Introduction to
Kubernetes

Vikram
IoT Application Dev

git:kunchalavikram1427

Contents

- Kubernetes Introduction
- Installation
- Architecture
- Pods and Init Containers
- ReplicaSets & Deployments
- Static Pods & DaemonSets
- Services
- Ingress Controllers
- Scheduling
- Storage
- Networking
- Monitoring

git:kunchalavikram1427

Container Orchestration

- Container orchestration automates the deployment, management, scaling, and networking of containers across the cluster. It is focused on managing the life cycle of containers.
- Enterprises that need to deploy and manage hundreds or thousands of Linux containers and hosts can benefit from container orchestration.
- Container orchestration is used to automate the following tasks at scale:

  - ✓ Configuring and scheduling of containers
  - ✓ Provisioning and deployment of containers
  - ✓ Redundancy and availability of containers
  - ✓ Scaling up or removing containers to spread application load evenly across host infrastructure
  - ✓ Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
  - ✓ Allocation of resources between containers
  - ✓ External exposure of services running in a container with the outside world
  - ✓ Load balancing of service discovery between containers
  - ✓ Health monitoring of containers and hosts

git.kunchalavikram1427

Container orchestration

Docker Swarm

- Docker Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker
- Any software, services, or tools that run with Docker containers run equally well in Swarm
- It is an alternative to Kubernetes, manages containers and turns the desired state into reality
- It also fixes any future deviations from the desired state
- Docker CLI manages creation of a swarm, deploy application services to a swarm, and manage swarm behaviour

git:kunchalavikram1427

https://github.com/kunchalavikram1427/Docker_public/blob/master/Docker_Made_Easy.pdf

Docker Swarm

- Run containers in Docker Swarm
  https://github.com/kunchalavikram1427/Docker_public/blob/master/Docker_Made_Easy.pdf

- More on Load Balancing and DNS Service Discovery in Docker Swarm using HAProxy
  https://www.haproxy.com/blog/haproxy-on-docker-swarm-load-balancing-and-dns-service-discovery/

git:kunchalavikram1427

Docker Swarm vs Kubernetes

Both Kubernetes and Docker Swarm are two of the most used open-source orchestration platforms providing much of the same functionalities. However, there are several notable differences between them

Features	Kubernetes	Docker Swarm
Installation & Cluster Configuration	Installation is complicated; but once setup, the cluster is very strong	Installation is very simple; but cluster is not very strong
GUI	GUI is the Kubernetes Dashboard	There is no GUI, only 3rd party tools
Scalability	Highly scalable & scales fast	Highly scalable & scales 5x faster than Kubernetes
Auto-Scaling	Kubernetes can do auto-scaling	Docker Swarm cannot do auto-scaling
Rolling Updates & Rollbacks	Can deploy Rolling updates & does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
Data Volumes	Can share storage volumes only with other containers in same Pod	Can share storage volumes with any other container
Logging & Monitoring	In-built tools for logging & monitoring	3rd party tools like ELK should be used for logging & monitoring

git:kunchalavikram1427

Kubernetes

- Kubernetes, also known as K8s, is an open-source Container Management tool
- It provides a container runtime, container orchestration, container-centric infrastructure orchestration, self-healing mechanisms, service discovery, load balancing and container (de)scaling.
- Initially developed by Google, for managing containerized applications in a clustered environment but later donated to CNCF
- Written in Golang
- It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability
- Kubernetes has many moving parts and there are countless ways to configure its pieces - from the various system components, network transport drivers, CLI utilities not to mention applications and workloads

git.kunchalavikram1427

kubernetes

Kubernetes

Certified Kubernetes Distributions

- Cloud Managed: EKS by AWS, AKS by Microsoft and GKE by google
- Self Managed: OpenShift by Redhat and Docker Enterprise
- Local dev/test: Micro K8s by Canonical, Minikube
- Vanilla Kubernetes: The core Kubernetes project(BareMetal), Kubeadm
- Special builds: K3s by Rancher, a light weight K8s distribution

git:kunchalavikram1427

https://www.cncf.io/certification/software-conformance/

Kubernetes Architecture

Cluster Setup

- There are different ways to bootstrap a cluster, like Minikube, Kind, MicroK8s, K3s, Kubeadm, Docker desktop, KOPS and many more
- We will see how to bootstrap a cluster using Kubeadm method

Kubeadm

ONLINE METHOD

https://github.com/kunchalavikram1427/Kubernetes_public/blob/master/Bootstrap_K8s_Cluster_Kubeadm.pdf

OFFLINE METHOD

https://github.com/kunchalavikram1427/Kubernetes_public/blob/master/Kubernetes_offline_installation_RHEL7.pdf

git:kunchalavikram1427

https://kubernetes.io/docs/tasks/tools/

Kubernetes Cluster

- A Kubernetes cluster is a set of physical or virtual machines and other infrastructure resources that are needed to run your containerized applications. Each machine in a Kubernetes cluster is called a node
- A node is the smallest unit of computing hardware in Kubernetes, likely be either a physical machine in a datacenter, or virtual machine hosted on a cloud provider like AWS, Azure, GCP or even small computing devices like RaspberryPi
- There are two types of node in each Kubernetes cluster:
  Master node(s): hosts the Kubernetes control plane components and manages the cluster
  Worker node(s): runs your containerized applications

git:kunchalavikram1427

Kubernetes Architecture

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes Architecture

git:kunchalavikram1427

https://blog.alexellis.io/kubernetes-in-10-minutes/

Kubernetes Architecture

Master Node

- Master is responsible for managing the complete cluster.
- You can access master node via the CLI, GUI, or API
- The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes
- For achieving fault tolerance, there can be more than one master node in the cluster – High Availability cluster setup
- It is the access point from which administrators and other users interact with the cluster to manage the scheduling and deployment of containers.
- It has four components: ETCD, Scheduler, Controller and API Server together known as Control Plane

git:kunchalavikram1427

https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Architecture

Master Node

API server

- Exposes k8s APIs and serves like front end for the control plane
- Masters communicate with the rest of the cluster through the kube-apiserver
- It validates and executes user's REST commands
- kube-apiserver also makes sure that configurations in etcd match with configurations of containers deployed in the cluster.

git:kunchalavikram1427

Controller manager

- The controllers are the brain behind orchestration
- They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases
- The kube-controller-manager runs control loops that manage the state of the cluster by checking if the required deployments, replicas, and nodes are running in the cluster
- Ex: Node controller, Replication controller, Endpoint controller etc

Kubernetes Architecture

Master Node

ETCD

- ETCD is a distributed, reliable key-value store used by Kubernetes to store all data used to manage the cluster
- When you have multiple nodes and multiple masters in your cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner
- ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters

git:kunchalavikram1427

Scheduler

- The scheduler is responsible for distributing work or containers across multiple nodes
- It looks for newly created containers and schedules them onto the Nodes
- Scheduling decisions include factors such as : pod resource requirements, hardware/software/policy constraints, affinity and anti-affinity rules, taints and tolerations etc

Kubernetes Architecture

Worker Nodes

Kubelet

- Worker nodes have the kubelet agent that is responsible for interacting with the master to provide pod & health information of the worker node
- Carry out actions requested by the master on the worker nodes
- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy
- The kubelet doesn't manage containers which were not created by Kubernetes. Ex: Containers created using docker commands

git:kunchalavikram1427

Kube proxy

- kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept
- It is responsible for ensuring network traffic is routed properly to internal and external services as required and is based on the rules defined by network policies in kube-controller-manager and other custom controllers

Kubernetes Architecture

In short

- Kubernetes consists of a set of independent components that run as separate processes on the nodes of a cluster. Some components run on the master nodes and others run on the worker nodes, and each component has a very specific function.

- These are the most important components on the master nodes:
  1. **Storage backend**: stores resource definitions (usually etcd is used)
  2. **API server**: provides Kubernetes API and manages storage backend
  3. **Controller manager**: ensures resource statuses match specifications
  4. **Scheduler**: schedules Pods to worker nodes

- And this is the most important component on the worker nodes:
  1. **Kubelet**: manages execution of containers on a worker node
  2. **Container Runtime**: Runs containerized applications
  3. **Kube-proxy**: runs on each worker node as the network proxy. It listens to the API server for each service point creation or deletion. For each service point, kube-proxy sets the routes so that it can reach to it. In simple, it just redirects requests from Services to Pod IP

git:kunchalavikram1427

Kubernetes Architecture

/kunchalavikram1427

Kubernetes Architecture

Kubectl

- kubectl is the command line utility using which we can interact with k8s cluster, like a client
- Kubernetes is fully controlled through its REST API
- Every Kubernetes operation is exposed as an API endpoint and can be executed by an HTTP request to this endpoint
- Kubectl uses these APIs to interact with the cluster
- Can deploy and manage applications on a Kubernetes

git:kunchalavikram1427

- `kubectl run nginx` deploy an application to the cluster
- `kubectl cluster-info` view information about the cluster
- `kubectl get nodes` list all the nodes that are part of the cluster
- `kubectl get componentstatuses` get health status of control plane components

