
Containers
Containers provide isolation similar to virtual machines (VMs), except provided by the OS and at the process level. Each container is a process or group of processes that are run in isolation. Typical containers explicitly run only a single process because they have no need for the standard system services. What they usually need to do can be provided by system calls to the base OS kernel.

The isolation on Linux is provided by a feature called namespaces. Each different kind of isolation, that is, user and cgroups, is provided by a different namespace.

This is a list of some of the namespaces that are commonly used and visible to the user:

PID:	process IDs
USER:	user and group IDs
UTS:	host name and domain name
NS:	mount points
NET:	network devices, stacks, and ports
CGROUPS:	control limits and monitoring of resources

POD:
A Pod is a group of one or more containers that operate together. Pods reside on Nodes; more than one Pod can share the same Node. The containers within each Pod share common networking and storage resources from that host Node, as well as specifications that determine how the containers run
----
ReplicaSet & Replica Controller:

Replica Set ensures how many replica of pod should be running. It can be considered as a replacement of replication controller. The key difference between the replica set and the replication controller is, the replication controller only supports equality-based selector whereas the replica set supports set-based selector

----
DEPLOYMENT:

A Deployment provides declarative updates for pods and replicas. The Deployment object defines the strategy for transitioning between deployments of the same application

Deployment Strategies:
Deployment strategies help in defining how the new RC should replace the existing RC.

Recreate − This feature will kill all the existing RC and then bring up the new ones. This results in quick deployment however it will result in downtime when the old pods are down and the new pods have not come up.

Rolling Update − This feature gradually brings down the old RC and brings up the new one. This results in slow deployment, however there is no deployment. At all times, few old pods and few new pods are available in this process.

NAMESPACES:

Kubernetes supports multiple virtual clusters backed by the same physical cluster.

These virtual clusters are called namespaces. Within the same namespace, kubernetes objects name should be unique. Different objects in different namespaces may have the
same name.
Kubernetes comes with three initial namespaces

default: the default namespace for objects with no other namespace
kube-system the namespace for objects created by the kubernetes system
kube-public The namespace is created automatically and readable by all users

----

Users in Kubernetes
In Kubernetes there are two categories of users:

Service accounts managed by Kubernetes
Normal users – Outside of kubernetesAn admin distributing private keys, a user store like Keystone or Google Accounts,even a file with a list of usernames and passwords.

Service accounts are users managed by the Kubernetes API. They are bound
to specific namespaces, and created automatically by the API server or manually through API calls. Service accounts are tied to a set of credentials stored as Secrets,
which are mounted into pods allowing in-cluster processes to talk to the Kubernetes API.

LABELS:

In Kubernetes, labels are a system to organize objects into groups. Labels are key-value pairs that are attached to each object.
Label selectors can be passed along with a request to the apiserver to retrieve a list of
objects which match that label selector

DEAMONS;

A Daemon Set is a controller type ensuring each node in the cluster runs a pod. As new
node is added to the cluster, a new pod is added to the node. As the node is removed
from the cluster, the pod running on it is removed and not scheduled on another node.
Deleting a Daemon Set will clean up all the pods it created

VOLUMES:

Containers are ephemeral, meaning the container file system only lives as long as the
container does. Volumes are simplest way to achieve data persistance. In kubernetes, a
more flexible and powerful model is available.

Persistent Volume :

it models shared storage that has been provisioned by the
cluster administrator. It is a resource in the cluster just like a node is a
cluster
resource. Persistent volumes are like standard volumes, but having a lifecycle
independent of any individual pod. Also they hide to the users the details of the
implementation of the storage, e.g. NFS, iSCSI, or other cloud storage systems

Persistent Volume Claim:
it is a request for storage by a user. It is similar to a pod.
Pods consume node resources and persistent volume claims consume persistent
volume objects. As pods can request specific levels of resources like cpu and
memory, volume claimes claims can request the access modes like read-write or
read-only and stoarage capacity.

Kubernetes provides two different ways to provisioning storage:
Manual Provisioning: the cluster administrator has to manually make calls to the
storage infrastructure to create persisten volumes and then users need to create
volume claims to consume storage volumes.

