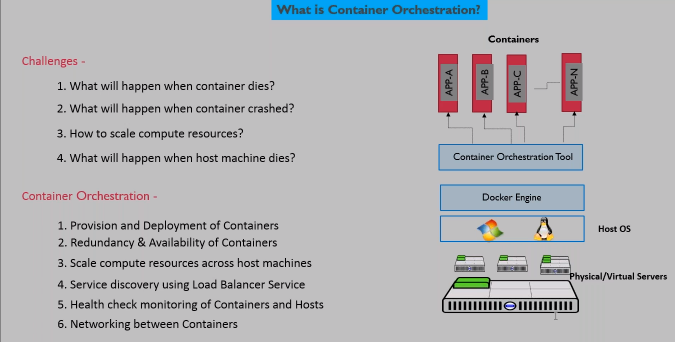
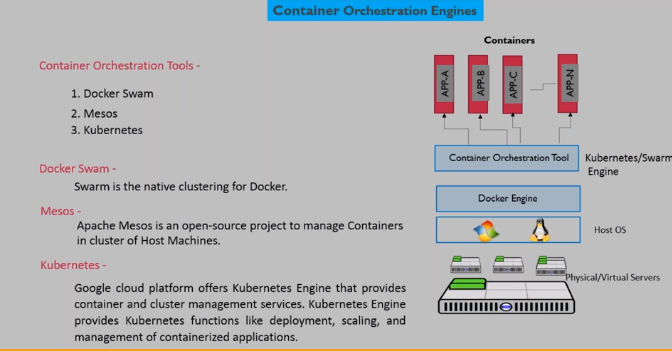
KUBURNETES

**Container Orchestration tool:** It is running on top of the Docker Engine and it will take care of High availability and scalability of the containers. It takes care of managing containers and taking decisions in the container Life cycle.

Example: Kubernetes, Docker Swarm, Mesos





**What is Kubernetes?**

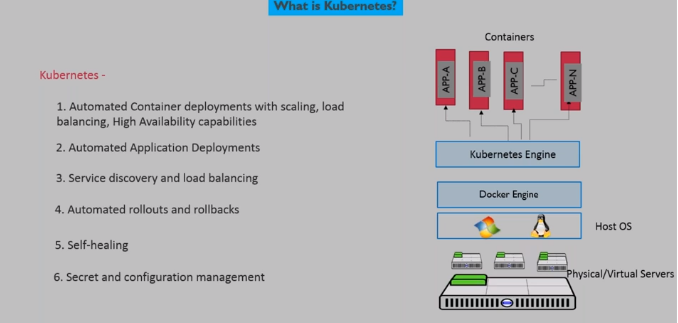
* Kubernetes is an open-source, portable, cluster managed orchestration framework. Kubernetes allows to run containerized applications on multiple clusters for more reliable accessibility and organization.
* Kubernetes supports large scale deployments hence is best suited for enterprise-level containers and cluster management.

**Key Features**

* Automates deployments and Upgrades, and rollbacks.
* Automates containers scalability and availability
* Maintain network setup for all containers within the cluster.
* Monitor service health and replace the containers when unhealthy
* Supports Service discovery and load balancing

**Advantages**

* Provides enterprise-level container and cluster managementservices.
* Adjust the workload without redesigning the application.
* Cost effective solution to deploy microservices
* Flexibility in deploying and managing containers.
* Enhanced portability due to container isolation within the cluster.



**What is Monolithic?**

Monolithic Application describes a single-tiered application software and database interface code that are combined into a single program and deployed into a single platform.

Monolith model deploy all services as single entity in single platform i.e. Monolithic applications have a single code base with multiple modules.

###### What is Monolithic Architecture?

###### 

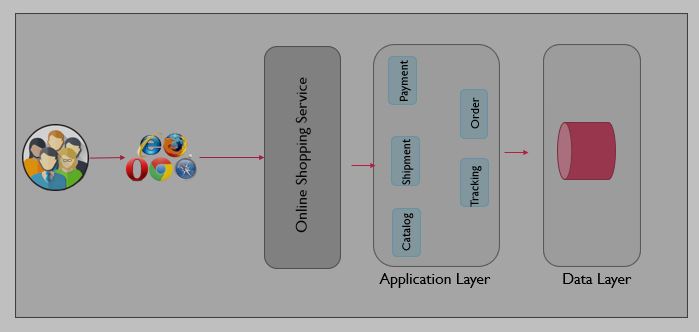
###### Consider an example of an online retail portal, where customers can browse the services through a browser and can add products to the cart, order a product, authorize payments, and track shipments. These are different components involved to make a single web application that the customer uses to purchase a product.

Monolithic Architecture deploys these components into a single unit of application code in a single server or platform.