Kubernetes Architecture

Container Runtime Environment

- Containers are not first class objects in the Linux kernel
- Containers are fundamentally composed of several underlying kernel primitives: namespaces (who you are allowed to talk to), cgroups (the amount of resources you are allowed to use), and LSMs (Linux Security Modules—what you are allowed to do). Together, these kernel primitives allow us to set up secure, isolated, and metered execution environments for our processes
- Creating these environment manually each time we want to create a new isolated process would be tiresome and error prone
- To avoid this, all the components have been bundled together in a concept called a container
- The container runtime is the software that is responsible for running these containers
- The runtime executes the container, telling the kernel to assign resource limits, create isolation layers (for processes, networking, and filesystems), and so on, using a cocktail of mechanisms like control groups (cgroups), namespaces, capabilities, SELinux etc
- For Docker, `docker run` is what creates and runs the container, behind the scenes it is runc that is doing the process
- Kubernetes supports several container runtimes: Docker, containerd, CRI-O, rtk etc

git:kunchalavikram1427

Kubernetes Architecture

Container Runtime Environment

git:kunchalavikram1427

Kubernetes Architecture

CRI (Container Runtime Interface)

- CRI was introduced in Kubernetes 1.5 and acts as a bridge between the kubelet and the container runtime
- High-level container runtimes that want to integrate with Kubernetes are expected to implement CRI. The runtime is expected to handle the management of containers, pods, images etc
- When kubelet wants to run the workload, it uses CRI to communicate with the container runtime running on that same node
- In this way, CRI is simply an abstraction layer or API that allows you to switch out container runtime implementations instead of having them baked into the kubelet
- K8s after trying to support multiple versions of kubelet for different container runtime environments, and trying to keep up with the Docker interface changes, it decided to set a standard interface(CRI) to be implemented by all container runtimes
- This is to avoid large codebase for kubelet for supporting different Container Runtimes
- To implement a CRI, a container runtime environment must be compliant with the Open Container Initiative (OCI)
- OCI includes a set of specifications that container runtime engines must implement and a seed container runtime engine called runc, s a CLI tool for spawning and running containers according to the OCI specification

git:kunchalavikram1427

Kubernetes Architecture

CRI (Container Runtime Interface)

/kunchalavikram1427

Kubernetes Architecture

Kubernetes deprecated Docker

- Kubernetes is deprecating Docker as a container runtime after version 1.20, in favor of runtimes like *containerd* that use the Container Runtime Interface(CRI) created for Kubernetes
- Kubernetes is actually deprecating dockershim, which is a component in Kubernetes' kubelet implementation, communicating with Docker Engine
- Docker does not support Kubernetes Runtime API called CRI(Container Runtime Interface) and Kubernetes have been using a bridge service called dockershim. It converts Docker API and CRI, but it will no longer be provided from Kubernetes side within a few minor releases
- Kubernetes actually needs only container runtime. It doesn't need extra features provided by Docker like Docker Networks and Volumes which are never used by K8s and having them could pose a security risk. The less features you have, the smaller the attack surface becomes

gir:kunchalavikram1427

Docker support in the kubelet is now deprecated and will be removed in a future release. The kubelet uses a module called **dockershim** which implements CRI support for Docker, and it has seen maintenance issues in the Kubernetes community. It is advised to evaluate moving to a container runtime that is a full-fledged implementation of CRI (v1alpha1 or v1 compliant) as they become available.

Kubernetes Architecture

Kubernetes deprecated Docker

Kubernetes using Docker

Kubernetes without Dockershim

git:kunchalavikram1427

With docker being deprecated, we should now use container runtimes like containerd, rkt, cri-o which supports container runtime interfaces developed for k8s

Kubernetes Architecture

Kubernetes deprecated Docker

git:kunchalavikram1427

With docker being deprecated, we should now use container runtimes like containerd, rkt, cri-o which supports container runtime interfaces developed for k8s

Kubernetes Architecture

Kubernetes deprecated Docker

- Developers can still use the Docker platform to build, share, and run containers on Kubernetes! This change primarily impacts operators and administrators for Kubernetes and doesn't impact developer work flows. The images Docker builds are compliant with OCI (Open Container Initiative), are fully supported on containerd, and will continue to run great on Kubernetes
- The OCI runtime specification governs how containers are expressed at runtime, and almost every container engine on the planet uses runc which is not only the reference implementation, but also the gold standard for communicating with the Linux kernel to start containers
- If you're using Docker, you're already using containerd. Docker's runtime is build upon containerd while providing a great developer experience around it. Production environments may have no need for Docker's great developer experience, it's reasonable to directly use lightweight runtimes like containerd
- Docker set up in 2015 the Open Container Initiative (OCI) in order to support fully interoperable container standards, and created the containerd project, along with Google and IBM, in 2016, with the goal of this transition in mind
- Containerd was donated to the CNCF in 2017, and has grown to incorporate the containerd CRI project to interface with Kubernetes, as well as seeing a host of innovation and investment from across the industry, including from Amazon, Google, Microsoft and IBM

github.com/kunchalavikram1427

git:kunchalavikram1427

Pods

Kubernetes

Pods

- Basic scheduling unit in Kubernetes. Pods are often ephemeral
- Kubernetes doesn't run containers directly; instead it wraps one or more containers into a higher-level structure called a pod
- It is also the smallest deployable unit that can be created, scheduled, and managed on a Kubernetes cluster. Each pod is assigned a unique IP address within the cluster
- Pods can hold multiple containers as well, but you should limit yourself when possible. Because pods are scaled up and down as a unit, all containers in a pod must scale together, regardless of their individual needs. This leads to wasted resources.

git:kunchalavikram1427

Ex: nginx, mysql, wordpress..

10.244.0.22

/kunchalavikram1427

Kubernetes

Pods

- A pod can have one or more tightly related containers that will always run together on the same worker node and in the same Linux namespace
- Each pod is like a separate logical machine with its own IP, hostname, processes, and so on, running a single application
- All the containers in a pod will appear to be running on the same logical machine, whereas containers in other pods, even if they're running on the same worker node, will appear to be running on a different one

git:kunchalavikram1427

Kubernetes

Pods

- Any containers in the same pod will share the same storage volumes and network resources and communicate using localhost
- K8s uses YAML to describe the desired state of the containers in a pod. This is also called a Pod Spec. These objects are passed to the kubelet through the API server
- Pods are used as the unit of replication in Kubernetes. If your application becomes too popular and a single pod instance can't carry the load, Kubernetes can be configured to deploy new replicas of your pod to the cluster as necessary.

Using the example from the above figure, you could run curl 10.1.0.1:3000 to communicate to the one container and curl 10.1.0.1:5000 to communicate to the other container from other pods. However, if you wanted to talk between containers - for example, calling the top container from the bottom one, you could use http://localhost:3000.

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes

Scaling Pods

- All containers within the pod get scaled together
- You cannot scale individual containers within the pods. The pod is the unit of scale in K8s
- Recommended way is to have only one container per pod. Multi container pods are very rare
- In K8s, **initcontainer** is sometimes used as a second container inside pod

git:kunchalavikram1427

Kubernetes

Init Containers

- Init containers are exactly like regular containers, except that they always run to completion
- Each init container must complete successfully before the next one starts
- They can contain utilities or custom code for setup that are not present in an app image Ex: sed, awk, python, or dig during setup, also functionalities like Clone a Git repository into a Volume
- If a Pod's init container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds
- However, if the Pod has a restartPolicy set to Never, and an init container fails during startup of that Pod, Kubernetes treats the overall Pod as failed
- Init containers do not support lifecycle, livenessProbe, readinessProbe, or startupProbe because they must run to completion before the Pod can be ready
- If you specify multiple init containers for a Pod, kubelet runs each init container sequentially.
- Each init container must succeed before the next can run
- When all of the init containers have run to completion, kubelet initializes the application containers for the Pod and runs them as usual
- Because init containers run to completion before any app containers start, init containers offer a mechanism to block or delay app container startup until a set of preconditions are met. Once preconditions are met, all of the app containers in a Pod can start in parallel

git:kunchalavikram1427

https://kubernetes.io/docs/concepts/workloads/pods/init-containers/

Kubernetes

Imperative vs Declarative commands

- Kubernetes API defines a lot of objects/resources, such as namespaces, pods, deployments, services, secrets, config maps etc
- There are two basic ways to deploy these objects in Kubernetes: Imperatively and Declaratively

Imperatively

- Involves using any of the verb-based commands like kubectl run, kubectl expose, kubectl delete, kubectl scale and kubectl edit
- Suitable for testing and interactive experimentation

git:kunchalavikram1427

Declaratively

- Objects are written in YAML files and deployed using kubectl create or kubectl apply
- Best suited for production environments

Kubernetes

Manifest /Spec file

- K8s object configuration files - Written in YAML or JSON
- They describe the desired state of your application in terms of Kubernetes API objects
- A file can include one or more API object descriptions (manifests)
- Manifest file has 4 mandatory fields as shown below

✓ **apiVersion** - version of the Kubernetes API used to create the object

✓ **kind** - kind of object being created

✓ **metadata** - data that helps uniquely identify the object, including a name and optional namespace