Dynamic Provisioning: storage volumes are automatically created on-demand
when users claim for storage avoiding the cluster administrator to pre-provision
storage.

LIVENESSPROBE:
The kubelet uses liveness probes to know when to restart a Container. liveness
probes could catch a deadlock, where an application is running, but unable to make
progress. Restarting a Container in such a state can help to make the application
more
available despite bugs

READYNESS PROBE:
The kubelet uses readiness probes to know when a Container is ready to start
accepting traffic. A Pod is considered ready when all of its Containers are ready.
One
use of this signal is to control which Pods are used as backends for Services. When
a
Pod is not ready, it is removed from Service load balancers.

CONFIGMAPS:

ConfigMaps allow you to decouple configuration artifacts from image content to keep
containerized applications portable

SECRETS

Objects of type secret are intended to hold sensitive information, such as
passwords,
OAuth tokens, and ssh keys. Putting this information in a secret is safer and more
flexible than putting it verbatim in a pod definition or in a docker image

----
restartPolicy: Never →This condition of image restart is given as never which means
that if the container is killed or if it is false, then it will not restart itself.

SERVICE:

A service can be defined as a logical set of pods. It can be defined as an abstraction on the top of the pod which provides a single IP address and DNS name by which pods can be accessed. With Service, it is very easy to manage load balancing configuration. It helps pods to scale very easily.

A service is a REST object in Kubernetes whose definition can be posted to Kubernetes apiServer on the Kubernetes master to create a new instance.

Types of sevices:

ClusterIP: Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster. This is the default ServiceType.

NodePort: Exposes the service on each Node’s IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service will route, is automatically created. You’ll be able to contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

LoadBalancer: Exposes the service externally using a cloud provider’s load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created.

ExternalName: Maps the service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

JOBS:

A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created.

What is Kubernetes:

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Why do I need Kubernetes and what can it do
Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to restart. Wouldn’t it be easier if this behavior was handled by a system?

That’s how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of your scaling requirements, failover, deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

Service discovery and load balancing
Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
Storage orchestration
Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
Automated rollouts and rollbacks
You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
Automatic bin packing
Kubernetes allows you to specify how much CPU and memory (RAM) each container needs. When containers have resource requests specified, Kubernetes can make better decisions to manage the resources for containers.
Self-healing
Kubernetes restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to clients until they are ready to serve.
Secret and configuration management
Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

PROMETHEUS:

Prometheus is an open source toolkit to monitor and alert, inspired by Google Borg Monitor. It was previously developed by SoundCloud and afterwards donated to the CNCF.

monitoring Areas

CPU
Memory
network
disk IO
System load

Grafana --The leading open source Software for time series analytics, Grafana - Used to visualise the data from Prometheus.

Alert configurations on below parameters

All the severity can get changed since we are not really using it for our filtering.

```
{
DeploymentReplicasNotUpdated -> this is for when the pod did not start at all for any reason.
severity: warning
}

{
PodFrequentlyRestarting -> if any pods restart more than 5 times we send an alert
severity: warning
}

{
DaemonSetRolloutStuck: when we missing pod for any DaemonSet config
severity: warning
}
{
DaemonSetsMissScheduled:A number of daemonsets are running where they are not
supposed to run.
severity: warning
}
{
DeploymentGenerationMismatch: Observed deployment generation does not match
expected one for deployment
severity: warning
}

{
FdExhaustionClose; instance will exhaust in file/socket descriptors within the next
4 hours
severity: warning
}

{
FdExhaustionClose: instance will exhaust in file/socket descriptors within the next
hour
severity: critical
}

{
NodeCPUUsage: how much each nodes using cpu
severity: critical
}
{
NodeDiskRunningFull: Node Disk is running full within the next 24 hours
severity: warning
}

{
NodeDiskRunningFull: Node Disk is running full within the next 2 hours
severity: critical
}


----Grafana json dash board file :
https://github.com/arpankhagram/Youtube/blob/master/Generic_Node1.json
```