Namespace

In Kubernetes, Namespace is a way to organize and isolate resources within a cluster. A namespace provides a virtual cluster within a physical cluster, allowing multiple teams or applications to coexist within the same Kubernetes cluster

To create a namespace , you can use the kubectl create namespace command

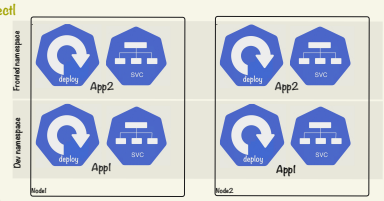
```
kubectl create namespace dev
```

1

```
apiVersion: v1
kind: Namespace
metadata:
 name:dev
```

2

you can also create a YAML file that defines your namespace and use the kubectl apply command to create the namespace



Each namespace has its own set of resources, such as pods, services, storage volumes that are isolated from resources in other namespaces. This helps to prevent naming conflicts between resources and allows different teams or applications to manage their own resources independently

Namespaces provide a way to organize resources and apply resource quotas, network policies, and other settings at a namespace level. For example, you can limit the number of pods or services that can be created in a namespace, or restrict network traffic between pods in different namespaces.

Resource quotas

Resource quotas in k8s are a way to limit the amount of compute resources that can be consumed by a set of pods in a namespace. A resource quota is defined as a Kubernetes object that specifies the maximum amount of CPU, memory, and other resources that can be used by pods in a namespace

```
kubectl describe resourcequota saas-team-quota
```

=>

```
Name:                   saas-team-quota
Namespace:              dev
Resource                Used  Hard
--------                ----  ----
configmaps              0     5
limits.cpu              1     4
limits.memory           2     4Gi
persistentvolumeclaims  0     5
pods                    5     10
requests.cpu            1     2
requests.memory         2     2Gi
secrets                 0     5
services                1     5
services.loadbalancers  0     2
services.nodeports      0     3
count/deployment.apps   1     4
```

This command will display detailed information about the saas-team-quota ResourceQuota object, including the current usage and maximum limits for each resource

2

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: saas-team-quota
  namespace: dev
spec:
  hard:
    pods: "10"
    requests.cpu: "2"
    requests.memory: 2Gi
    limits.cpu: "4"
    limits.memory: 4Gi
    configmaps: "5"
    persistentvolumeclaims: "5"
    replicationcontrollers: "5"
    secrets: "5"
    services: "5"
    services.loadbalancers: "2"
    services.nodeports: "3"
    count/deployment.apps: "4"
```

ResourceQuota object specifies the maximum limits for the following resources

The maximum number of pods that can be created in the namespace

The total amount of CPU that can be requested by all pods in the namespace

The total amount of memory that can be requested by all pods in the namespace

The total amount of CPU that can be used by all pods in the namespace

The total amount of memory that can be used by all pods in the namespace

If a ResourceQuota is applied to a namespace but no resource constraints are defined for the pods in the template section of a Deployment YAML file, then the Deployment and ReplicaSet will still be created. However, no pods will enter the running state, as the ResourceQuota will prevent them from consuming any resources

```
k -n dev get events
```

```
$ k describe ns dev
Name:         dev
Labels:       <none>
Annotations:  <none>
Status:       Active
Resource Quotas
 Name:                   comput-quota
 Resource                Used  Hard
 --------                ---   ---
 count/deployments.apps  1     2
 cpu                     6m    100m
 memory                  60M   100M
 pods                    6     10
No LimitRange resource.
```

LimitRange

LimitRange is a resource object that is used to specify default and maximum resource limits for a set of pods in a namespace

When a LimitRange is applied to a namespace, it will only affect newly created pods. Existing pods will not have their resource limits automatically updated to match the LimitRange settings

3

```
apiVersion: v1
kind: LimitRange
metadata:
  name: dev-resource-limits
  namespace: dev
spec:
  limits:
  - default:
      cpu: 100m
      memory: 128Mi
    defaultRequest:
      cpu: 50m
      memory: 64Mi
    max:
      cpu: 500m
      memory: 512Mi
    min:
      cpu: 50m
      memory: 32Mi
    type: Container
```

LimitRange is used to set default and maximum resource limits for individual pods or containers within a namespace, while ResourceQuota is used to set hard limits on the total amount of resources that can be used by all the pods in a namespace

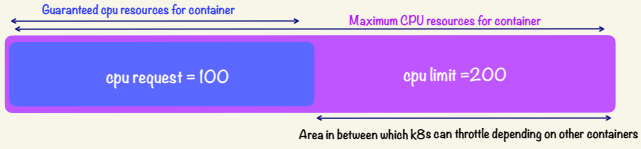
---

Resource Requirements & Limits

Resource requirements and limits are used to specify the amount of CPU and memory resources that a container requires in order to run properly

Resource requirements are set in the pod specification and indicate the minimum amount of CPU and memory resources that a container needs to run. Kubernetes uses these requirements to determine which nodes in the cluster have the necessary resources to schedule the pod

Resource limits specify the maximum amount of CPU and memory resources that a container is allowed to use. Kubernetes enforces these limits by throttling the container's resource usage if it exceeds the specified limit



Container requires at least 100 milliCPU (0.1 CPU) and 10 megabytes of memory to run

Container is limited to using no more than 200 milliCPU (0.2 CPU) and 50 megabytes of memory

```
kunectl describe node kubeworker-1
```

==>

```
...
Capacity:
  cpu:                4
  memory:             8192Mi
  pods:               110
Allocatable:
  cpu:                3
  memory:             7168Mi
  pods:               110
...
```

The Capacity section shows the maximum amount of resources (such as CPU and memory) that a node in the Kubernetes cluster has available

Allocatable section, shows the amount of resources that Kubernetes has allocated for use by containers and pods on the node

4

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: dev
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.17
          resources:
            requests:
              cpu: "100m"
              memory: "10M"
            limits:
              cpu: "200m"
              memory: "50M"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
```

if you do not specify the request and limits values for a container , the pod will be assigned default values for CPU and memory. The default request value is 0.5 CPU and 256Mi memory, while the default limits value is 1 CPU and 256Mi memory

Setting resource requirements and limits is important for ensuring that containers have the necessary resources to run effectively without overloading the system. By specifying resource limits, you can prevent containers from using too many resources and causing performance issues or crashes

When a container reaches or exceeds its memory limit, the Linux kernel's Out of Memory Killer (OOM Killer) is invoked. The OOM Killer is responsible for selecting and terminating processes to free up memory when system memory becomes critically low. By default, Kubernetes lets the OOM Killer select and terminate the process within the container that triggered the OOM condition.