✓ **spec** - configuration that defines the desired for the object

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ...
spec:
  containers:
  - name: ...
---
apiVersion: v1
kind: Pod
metadata:
  name: ...
spec:
  containers:
  - name: ...
```

Multiple resource definitions

git:kunchalavikram1427

Kubernetes

Man Pages

List all K8s API supported Objects and Versions

```
kubectl api-resources
kubectl api-versions
```

Man pages for objects

```
kubectl explain <object>.<option>
kubectl explain pod
kubectl explain pod.apiVersion
kubectl explain pod.spec
```

Terminal

```
$ kubectl api-resources
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
componentstatuses                 cs                                          false        ComponentStatus
configmaps                        cm                                          true         ConfigMap
endpoints                         ep                                          true         Endpoints
events                            ev                                          true         Event
limitranges                       limits                                      true         LimitRange
```

```
$ kubectl api-versions
admissionregistration.k8s.io/v1
admissionregistration.k8s.io/v1beta1
apiextensions.k8s.io/v1
apiextensions.k8s.io/v1beta1
apiregistration.k8s.io/v1
apiregistration.k8s.io/v1beta1
apps/v1
```

```
$ kubectl explain pod or kubectl explain pod.apiVersion
KIND:     Pod
VERSION:  v1

DESCRIPTION:
     Pod is a collection of containers that can run on a host. This resource is
     created by clients and scheduled onto hosts.

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this representation of an
     object. Servers should convert recognized schemas to the latest internal
     value, and may reject unrecognized values. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

   kind <string>
     Kind is a string value representing the REST resource this object
     represents. Servers may infer this from the endpoint the client submits
     requests to. Cannot be updated. In CamelCase. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

   metadata     <Object>
     Standard object's metadata. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

   spec <Object>
     Specification of the desired behavior of the pod. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
```

git:kunchalavikram1427

Kubernetes

Man Pages

Few more...

```
kubectl api-resources --namespaced=true      # All namespaced resources
kubectl api-resources --namespaced=false     # All non-namespaced resources
kubectl api-resources -o name                # All resources with simple output
                                              (just the resource name)
kubectl api-resources -o wide                # All resources with expanded output
kubectl api-resources --verbs=list,get       # All resources that support the
                                              "list" and "get" request verbs
kubectl api-resources --api-group=extensions # All resources in the "extensions"
                                              API group
```

gitkunchalavikram1427

Kubernetes

In the cluster

kubectl get componentstatuses – status of k8s components
kubectl version -o yaml / kubectl version --short

```
$ k version -o yaml
clientVersion:
  buildDate: "2020-07-17T19:00:19Z"
  compiler: gc
  gitCommit: 4c6976793196d70bc5cd29d56ce5440c9473648e
  gitTreeState: clean
  gitVersion: v1.17.9-eks-4c6976
  goVersion: go1.13.9
  major: "1"
  minor: 17+
  platform: windows/amd64
serverVersion:
  buildDate: "2020-07-16T20:46:15Z"
  compiler: gc
  gitCommit: 6f56fa1d68a5a48b8b6fdefa8eb7ead2015a4b3a
  gitTreeState: clean
  gitVersion: v1.18.6+k3s1
  goVersion: go1.13.11
  major: "1"
  minor: "18"
  platform: linux/amd64
```

kubectl get nodes -o wide (or)
kubectl get nodes -o yaml/json – get nodes info in the output format of wide/json/yaml

```
root@k8s-master:/home/osboxes# kubectl get nodes -o wide
NAME          STATUS   ROLES                  AGE   VERSION   INTERNAL-IP       EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-RUNTIME
k8s-master    Ready    control-plane,master   17d   v1.20.2   192.168.107.127   <none>        Ubuntu 18.04.3 LTS   5.0.0-23-generic   docker://20.10.2
k8s-slave01   Ready    <none>                 17d   v1.20.2   192.168.107.90    <none>        Ubuntu 18.04.3 LTS   5.0.0-23-generic   docker://20.10.2
k8s-slave02   Ready    <none>                 17d   v1.20.2   192.168.107.76    <none>        Ubuntu 18.04.3 LTS   5.0.0-23-generic   docker://20.10.2
root@k8s-master:/home/osboxes#
```

kubectl cluster-info

```
root@k8s-master:/home/osboxes# kubectl cluster-info
Kubernetes control plane is running at https://192.168.56.2:6443
KubeDNS is running at https://192.168.56.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

kubectl cluster-info dump --output-directory=/path/to/cluster-state # Dump current cluster state to /path/to/cluster-state

Kubernetes

Creating Pods

kubectl run <pod-name> --image <image-name>

```
kubectl run nginx --image=nginx --dry-run=client
```

dry-run doesn't run the command but will show what resources the command would deploy in the cluster

```
root@k-master:/home/osboxes# kubectl run nginx --image nginx --dry-run=client
pod/nginx created (dry run)
```

```
kubectl run nginx --image=nginx --dry-run=client -o yaml
```

shows the command output in YAML. Shortcut to create a declarative yaml from imperative commands

```
root@k-master:/home/osboxes# kubectl run nginx --image nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

git:kunchalavikram1427

https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Kubernetes

Creating Pods: Imperative Way

`kubectl run nginx-pod --image=nginx --port=80` – Exposes port 80 of the container

`kubectl get pods -o wide` – Get list of Pods

```
root@k8s-master:/home/osboxes# kubectl run nginx-pod --image=nginx
pod/nginx-pod created
root@k8s-master:/home/osboxes# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP            NODE          NOMINATED NODE   READINESS GATES
nginx-pod   1/1     Running   0          22s   10.244.1.83   k8s-slave01   <none>           <none>
root@k8s-master:/home/osboxes#
```

`kubectl describe pod nginx-pod` – display extended information of pod like its IP, Port, Node on which it is scheduled, labels etc

```
root@k8s-master:/home/osboxes# kubectl describe pod nginx-pod
Name:         nginx-pod
Namespace:    default
Priority:     0
Node:         k8s-slave02/192.168.0.103
Start Time:   Fri, 05 Feb 2021 02:50:46 -0500
Labels:       run=nginx-pod
Annotations:  <none>
Status:       Running
IP:           10.244.2.75
IPs:
  IP:  10.244.2.75
```

git:kunchalavikram1427

10.244.2.75

Kubernetes

Creating Pods: Imperative Way

Checking pod logs

kubectl logs nginx-pod – check logs of the pod
kubectl logs -f nginx-pod – check logs of the pod in follow mode(real time)
kubectl logs -f --tail 10 nginx-pod – check last ‘n’ logs of the pod in follow mode

```
root@k8s-master:/home/osboxes/kubernetes# kubectl logs -f --tail 2 nginx-pod
/docker-entrypoint.sh: Configuration complete; ready for start up
10.244.0.0 - - [05/Feb/2021:11:36:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.58.0" "-"
```

Interactive shell access to a running pod

git:kunchalavikram1427

kubectl exec --stdin --tty nginx-pod -- /bin/sh
kubectl exec -it nginx-pod -- /bin/sh

```
root@k8s-master:/home/osboxes/kubernetes# kubectl exec --stdin --tty nginx-pod  -- /bin/sh
# hostname
nginx-pod
#
```

10.244.2.75

Kubernetes

Behind the Scenes

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes

Creating Pods: Imperative Way

- As mentioned earlier, each pod gets its own IP address, but this address is internal to the cluster and isn't accessible from outside of it.
- To make the pod accessible from the outside, you'll expose it through a Service object(to be discussed later)
- From within the cluster, we can curl the IP of the Pod

```
curl 10.244.2.75
```

```
root@k8s-master:/home/osboxes# curl 10.244.2.75
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

git:kunchalavikram1427

Kubernetes

Creating Pods: Declarative Way

- `kubectl create -f pod-definition.yml` – deploy the pod
- `kubectl apply -f pod-definition.yml` – if manifest file is changed/updated after pod deployment and need to re-deploy the manifest again
- `kubectl delete -f pod-definition.yml` – delete the pod deployment

```
root@k8s-master:/home/osboxes/kubernetes# kubectl apply -f pod-definition.yml
pod/nginx-pod created
root@k8s-master:/home/osboxes/kubernetes# kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          6s
root@k8s-master:/home/osboxes/kubernetes# kubectl describe pod nginx-pod
Name:         nginx-pod
Namespace:    default
Priority:     0
Node:         k8s-slave02/192.168.0.103
Start Time:   Fri, 05 Feb 2021 06:52:33 -0500
Labels:       app=myapp
              env=dev
              project=iot
              region=asia
Annotations:  <none>
Status:       Running
IP:           10.244.2.77
IPs:
  IP:  10.244.2.77
```

Terminal

```
$ cat pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: myapp
    env: dev
    project: iot
    region: asia
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

git:kunchalavikram1427

```
# nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
    tier: dev
spec:
  containers:
  - name: nginx-container
    image: nginx
```

Kind	apiVersion
Pod	v1
ReplicationController	V1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1
DaemonSet	apps/v1
Job	batch/v1

Kubernetes

Creating Pods: Declarative Way

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
      protocol: TCP
```

Descriptor conforms to version vl of Kubernetes API

You're describing a pod.

