DevOps on Kubernetes

...

Goals of this project:

- Understand the components of K8s
- How and when use K8s resources
- The management of a K8s cluster, including the access to its resources and a basic role management
- The configuration needed for cluster users
- A bare-metal cluster configuration

The main purpose of this project is to understand how to deploy simulations inside of a K8s cluster.

What is?

Kubernetes (K8s) is an open-source system that let you automate the deployment of containers on a cluster of machines distributed over the network. Those machine are called ***nodes***.

The strength of K8s is that let your deployed services always available, even without the direct cluster administrator management.

kubernetes

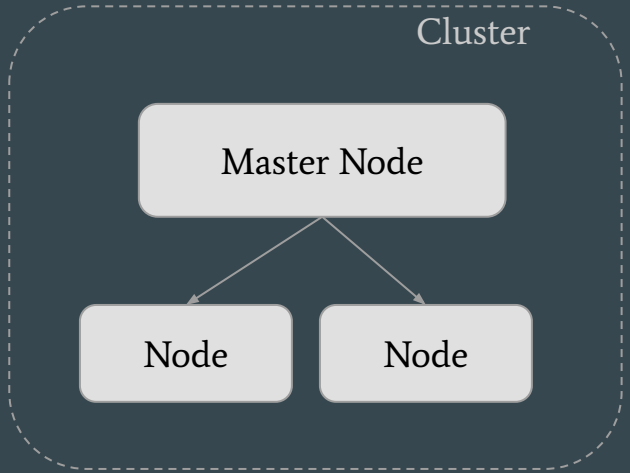
Kubernetes Metamodel

Nodes

Nodes are virtual or bare-metal machines that needs to be interconnected on the same LAN.

There are two types of nodes:

1. ***Master Node*** (or ***Control Plane***), unique inside of a cluster. The Control Plane manages all the resources deployed inside the cluster.
2. ***Worker Node***, they can be 0, 1 or many inside of a cluster. Their purpose is to offer their computational resources to the cluster.



Kubernetes is composed of Resources

Everything inside of a Kubernetes cluster is a ***resource***. Users can create, read, modify and delete them (CRUD operations), typically using ***yaml*** language.

Resources can be split up in three macro-groups:

1. **Workload resources**
2. **Storage resources**
3. **Access Control resources**

Kubernetes Resources

Namespaces

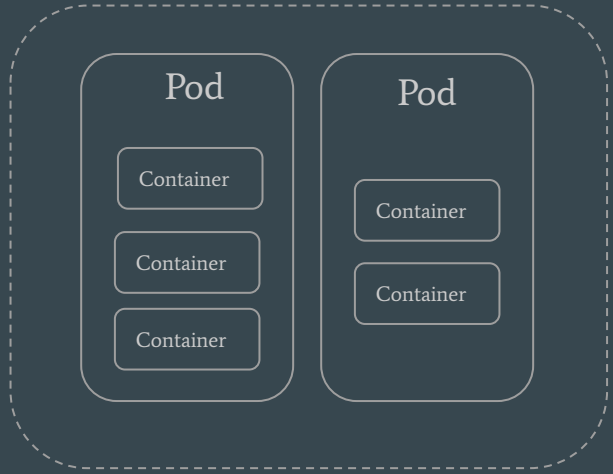
Cluster resources are scoped inside of a Namespace. A Namespace could be see as a virtual cluster where all the resources inside of it needs to have a unique name.

Namespaces are resources themself that users with a proper ***Role*** can interact with.

Pods

Pods are the computing units that are deployed inside of the cluster. All the applications deployed inside of a K8s cluster runs on Pods.

More technically, Pods are composed by many containers that together expose a single service.



Workload Resources

A Workload is an application running on Kubernetes.

Inside of a cluster, a Workload runs on the Pod resource.

There are 5 main ***kinds*** of Workload resources:

1. ***Deployment***
2. ***ReplicaSet***
3. ***StatefulSet***
4. ***DaemonSet***
5. ***Job and CronJob***

Yaml definition of a Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

Services

Services are the resources used to expose an application running on a set of Pods as a ***Network Service***. They're useful because they keep track of IP address changes of the pods and can expose them to the outside of the cluster. They can provide this feature because they look for a label that was put on Pods (see yaml `selection.matchLabels`).

