
    metadata:
    name: my-pod
    namespace: demokube
    spec:
    containers:
    - name: my-container
    image: nginx

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: demokube
spec:
  containers:
  - name: my-container
    image: nginx
```

11. Run the following command to apply the Pod manifest. This command will create the Pod defined in the YAML file within the demokube namespace.

kubectl apply -f pod.yaml

12. You can verify that the Pod has been created successfully by running. This command lists all Pods in the demokube namespace. You should see my-pod listed among them.

kubectl get pods -n demokube

```
ubuntu@ip-172-31-25-114:~$ kubectl apply -f pod.yaml
pod/my-pod created
ubuntu@ip-172-31-25-114:~$ kubectl get pods -n demokube
NAME     READY   STATUS    RESTARTS   AGE
my-pod   1/1     Running   0          12s
ubuntu@ip-172-31-25-114:~$
```

13. To delete the Pod named my-pod in the demokube namespace, you can use the kubectl delete command. This command will delete the Pod named my-pod in the demokube namespace.

kubectl delete pod my-pod -n demokube

14. If you want to delete all resources (including Pods, Deployments, Services, etc.) in the demokube namespace, you can use the below command. This command will delete all resources in the demokube namespace.

kubectl delete all --all -n demokube

15. If you delete a namespace in Kubernetes, it will delete all the resources (Pods, Deployments, Services, etc.) that are part of that namespace. Here's what happens when you delete a namespace:

- **All resources in the namespace are deleted:** Deleting a namespace effectively deletes all the resources within that namespace. This includes Pods, Deployments, Services, ConfigMaps, Secrets, and any other Kubernetes resources that belong to that namespace.
- **Graceful termination of Pods:** Kubernetes will attempt to gracefully terminate all the Pods running in the namespace. This means that the Pods will go through their termination lifecycle, allowing them to clean up resources and handle any ongoing processes before being terminated.
- **Resources are removed from API server:** Once all the resources in the namespace have been deleted, Kubernetes removes all references to that namespace from the API server.
- **Namespace is removed:** Finally, Kubernetes removes the namespace itself from the cluster.

16. It's important to note that deleting a namespace is a destructive operation, and all data associated with the namespace will be lost. Therefore, you should exercise caution when deleting namespaces, especially in production environments.
17. To delete a namespace, you can use the kubectl delete namespace command followed by the name of the namespace you want to delete:
18. After executing this command, all resources within the demokube namespace will be deleted, and the namespace itself will be removed from the cluster

kubectl delete namespace demokube

```
ubuntu@ip-172-31-25-114:~$ kubectl delete namespace demokube
namespace "demokube" deleted
ubuntu@ip-172-31-25-114:~$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-25-114:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   76m
kube-node-lease   Active   76m
kube-public       Active   76m
kube-system       Active   76m
ubuntu@ip-172-31-25-114:~$ kubectl get pods -n demokube
No resources found in demokube namespace.
ubuntu@ip-172-31-25-114:~$
```