The name of the pod

Container image to create the container from

Name of the container

The port the app is listening on

git:kunchalavikram1427

Kubernetes

Creating Pods: Declarative Way

- We can directly generate the Manifest file from the imperative commands by using dry run and printing the output in yaml format
- Edit the YAML file as required and deploy it using `kubectl create` or `apply`

```
kubectl run nginx-pod --image=nginx --port=80 --dry-run=client -o yaml > pod-manifest.yml (or)
kubectl run nginx-pod --image=nginx --port=80 --dry-run=client -o yaml | tee pod-manifest.yml
```

```
root@k8s-master:/home/osboxes/kubernetes# kubectl run nginx-pod --image=nginx --port=80 --dry-run=client -o yaml > pod-manifest.yml
root@k8s-master:/home/osboxes/kubernetes# cat pod-manifest.yml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx-pod
  name: nginx-pod
spec:
  containers:
  - image: nginx
    name: nginx-pod
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
root@k8s-master:/home/osboxes/kubernetes#
```

git:kunchalavikram1427

Kubernetes

Multicontainer Pods

- The multi-container pods are the pods that contain two or more related containers that share resources like network namespace, shared volumes, Process namespace and work together as a single unit
- Since all the containers inside pod share the same network space, they can easily communicate on the localhost
- With shared process namespace, containers inside the pod can signal with each other. For this to be enabled, we need to have this setting shareProcessNamespace to true in the pod spec
- All the containers can have the same volume mounted so that they can communicate with each other by reading and modifying files in the storage volume. For example, there can be a helper container that pulls the files from the remote repo and updates the storage volume and the main container, which is a web server, serves those static files from the same storage to the outside world

git:kunchalavikram1427

https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/

Kubernetes

Multicontainer Design Patterns

- There are several advantages with multiple containers inside a pod. The commonly used multi-container design patterns include Sidecar pattern and Ambassador/proxy pattern

Sidecar pattern

- The sidecar pattern is a single-pod pattern made up of two containers. The first is the application container. It contains the core logic for the application. Without this container, the application would not exist
- In addition to the application container, there is a sidecar container. The role of the sidecar is to augment and improve the application container, often without the application container's knowledge.
- In its simplest form, a sidecar container can be used to add functionality to the main container that might otherwise be difficult to improve. Ex: Monitoring, logging, configuration, etc

git:kunchalavikram1427

Kubernetes

Multicontainer Design Patterns

- There are many advantages with multiple containers inside a pod. The commonly used multi-container design patterns include Sidecar pattern Ambassador/proxy pattern

Ambassador pattern

- In this pattern, the helper service can send network requests on behalf of the main application. This is considered as a proxy server that can be co-located with the main application.
- This pattern can be useful for offloading common client connectivity tasks such as monitoring, logging, routing, security (such as TLS), and resiliency patterns in a language agnostic way. It is often used with legacy applications, or other applications that are difficult to modify, in order to extend their networking capabilities.
- It can also enable a specialized team to implement those features.

git:kunchalavikram1427

Kubernetes

Multicontainer Pods

- Pod with 2 containers can share same volume space(to be discussed in the volumes section)

```
k apply -f multicontainer-pod-01.yml
k logs -f pod/multi-container-pod -c ubuntu-container-01
k logs -f pod/multi-container-pod -c ubuntu-container-02
```

```
root@k8s-master:/home/osboxes/kubernetes# k apply -f multicontainer-pod-01.yml
pod/multi-container-pod created
root@k8s-master:/home/osboxes/kubernetes# k logs -f pod/multi-container-pod -c ubuntu-container-01
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
hello from container 1 of host multi-container-pod
^C
root@k8s-master:/home/osboxes/kubernetes# k logs -f pod/multi-container-pod -c ubuntu-container-02
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
hello from container 2 of host multi-container-pod
^Xhello from container 2 of host multi-container-pod
^C
root@k8s-master:/home/osboxes/kubernetes#
```

Terminal

```
$ cat multicontainer-pod-01.yml

apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
  - name: ubuntu-container-01
    image: ubuntu
    command: ["/bin/sh"]
    args: ["c", "while true; do echo hello
        from container 1 of host `hostname`
        ; sleep 5;done"]
  - name: ubuntu-container-02
    image: ubuntu
    command: ["/bin/sh"]
    args: ["c", "while true; do echo hello
        from container 2 of host `hostname`
        ; sleep 5;done"]
```

git:kunchalavikram1427

Kubernetes

Run a temp pod

- You can also fire up an interactive Pod within a Kubernetes cluster that is deleted once you exit the interactive session
- `--rm` ensures that the pod is deleted when you exit the interactive shell
- If you omit --rm, you should delete the pod manually with `kubectl delete pod/<pod-name>`
- If you want to detach from the shell and leave it running with the ability to re-attach, omit the --rm. You will then be able to reattach with: `kubectl attach $pod-name -c $pod-container -i -t` after you exit the shell
- -i/--tty: The combination of these two are what allows us to attach to an interactive session
- --: Delimits the end of the kubectl run options from the positional arg (bash)
- bash: Overrides the container's CMD. In this case, we want to launch bash as our container's command

```
kubectl run test-pod --image=alpine --rm -it
kubectl run my-shell --rm -it --image ubuntu -- bash
kubectl run tmp-shell --rm -i --tty --image centos -- /bin/bash
```

git.kunchalavikram1427

```
root@k8s-master:/home/osboxes# kubectl run test-pod --image=alpine --rm -it
If you don't see a command prompt, try pressing enter.
/ # wget -qO- www.nginx.com
<!DOCTYPE html>
<html lang="en-US">
<head>

  <script>
  //some global vars. DO NOT change these variables names. These variables are being used in GTM.
  var NX_GDPR_FUNCTIONAL_COOKIE_CONSENT = NX_GDPR_FUNCTIONAL_COOKIE_CONSENT || "no"; // possible values are 'yes' and 'no'
  var NX_GDPR_SOCIAL_COOKIE_CONSENT = NX_GDPR_SOCIAL_COOKIE_CONSENT || "no"; // possible values are 'yes' and 'no'
  //end global vars
  </script>
```

In the above example, wget will quietly (flag -q) download and output the content of URL to stdout (flag -O -)
We can also use options like timeout wget --timeout=10 -q -O- www.google.com

Kubernetes

Interacting with running Pods

- kubectl logs my-pod # dump pod logs (stdout)
- kubectl logs -l name=myLabel # dump pod logs, with label name=myLabel (stdout)
- kubectl logs my-pod -c my-container # dump pod container logs (stdout, multi-container case)
- kubectl logs -l name=myLabel -c my-container # dump pod logs, with label name=myLabel (stdout)
- kubectl logs -f my-pod # stream pod logs (stdout)
- kubectl logs -f my-pod -c my-container # stream pod container logs (stdout, multi-container case)
- kubectl logs -f -l name=myLabel --all-containers # stream all pods logs with label name=myLabel (stdout)
- kubectl run -i --tty busybox --image=busybox -- sh # Run pod as interactive shell
- kubectl run nginx --image=nginx -n mynamespace # Run pod nginx in a specific namespace
- kubectl attach my-pod -i # Attach to Running Container
- kubectl exec my-pod -- ls / # Run command in existing pod (1 container case)
- kubectl exec --stdin --tty my-pod -- /bin/sh # Interactive shell access to a running pod (1 container case)
- kubectl exec my-pod -c my-container -- ls / # Run command in existing pod (multi-container case)

git:kunchalavikram1427

Kubernetes

Init Containers

- Init container must complete successfully before the next one starts
- In this case, the init container counts from 10 to 0 before the main container executes its process

```
k apply -f initcontainers-03.yml
k logs -f pod/init-containers-pod -c busybox-int
k logs -f pod/init-containers-pod -c busybox-main
```

```
root@k8s-master:/home/osboxes/kubernetes# k apply -f ic.yml
pod/init-containers-pod created
root@k8s-master:/home/osboxes/kubernetes# k logs -f pod/init-containers-pod -c busybox-int
10
9
8
7
6
5
4
3
2
1
0
root@k8s-master:/home/osboxes/kubernetes# k logs -f pod/init-containers-pod -c busybox-main
hello from main container
^C
root@k8s-master:/home/osboxes/kubernetes# kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
init-containers-pod   1/1     Running   0          28s
root@k8s-master:/home/osboxes/kubernetes#
```

https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-initialization/#create-a-pod-that-has-an-init-container