Access Control Resources

In order to use the cluster, the administrator needs to provide a ***ServiceAccount*** resource for each user that wants to operate inside of it. A ServiceAccount is a Namespaced resource, so its user can see only resources inside of it.

In order to protect the access to K8s resources, the administrator decide which operations each ServiceAccount can do inside of the cluster with an RBAC configuration. The RBAC resources are:

1. ***Role***
2. ***ClusterRole***
3. ***RoleBinding***
4. ***ClusterRoleBinding***

Storage Resources

The containers inside of each Pod can use some shared storage, this can be done using ***Volumes***. The Volume abstraction is simply an external directory accessible inside of a Pod, configured during its creation.

There are several types of Volumes, most of them provided from the underlying cloud infrastructure or as external plugins. The types covered from this project are the “native” ones that comes with a bare-metal Kubernetes cluster installation, but they’re also limited compared to the other ones:

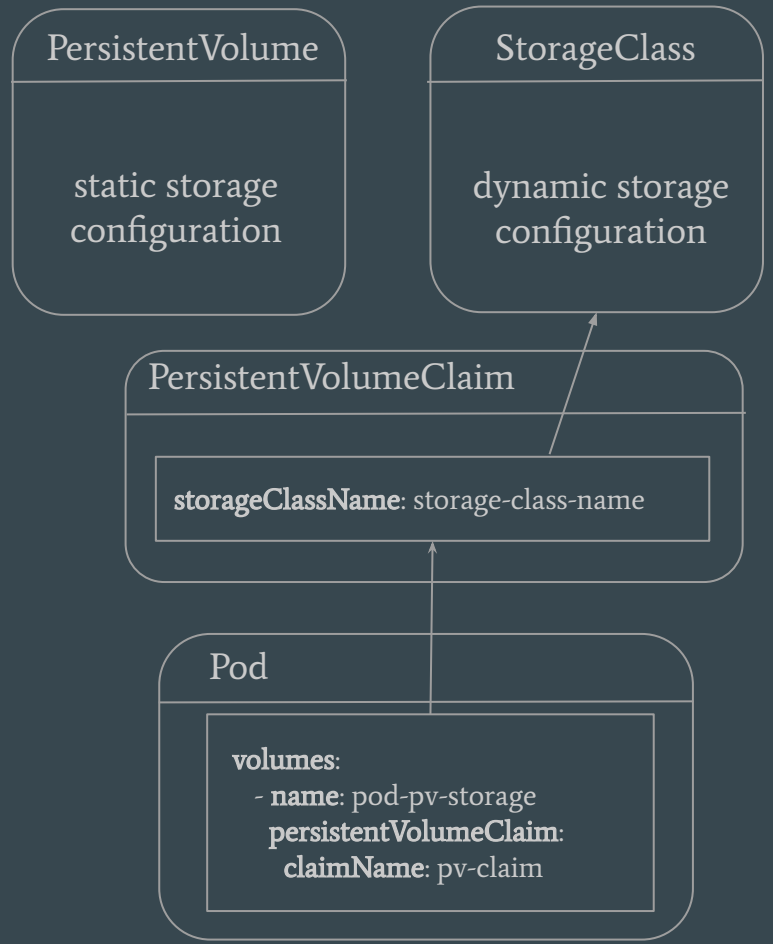
- They’re not available to be read from Pods inside of other nodes
- They can’t be provisioned dynamically with ***StorageClass*** resources

PersistentVolumes

The Persistent Volume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed.

They are themself Volume plugins but also they have a lifecycle independent of any individual Pod that uses them.

StorageClasses allows storage volumes to be created on-demand.



ResourceQuotas and LimitRange

Example of a ResourceQuota:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-resourcequota
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Example of a LimitRange:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: example-limitrange
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
```

ResourceQuota is a K8s resource aimed at define how many resources each pod can use inside of the namespace. To prevent that one single pod could monopolize all the available resources, K8s provide also the ***LimitRange*** resources.

Grazie per l’attenzione :)