Terminal

```
$ cat initcontainers-03.yml
  apiVersion: v1
  kind: Pod
  metadata:
    name: init-containers-pod
  spec:
    containers:
    - name: busybox-main
      image: busybox
      command: ["/bin/sh"]
      args: ["-c", "echo 'hello from main
            container' ; sleep 10m"]

    # This containers counts from 10 to 1
    initContainers:
    - name: busybox-int
      image: busybox
      command: ["/bin/sh"]
      args: ["-c", 'i=10; while [ $i -ge 0 ]; do
            echo $i; sleep 0.5; i=$(expr $i - 1);
            done']
```

git:kunchalavikram1427

Kubernetes

Kubectl shortcuts

- alias k=kubectl
  - k version --short
- echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently to your bash shell
- source <(kubectl completion bash | sed s/kubectl/k/g) # if k is used as alias for kubectl
- export do="--dry-run=client -o yaml"
  - k run pod1 --image=nginx $do
- Watch the cluster in Realtime: It prints out the current context, namespace and then all possible Kubernetes objects
  - watch -n 0.5 "kubectl config current-context; echo ''; kubectl config view | grep namespace; echo ''; kubectl get namespace,node,ingress,pod,svc,job,cronjob,deployment,rs,pv,pvc,secret,ep -o wide"

git:kunchalavikram1427

The completion script for Bash depends on the bash-completion project, so you must install that first

- apt-get install bash-completion (or) yum install bash-completion
- source /usr/share/bash-completion/bash_completion

Reload the shell after this

Kubernetes

Labels and Selectors

- Even a small Kubernetes cluster may have hundreds of Containers, Pods, Services and many other Kubernetes objects in it
- It quickly becomes annoying to page through pages of kubectl output to find the required object
- Labels and selectors address this issue in k8s

Labels

- Labels are key/value pairs that are attached to objects, such as pods
- Labels allows to logically group certain objects by giving various names to them
- You can label pods, services, deployments and even nodes
- Ex:

```
"release" : "stable", "release" : "canary"
"environment" : "dev", "environment" : "production"
"tier" : "frontend", "tier" : "backend"
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Terminal

```
$ cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
   application-ID: my-app
   version: version-nr
   stage: dev
   release: release-nr
   owner: team-kube
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes

Labels and Selectors

- If labels are not mentioned while deploying k8s objects using imperative commands, the label is auto set as `run: <object-name>`

```
kubectl run nginx-pod --image=nginx
kubectl get pods --show-labels
```

```
root@k8s-master:/home/osboxes# kubectl run nginx-pod --image=nginx
pod/nginx-pod created
root@k8s-master:/home/osboxes# kubectl get pod --show-labels
NAME        READY   STATUS    RESTARTS   AGE   LABELS
nginx-pod   1/1     Running   0          11s   run=nginx-pod
root@k8s-master:/home/osboxes#
```

Adding Labels to running pods

`kubectl label pod nginx-pod environment=dev app=nginx` – added 2 labels to the pod

```
root@k8s-master:/home/osboxes# kubectl label pod nginx-pod environment=dev app=nginx
pod/nginx-pod labeled
root@k8s-master:/home/osboxes# kubectl get pod --show-labels
NAME        READY   STATUS    RESTARTS   AGE   LABELS
nginx-pod   1/1     Running   0          42s   app=nginx,environment=dev,run=nginx-pod
root@k8s-master:/home/osboxes#
```

git:kunchalavikram1427

Adding Labels while creating the pod

`kubectl run nginx-pod --image=nginx -l env=dev,tier=front-end` – added 2 labels to the pod using -l

```
root@k8s-master:/home/osboxes# kubectl get pod --show-labels
NAME        READY   STATUS    RESTARTS   AGE   LABELS
nginx-pod   1/1     Running   0          8s    env=dev,tier=front-end
```

Kubernetes

Labels and Selectors

Delete a Label assigned to a pod

`kubectl delete label -l env-` - deletes label env

```
root@k8s-master:/home/osboxes# kubectl run nginx-pod --image=nginx -l env=dev,tier=front-end
pod/nginx-pod created
root@k8s-master:/home/osboxes# kubectl get po --show-labels
NAME        READY   STATUS              RESTARTS   AGE   LABELS
nginx-pod   0/1     ContainerCreating   0          2s    env=dev,tier=front-end
root@k8s-master:/home/osboxes# kubectl label pod nginx-pod env-
pod/nginx-pod labeled
root@k8s-master:/home/osboxes# kubectl get po --show-labels
NAME        READY   STATUS    RESTARTS   AGE   LABELS
nginx-pod   1/1     Running   0          8s    tier=front-end
root@k8s-master:/home/osboxes#
```

gitkunchalavikram1427

Kubernetes

Annotations

- Kubernetes labels and annotations are ways of adding metadata to Kubernetes object, but the similarities end there
- Kubernetes labels allow you to identify, select and operate on Kubernetes objects, whereas annotations are non-identifying metadata i.e., stuff not used internally by k8s. You can't specify selectors over them within Kubernetes, but they can be used by external tools and libraries
- The metadata to Kubernetes objects can include phone numbers of persons responsible for the object, tool information for debugging purposes, pointers to logging/monitoring, rollout data, git info etc.,
- In short, annotations can hold any kind of information that is useful and can provide context to DevOps teams

Terminal

```
$ cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  annotations:
      gitbranch: master
      prometheus.io/path: /metrics
      prometheus.io/port: "9102"
  labels:
   application-ID: my-app
   stage: dev
   release: release-nr
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

git:kunchalavikram1427

https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/

Kubernetes

Labels and Selectors

Selectors

- Selectors allows to filter the objects based on labels
- The API currently supports two types of selectors: equality-based and set-based
- A label selector can be made of multiple requirements which are comma-separated
- Use the `-l` or `--selector` option to specify these criteria to filter the name of the object

Equality-based Selector

- Three operators used in equality-based selectors are `=` , `==` , `!=`
- The first two represent equality (and are simply synonyms), while the latter represents inequality
- Ex: `env = prod, tier != frontend`
- The first selector above represents key equal to `environment` and `value` equal to `production`. The latter selects all resources with key equal to tier and value distinct from frontend

```
environment = production
tier != frontend
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cuda-test
spec:
  containers:
    - name: cuda-test
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    accelerator: nvidia-tesla-p100
```

git:kunchalavikram1427

Equality-based Selector are used by Replication Controllers and Services

Kubernetes

Labels and Selectors

Set-based Selector

- Set-based label requirements allow filtering keys according to a set of values
- Three kinds of operators are supported: in,notin and exists(only the key identifiers)
- Ex: `env in (prod, qa)`
  `tier notin (frontend, backend)`
  `partition`
  `!partition`

```
environment in (production, qa)
tier notin (frontend, backend)
partition
!partition
```

- Here, the first selector will selects all resources with key equal to environment and value equal to production or qa
- The second selects all resources with key equal to tier and values other than frontend and backend
- The third example selects all resources that includes a label with key partition; no values are checked.
- The fourth checks selects the resources with no partition key
- The set-based label selector is a general form of equality based selector as `environment=production` is equivalent to `environment in (production)` and similarly `!=` is equivalent to `notin`

```
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

git:kunchalavikram1427

Set-based Selector are used ReplicaSets, Deployments, DaemonSets

gh: kunchalavikram1427

Demo – Labels & Selectors

/kunchalavikram1427

Kubernetes

Demo - Labels and Selectors

- Create 5 pods with different labels

```
kubectl run nginx-1 --image=nginx -l env=dev,tier=front-end,ingress=true
kubectl run nginx-2 --image=nginx -l env=dev,tier=back-end,ingress=false
kubectl run nginx-3 --image=nginx -l env=prod,tier=front-end,ingress=true
kubectl run nginx-4 --image=nginx -l env=prod,tier=back-end,ingress=false
kubectl run nginx-5 --image=nginx -l env=qa,tier=db
```

```
root@k8s-master:/home/osboxes# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
nginx-1   1/1     Running   0          18s   env=dev,ingress=true,tier=front-end
nginx-2   1/1     Running   0          17s   env=dev,ingress=false,tier=back-end
nginx-3   1/1     Running   0          17s   env=prod,ingress=true,tier=front-end
nginx-4   1/1     Running   0          17s   env=prod,ingress=false,tier=back-end
nginx-5   1/1     Running   0          17s   env=qa,tier=db
root@k8s-master:/home/osboxes#
```

GitHub/kunchalavikram1427

- Select pods with env = dev

```
kubectl get pods -l "env=dev"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "env=dev"
NAME      READY   STATUS    RESTARTS   AGE
nginx-1   1/1     Running   0          2m1s
nginx-2   1/1     Running   0          2m
root@k8s-master:/home/osboxes#
```

Kubernetes

Demo - Labels and Selectors

- Select pods with env = dev and tier not equal to front-end

```
kubectl get pods -l "env=dev, tier!=front-end"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "env=dev, tier!=front-end"
NAME      READY   STATUS    RESTARTS   AGE
nginx-2   1/1     Running   0          3m29s
root@k8s-master:/home/osboxes#
```

- Select pods with env = dev or qa

```
kubectl get pods -l "env in (dev, qa)"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "env in (dev, qa)"
NAME      READY   STATUS    RESTARTS   AGE
nginx-1   1/1     Running   0          7m37s
nginx-2   1/1     Running   0          7m36s
nginx-5   1/1     Running   0          7m36s
root@k8s-master:/home/osboxes#
```

git:kunchalavikram1427

- Select pods with no label ingress

```
kubectl get pods -l "! ingress"
kubectl get pods -l "env=qa, ! ingress"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "! ingress"
NAME      READY   STATUS    RESTARTS   AGE
nginx-5   1/1     Running   0          8m53s
root@k8s-master:/home/osboxes#
```

Kubernetes

Demo - Labels and Selectors

- Select pods with tier not in front-end or back-end

```
kubectl get pods -l "tier notin (front-end, back-end)"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "tier notin (front-end, back-end)"
NAME      READY   STATUS    RESTARTS   AGE
nginx-5   1/1     Running   0          12m
root@k8s-master:/home/osboxes#
```

- Select pods with key env

```
kubectl get pods -l "env"
```

```
root@k8s-master:/home/osboxes# kubectl get pods -l "tier"
NAME      READY   STATUS    RESTARTS   AGE
nginx-1   1/1     Running   0          13m
nginx-2   1/1     Running   0          13m
nginx-3   1/1     Running   0          13m
nginx-4   1/1     Running   0          13m
nginx-5   1/1     Running   0          13m
root@k8s-master:/home/osboxes#
```

git:kunchalavikram1427

- Delete pods with no ingress

```
kubectl delete pods -l "! ingress"
```

```
root@k8s-master:/home/osboxes# kubectl delete pods -l "! ingress"
pod "nginx-5" deleted
root@k8s-master:/home/osboxes#
```

Kubernetes

The Desired State

- Desired state is one of the core concepts of Kubernetes
- Through a declarative or an imperative API, we describe the state of the objects like Pod, ReplicatSet, Deployment etc., in the cluster
- If, due to some failures, a container stops running, the Kubelet recreates the Pod based on the lines of the desired state stored in etcd
- Kube controller managers in the master are responsible for regulating the state of the system. It they detect any drift in the current state of the cluster, they instruct the kubelets component in the workers to spin up additional pods(depending on desired object) to bring the cluster back to desired state
- So, Kubernetes strictly ensures that all the containers running across the cluster are always in the desired state

git:kunchalavikram1427

Kubernetes

Need for replication in Kubernetes

There are several reason why we need to replicate the containers/applications

Reliability

By having multiple pods for an application, we can prevent problems like application downtimes if one or more containers fails. This is particularly true if the system replaces any containers that fail due to issues with node or the application

Load balancing

Having multiple container enables us to easily send traffic to different instances to prevent overloading a single instance or node(pods spread across the cluster). This is something that Kubernetes does out of the box, making it extremely convenient

dlt.kunchalavikram1427

Scaling

When load does become too much for the number of existing instances, Kubernetes enables us to easily scale up the application, adding additional instances as and when needed

Kubernetes

Replication Controller

- The Replication Controller is the original form of replication in Kubernetes
- Sometimes, a single instance of the application may not be sufficient to handle the growing user traffic to the application
- Also, if this only instance goes down because of a failure, K8s will not bring this pod up again automatically unless it is managed by a controller
- In order to prevent this, we would like to have more than one instance or POD running at all times inside the cluster
- Kubernetes supports different controllers(Replicacontroller & ReplicaSet) to handle multiple instances of a pod. Ex: 3 replicas of nginx webserver
- Replication Controller ensures high availability by replacing the unhealthy/dead pods with a new one to ensure required replicas are always running inside a cluster
- Does this mean you can't use a replication controller if you plan to have a single POD? No! Even if you have a single POD, the replication controller can help by automatically bringing up a new POD when the existing one fails
- Another reason we need replication controller is to create multiple PODs to share the load across them
- Replication controller uses equality-based selectors while ReplicaSets uses set-based selectors
- Replica controller is deprecated and replaced by ReplicaSets

git:kunchalavikram1427

https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/

Kubernetes

ReplicaSets

- ReplicaSets are a higher-level API that gives the ability to easily run multiple instances of a given pod
- ReplicaSets ensures that the exact number of pods(replicas) are always running in the cluster by replacing any failed pods with new ones
- The replica count is controlled by the `replicas` field in the resource definition file
- ReplicaSets uses set-based selectors to manage the pods
- The major difference between a replication controller and replica set is that the rolling-update command works with Replication Controllers, but won't work with a Replica Set. This is because Replica Sets are meant to be used as the backend for Deployments (to be covered next)
- Deployments are recommended over ReplicaSets

git:kunchalavikram1427

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Kubernetes

ReplicaSets

- ReplicaSets ensures that the exact number of replicas/pods mentioned by `replicas` tag are always running in the cluster
- It uses the pod labels as the selector to manage them
- If any pod is deleted, it recreates it automatically and ensures that desired state is always maintained

Terminal

```
$ cat pod.yml
  apiVersion: v1
  kind: Pod
  metadata:
    name: nginx-pod
    labels:
      app: webapp
  spec:
    containers:
    - name: nginx-container
      image: nginx
      ports:
      - containerPort: 80
```

Actual definition of POD itself

Terminal

```
$ cat replica-set.yml
  apiVersion: apps/v1
  kind: ReplicaSet
  metadata:
    name: nginx-replicaset
    labels:
      app: webapp
      type: front-end
  spec:
    replicas: 3
    selector:
      matchLabels:
        app: webapp
    template:
      metadata:
        name: nginx-pod
        labels:
          app: webapp
      spec:
        containers:
          - name: nginx-container
            image: nginx
            ports:
            - containerPort: 80
```

Number of pod replicas

ReplicaSet manages pods with specific labels. Here it manages pods with labels app: webapp

label set to pod

git:kunchalavikram1427

Kubernetes

ReplicaSet Manifest file

Number of pods (replicas)

Which pods to watch?

Pod template to use

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
  labels:
    app: hello
spec:
  replicas: 5
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello-container
        image: busybox
        command: [ ... ]
```

git:kunchalavikram1427

Kubernetes

ReplicaSets

Create ReplicaSet (rs)

```
kubectl create -f replicaset-01.yml
```

Get the current ReplicaSets deployed

```
kubectl get rs -o wide
```

Get all pods

```
kubectl get po -o wide
```

Describe replicaset

```
kubectl describe rs <rs-name>
```

```
root@k8s-master:/home/osboxes# kubectl create -f replicaset-01.yml
replicaset.apps/nginx-replicaset created
root@k8s-master:/home/osboxes# kubectl get rs -o wide
NAME               DESIRED   CURRENT   READY   AGE   CONTAINERS        IMAGES   SELECTOR
nginx-replicaset   3         3         3       52s   nginx-container   nginx    app=webapp
root@k8s-master:/home/osboxes# kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP             NODE          NOMINATED NODE   READINESS GATES
nginx-replicaset-7svd7   1/1     Running   0          65s   10.244.1.100   k8s-slave01   <none>           <none>
nginx-replicaset-9pdc6   1/1     Running   0          65s   10.244.2.91    k8s-slave02   <none>           <none>
nginx-replicaset-lh7qv   1/1     Running   0          65s   10.244.1.99    k8s-slave01   <none>           <none>
root@k8s-master:/home/osboxes# kubectl describe rs nginx-replicaset
Name:         nginx-replicaset
Namespace:    default
Selector:     app=webapp
Labels:       app=webapp
              type=front-end
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=webapp
```

There is no imperative way of creating ReplicaSet as per official documentation. Run `kubectl create --help` for available commands

Terminal

```
$ cat replicaset-01.yml

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: webapp
    type: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      name: nginx-pod
      labels:
        app: webapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
          - containerPort: 80
```

git:kunchalavikram1427

Kubernetes

ReplicaSets

Let's delete a pod manually and see if ReplicaSet recreates it

`kubectl delete pod <pod-name>`

Observe that ReplicaSet recreated the deleted pod with a different hash appended to the pod name and maintained the desired stare

```
root@k8s-master:/home/osboxes# kubectl get po
NAME                      READY   STATUS    RESTARTS   AGE
nginx-replicaset-9pdc6    1/1     Running   0          9m35s
nginx-replicaset-btk58    1/1     Running   0          49s
nginx-replicaset-lh7qv    1/1     Running   0          9m35s
root@k8s-master:/home/osboxes# kubectl delete pod nginx-replicaset-9pdc6
pod "nginx-replicaset-9pdc6" deleted
root@k8s-master:/home/osboxes# kubectl get po
NAME                      READY   STATUS    RESTARTS   AGE
nginx-replicaset-4fc6v    1/1     Running   0          46s
nginx-replicaset-btk58    1/1     Running   0          104s
nginx-replicaset-lh7qv    1/1     Running   0          10m
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME               DESIRED   CURRENT   READY   AGE
nginx-replicaset   3         3         3       10m
root@k8s-master:/home/osboxes#
```

There is no imperative way of creating ReplicaSet as per official documentation. Run `kubectl create --help` for available commands

Terminal

```
$ cat replicaset-01.yml

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: webapp
    type: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      name: nginx-pod
      labels:
        app: webapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
          - containerPort: 80
```

git:kunchalavikram1427

Kubernetes

Resources that support set-based requirements

Newer resources, such as Job, Deployment, ReplicaSet, and DaemonSet, support set-based requirements.

```
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

git:kunchalavikram1427

- **`matchLabels`** is a map of {key,value} pairs
- A single {key,value} in the **`matchLabels`** map is equivalent to an element of **`matchExpressions`**, whose key field is "key", the operator is "In", and the values array contains only "value"
- **`matchExpressions`** is a list of pod selector requirements. Valid operators include **In**, **NotIn**, **Exists**, and **DoesNotExist**
- The values set must be non-empty in the case of In and NotIn. For Exists and DoesNotExists, values filed may be ignored
- All of the requirements, from both **`matchLabels`** and **`matchExpressions`** are ANDed together -- they must all be satisfied in order to match

Kubernetes

Resources that support set-based requirements

- Consider a Pod with only label **`app: webapp`**

```
selector:
  matchLabels:
    app: webapp
    tier: front-end
```

- ReplicaSet will fail to deploy as pod should have both the labels selected through **matchLabels.** Each label is **AND** operated

```
selector:
    matchExpressions:
      - {key: app, operator: In, values: [webapp, nginx]}
```

- ReplicaSet will be deploy as values are **OR** operated. Any matching label should work

git:kunchalavikram1427

```
selector:
    matchExpressions:
      - {key: app, operator: In, values: [webapp, nginx]}
      - {key: tier, operator: Exists}
```

- ReplicaSet will fail as both expressions are **AND** operated and pod has no label called tier

```
selector:
    matchExpressions:
      - {key: app, operator: In, values: [webapp, nginx]}
      - {key: tier, operator: DoesNotExist}
```

- This works as pod has no label called tier and selector doesn't require that label

Kubernetes

ReplicaSets

Scale the ReplicaSet

```
kubectl scale rs nginx-replicaset --replicas=6
```

(or)

```
kubectl scale -f replicaset-01.yml --replicas=10
```

```
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME                DESIRED   CURRENT   READY   AGE
nginx-replicaset    3         3         3       101s
root@k8s-master:/home/osboxes# kubectl scale rs nginx-replicaset --replicas=6
replicaset.apps/nginx-replicaset scaled
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME                DESIRED   CURRENT   READY   AGE
nginx-replicaset    6         6         3       2m11s
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME                DESIRED   CURRENT   READY   AGE
nginx-replicaset    6         6         5       2m12s
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME                DESIRED   CURRENT   READY   AGE
nginx-replicaset    6         6         5       2m13s
root@k8s-master:/home/osboxes# kubectl get rs nginx-replicaset
NAME                DESIRED   CURRENT   READY   AGE
nginx-replicaset    6         6         6       2m14s
root@k8s-master:/home/osboxes#
```

Delete the ReplicaSet

```
kubectl delete rs nginx-replicaset
```

There is no imperative way of creating ReplicaSet as per official documentation. Run `kubectl create --help` for available commands

Terminal

```
$ cat replicaset-01.yml

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: webapp
    type: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      name: nginx-pod
      labels:
        app: webapp
    spec:
      containers:
        - name: nginx-container
          image: nginx
          ports:
          - containerPort: 80
```

git:kunchalavikram1427

Kubernetes

ReplicaSets

- `kubectl edit replicaset <replicaset-name>` - edit a replicaset; like image, replicas
- `kubectl delete -f replica-set.yml`
- `kubectl get all` - get pods, replicasets, deployments, services all in one shot
- `kubectl replace -f replicaset-01.yml` - replaces the replicaset with updated definition file

Can we remove the pod from a ReplicaSets?

Yes, we can. It is as simple as removing the label from the pod and it will be removed from the Set.

git:kunchalavikram1427

Kubernetes Deployments

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes

Deployment

- Deployments are intended to replace Replication Controllers and it provides declarative updates for Pods and ReplicaSets
- Deployments describe the number of desired identical pod replicas to run and the preferred update strategy used when updating the deployment
- Kubernetes will track pod health and will remove or add pods as needed to bring your application deployment to the desired state
- By default, Kubernetes performs deployments in rolling update strategy
- It seems similar to ReplicaSets but with advanced functions
- Deployment is the recommended way to deploy a pod or ReplicaSet
- Below are some of the key features of deployment:
  - ✓ Easily deploy a RS
  - ✓ Rolling updates pods
  - ✓ Rollback to previous deployment versions
  - ✓ Scale deployment
  - ✓ Pause and resume deployment

git:kunchalavikram1427

Kubernetes

Deployment strategies

- Whenever we create a new deployment, K8s triggers a Rollout
- Rollout is the process of gradually deploying or upgrading your application containers
- For every rollout/upgrade, a version history will be created, which helps in rolling back to working version in case of an update failure
- In Kubernetes there are a few different ways to release updates to an application
  - Recreate: terminate the old version and release the new one. Application experiences downtime.
  - Rolling Update: release a new version on a rolling update fashion, one after the other. It's the default strategy in K8s. No application downtime is required.
  - Blue/green: release a new version alongside the old version then switch traffic
  - Canary: release a new version to a subset of users, then proceed to a full rollout
  - A/B testing: release a new version to a subset of users in a precise way (HTTP headers, cookie, weight, etc.). This doesn't come out of the box with Kubernetes, it implies extra work to setup a more advanced infrastructure like Service Mesh (Istio, Linkerd, Traefik, custom nginx/haproxy, etc)

git:kunchalavikram1427

```
spec:
  replicas: 10
strategy:
    type: Recreate
```

```
spec:
  replicas: 10
strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 0
```

https://www.weave.works/blog/kubernetes-deployment-strategies

Kubernetes

Rolling update strategy

- By default, deployment ensures that only 25% of your pods are unavailable during an update and does not update more that 25% of the pods at a given time
- It does not kill old pods until/unless enough new pods come up
- It does not create new pods until a sufficient number of old pods are killed
- There are two settings you can tweak to control the process: `maxUnavailable` and `maxSurge`. Both have the default values set to 25%
- The `maxUnavailable` setting specifies the maximum number of pods that can be unavailable during the rollout process. You can set it to an actual number(integer) or a percentage of desired pods
  - When `maxUnavailable` is set to 40%. When the update starts, the old ReplicaSet is scaled down to 60%.
  - As soon as new pods are started and ready, the old ReplicaSet is scaled down again and the new ReplicaSet is scaled up. This happens in such a way that the total number of available pods (old and new, since we are scaling up and down) is always at least 60%
- The `maxSurge` setting specifies the maximum number of pods that can be created over the desired number of pods
  - If we use the same percentage as before (40%), the new ReplicaSet is scaled up right away when the rollout starts. The new ReplicaSet will be scaled up in such a way that it does not exceed 140% of desired pods. As old pods get killed, the new ReplicaSet scales up again, making sure it never goes over the 140% of desired pods

git.kunchalavikram1427

Kubernetes

Deployment

Create the deployment using imperative command

```
kubectl create deployment nginx --image nginx --replicas=5 --port=80 --dry-run=client – check with dry run
```

deployment name

image name

```
kubectl create deployment nginx --image nginx --replicas=5 --port=80 --dry-run=client
kubectl get deployment nginx – get deployment details; here nginx indicates deployment name
kubectl describe deployment nginx – get detailed information of the deployment
```

```
root@k8s-master:/home/osboxes# kubectl create deployment nginx --image nginx --replicas=5 --port=80
deployment.apps/nginx created
root@k8s-master:/home/osboxes# kubectl get deployment nginx
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   5/5     5            5           26s
root@k8s-master:/home/osboxes# kubectl get po -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP            NODE          NOMINATED NODE
nginx-7848d4b86f-27976   1/1     Running   0          36s   10.244.1.119  k8s-slave01   <none>
nginx-7848d4b86f-b2krr   1/1     Running   0          36s   10.244.2.108  k8s-slave02   <none>
nginx-7848d4b86f-qb4rj   1/1     Running   0          36s   10.244.2.109  k8s-slave02   <none>
nginx-7848d4b86f-t497v   1/1     Running   0          36s   10.244.1.118  k8s-slave01   <none>
nginx-7848d4b86f-xk85m   1/1     Running   0          36s   10.244.1.120  k8s-slave01   <none>
root@k8s-master:/home/osboxes# kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Sun, 07 Feb 2021 05:35:41 -0500
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               5 desired | 5 updated | 5 total | 5 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
```

git:kunchalavikram1427

Kubernetes

Deployment

Get ReplicaSets for the deployment

`kubectl get rs` or `kubectl get all` - to fetch pods, rs, deployments, services etc

```
root@k8s-master:/home/osboxes# kubectl get rs
NAME                 DESIRED   CURRENT   READY   AGE
nginx-7848d4b86f     5         5         5       95s
root@k8s-master:/home/osboxes#
```

Get pods for the deployment

`kubectl get po -o wide`

```
root@k8s-master:/home/osboxes# kubectl get po -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP             NODE
nginx-7848d4b86f-4dvhl    1/1     Running   0          21s   10.244.1.160   k8s-slave01
nginx-7848d4b86f-99gvk    1/1     Running   0          21s   10.244.2.150   k8s-slave02
nginx-7848d4b86f-b8vkv    1/1     Running   0          21s   10.244.2.151   k8s-slave02
nginx-7848d4b86f-krcwj    1/1     Running   0          21s   10.244.1.159   k8s-slave01
nginx-7848d4b86f-v6jmc    1/1     Running   0          21s   10.244.2.152   k8s-slave02
```

Replicas scheduled on to different nodes

/kunchalavikram1427

Kubernetes

Deployment

Scale the deployment to 10 instances

```
kubectl scale deployment nginx --replicas=10
```

```
root@k8s-master:/home/osboxes# kubectl get po
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7848d4b86f-6clnz   1/1     Running   0          29s
nginx-7848d4b86f-6dzl8   1/1     Running   0          29s
nginx-7848d4b86f-6snkd   1/1     Running   0          29s
nginx-7848d4b86f-8spm6   1/1     Running   0          29s
nginx-7848d4b86f-jhs7r   1/1     Running   0          11m
nginx-7848d4b86f-nh6x2   1/1     Running   0          29s
nginx-7848d4b86f-r2hd8   1/1     Running   0          29s
nginx-7848d4b86f-rcr7l   1/1     Running   0          29s
nginx-7848d4b86f-vdv8d   1/1     Running   0          29s
nginx-7848d4b86f-xz9hq   1/1     Running   0          29s
root@k8s-master:/home/osboxes#
```

Edit the deployment

```
kubectl edit deployment nginx – Make live changes to the yaml like increasing replica count
                                or container image etc., and save to reflect
```

git:kunchalavikram1427

Set new image for the containers

```
kubectl set image deployment nginx nginx=nginx:1.16.1 – new image that replaces old image
```

```
root@k8s-master:/home/osboxes# kubectl set image deployment nginx nginx=nginx:1.16.1
deployment.apps/nginx image updated
root@k8s-master:/home/osboxes# kubectl describe deploy nginx | grep -i image
                        kubernetes.io/change-cause: kubectl set image deployment nginx nginx=nginx:1.16.1 --record=true
    Image:        nginx:1.16.1
root@k8s-master:/home/osboxes#
```

Delete the deployment

```
kubectl delete deployment nginx
```

Kubernetes

Deployment

Create the manifest from imperative command

```
kubectl create deployment nginx --image nginx --replicas=5 --port=80 --dry-run=client -o yaml > deployment.yaml
```

```
root@k8s-master:/home/osboxes# kubectl create deployment nginx --image nginx --replicas=5 --port=80 --dry-run=client -o yaml > deployment.yml
root@k8s-master:/home/osboxes# cat deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
        resources: {}
status: {}
root@k8s-master:/home/osboxes#
```

git:kunchalavikram1427

Kubernetes

Deployment

Create the deployment using declarative approach

`kubectl create -f deployment-rolling-update.yml --record` (`--record` will record the events in the deployment)

Status the deployment

`kubectl rollout status deployment nginx`

```
root@k8s-master:/home/osboxes# kubectl apply -f deployment-rolling-update.yml --record
deployment.apps/nginx created
root@k8s-master:/home/osboxes# kubectl get deployment nginx
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   0/5     5            0           2s
root@k8s-master:/home/osboxes# kubectl rollout status deployment nginx
Waiting for deployment "nginx" rollout to finish: 0 of 5 updated replicas are available...
Waiting for deployment "nginx" rollout to finish: 1 of 5 updated replicas are available...
Waiting for deployment "nginx" rollout to finish: 2 of 5 updated replicas are available...
Waiting for deployment "nginx" rollout to finish: 3 of 5 updated replicas are available...
Waiting for deployment "nginx" rollout to finish: 4 of 5 updated replicas are available...
deployment "nginx" successfully rolled out
root@k8s-master:/home/osboxes#
```

Terminal

```
$ cat deployment-rolling-update.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
```

git:kunchalavikram1427

/kunchalavikram1427

Kubernetes

Deployment

Rollout history

`kubectl rollout history deployment nginx` – shows history of all the upgrades to the deployment

```
root@k8s-master:/home/osboxes# kubectl rollout history deployment nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=deployment-rolling-update.yml --record=true
```

Set new image

`kubectl set image deployment nginx nginx=nginx:1.14 --record`

```
root@k8s-master:/home/osboxes# kubectl set image deployment nginx nginx=nginx:1.14 --record
deployment.apps/nginx image updated
root@k8s-master:/home/osboxes# kubectl rollout status deployment nginx
Waiting for deployment "nginx" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx" rollout to finish: 4 of 5 updated replicas are available...
deployment "nginx" successfully rolled out
root@k8s-master:/home/osboxes# kubectl rollout history deployment nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=deployment-rolling-update.yml --record=true
2         kubectl set image deployment nginx nginx=nginx:1.14 --record=true
```

git:kunchalavikram1427

Kubernetes

Deployment

Scale the deployment

```
kubectl scale deployment nginx --replicas=10
```

```
root@k8s-master:/home/osboxes# kubectl scale deployment nginx --replicas=10
deployment.apps/nginx scaled
root@k8s-master:/home/osboxes# kubectl rollout history deployment nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=deployment-rolling-update.yml --record=true
2         kubectl scale deployment nginx --replicas=10 --record=true
```

Undo rollout

```
kubectl rollout undo deployment nginx – this will rollback to previous version
```

git:kunchalavikram1427

```
root@k8s-master:/home/osboxes# kubectl rollout history deployment nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
2         kubectl scale deployment nginx --replicas=10 --record=true
3         kubectl apply --filename=deployment-rolling-update.yml --record=true
```

Rollback to specific version

```
kubectl rollout undo deployment nginx --to-revision=2
```

```
root@k8s-master:/home/osboxes# kubectl rollout undo deployment nginx --to-revision=2
deployment.apps/nginx rolled back
root@k8s-master:/home/osboxes# kubectl rollout history deployment nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
3         kubectl apply --filename=deployment-rolling-update.yml --record=true
4         kubectl scale deployment nginx --replicas=10 --record=true
```

Kubernetes

Deployment

Pause a rollout

```
kubectl rollout pause deployment nginx
```

Delete the deployment

```
kubectl delete -f deployment-rolling-update.yml
```

or

```
kubectl delete deployment nginx
```

git:kunchalavikram1427

Kubernetes

Deployment

git:kunchalavikram1427

Kubernetes

What happens when API Server receives the request

When a command is given through kubectl

git:kunchalavikram1427

Kubernetes

Behind the scene

API Server updates the deployment details in ETCD

git:kunchalavikram1427

Kubernetes

Behind the scene

Controller manager through API Server identifies its workload and creates a ReplicaSet

git:kunchalavikram1427

Kubernetes

Behind the scene

ReplicaSet creates required number of pods and updates the ETCD.
Note the status of pods. They are still in PENDING state

git:kunchalavikram1427

Kubernetes

Behind the scene

Scheduler identifies its workload through API-Server and decides the nodes onto which the pod is to be scheduled.
At this stage, pods are assigned to a node

git:kunchalavikram1427

Kubernetes

Behind the scene

Kubelet identifies its workload through API-Server and understands that it needs to deploy some pods on its node

git:kunchalavikram1427

Kubernetes

Behind the scene

Kubelet instructs the docker daemon to create the pods. At the same time, it updates the status as 'Pods CREATING' to API Server

kubectl create deployment nginx --image nginx --replicas=5

git:kunchalavikram1427

Kubernetes

Behind the scene

Once pods are created and run, Kubelet updates the pod status as RUNNING to API Server

git:kunchalavikram1427

UP NEXT

Kubernetes Services

git:kunchalavikram1427

Subscribe to my Facebook page:
https://www.facebook.com/vikram.devops
and join my group:
https://www.facebook.com/groups/171043094400359
for all latest updates on DevOps, Python and IoT

https://www.youtube.com/channel/UCE1cGZfooxT7-VbqVbuKjMg

git.kunchalavikram1427

References

- Kubernetes 101
  https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16
  https://jamesdefabia.github.io/docs/user-guide/kubectl/kubectl_run/
  https://dev.to/inductor/wait-docker-is-deprecated-in-kubernetes-now-what-do-i-do-e4m
- Kubeadm
  https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
  https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
- Kubectl commands
  https://kubernetes.io/docs/reference/kubectl/cheatsheet/
  https://kubernetes.io/docs/reference/kubectl/overview/
- Deployments
  https://www.bmc.com/blogs/kubernetes-deployment/
  https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- Services
  https://kubernetes.io/docs/concepts/services-networking/service/#headless-services
  https://www.edureka.co/community/19351/clusterip-nodeport-loadbalancer-different-from-each-other
  https://theithollow.com/2019/02/05/kubernetes-service-publishing/
  https://www.ovh.com/blog/getting-external-traffic-into-kubernetes-clusterip-nodeport-loadbalancer-and-ingress/
  https://medium.com/@JockDaRock/metalloadbalancer-kubernetes-on-prem-baremetal-loadbalancing-101455c3ed48
  https://medium.com/@cashisclay/kubernetes-ingress-82aa960f658e
- Ingress
  https://www.youtube.com/watch?v=QUfn0EDMmtY&list=PLVSHGLlFuAh89j0mcWZnVhfYgvMmGI0lF&index=18&t=0s
- K8s Dashboard
  https://github.com/kubernetes/dashboard
  https://github.com/indeedeng/k8dash
- YAML
  https://kubeyaml.com/

git:kunchalavikram1427

Q&A

91:kunchalavikram1427

git:kunchalavikram1427