Application code deployment consists below components:

1. Catalog
2. Order
3. Payment
4. Shipment
5. Tracking

Below Architecture describes the “Monolithic Architecture” view of deploying all service components in single platform as single code or single artifact(.war)

****

###### Monolithic Advantages Vs. Disadvantages

**Advantages**

* Easy Development – Large scale IDEs are supporting the Monolithic model of code development hence it is easy for developers to develop the code for application.
* Easy to Test – Testing the application becomes easy as all service components are integrated and working as a single piece of software.
* Easy Deployment – Since the entire application services are built and created as a single artifact like .war, it is easy to deploy a single .war file in the application server.
* Easy to scale resources – Since it is a single unit of deployment, we can easily create horizontal servers and deploy war on new servers and add to the Load balancer to balance & distribute the load.

**Disadvantages:**

* Maintenance - Monolithic code base is very large and makes it complicated to understand for any code changes. For every small change in code of a specific component needs to deploy the entire code.
* Performance - Application can slow down as initial load time is high to load large scale of code.
* Reliability — Small bugs in one component cause failure of the whole application.
* Software Upgrade Strategy - Monolithic applications have difficulty to upgrade to latest versions or adopt new technologies as that might conflict with other components in the application.

**Challenges with Monolithic Architecture**

* Scalability: It is easy to scale computing resources on Monolithic Architecture, but different components might require different capacity of resources hence challenging to provision resources satisfying all different components in the single large applications.
* Flexibility: Monolithic Architecture is not flexible for frequent deployments, frequent code changes or releases as every time the whole application needs to be reviewed as its single application code base for all service components.

**What is a Micro Service?**

Microservices Application is a development practice to build a large application as a suite of modular services (i.e. loosely coupled components). Each component supports a specific business function and uses a simple, well-defined interface to communicate with other sets of services.

**Micro Services Architecture**

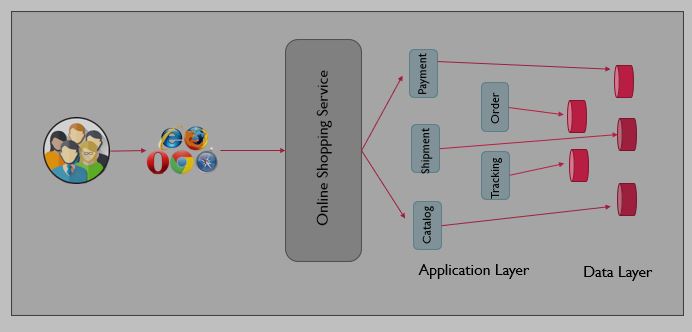
Consider an example of an online retail portal, where customers can browse the services through a browser and can add products to the cart, order a product, authorize payments, and track shipments. These are different components built based on the Micro services model. Each component has a separate loosely coupled service depending on the business requirement. Network established to have communication between all these service components but works as an independent code base.

Micro services Architecture deploy these components into separate units of application code in separate compute platforms.

Below listed each service component has its own code deployment:

1. Catalog – Listing Products inventory
2. Order – Takes customer order and process it
3. Payment – Manage Payments through payment gateway
4. Shipment – Manage Shipping the product
5. Tracking – Showing the Product Shipment status

Below Architecture describes the “Microservices Architecture” view of deploying all service components in separate compute platform with separate artifact(.war)

****

###### Micro Services Advantages Vs. Disadvantages

**Advantages:**

* Micro services are flexible to integrate with continuous integration and continuous deployment pipelines.
* It is easy to deploy code changes as each service component has its own artifact .war for deployment
* Development can be done by multiple contributors as each developer can take ownership of each service module and can develop independently and expose the functionality as API to integrate with other services.
* Maintenance is easy as each component code base is relatively small and easy to understand.
* Application load time is faster hence performance of the whole application is improved.
* If any specific bug in the code that impacts that individual service component and other services continue working without any issues.
* Easy to migrate software framework or upgrade to the latest technology versions that depend on that specific component.
* Easy to provision compute resources as per the requirement of each component in the same application.

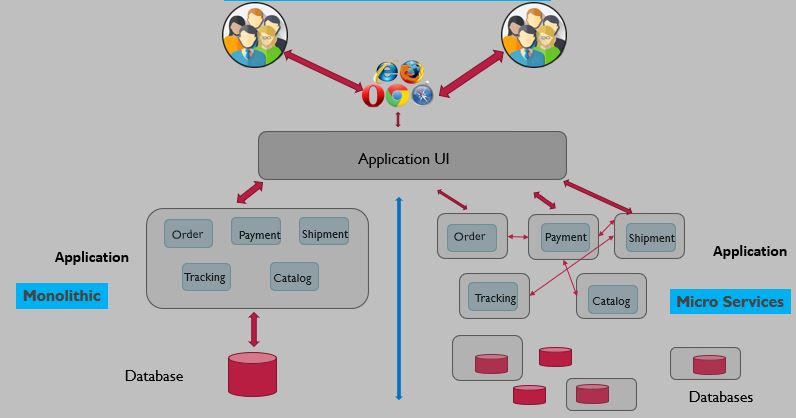
Disadvantages

* Micro services following distributed application architecture hence, developers require some efforts to understand the distributed environment for integrations.

###### Monolithic Vs. Micro Services Architecture

Monolithic Architecture is the traditional deployment practice and is more suitable for small scale applications which do not have much complexity in development and business functions.

Microservices Architecture is agile development practice and is more suitable for large scale applications which involve complexity in development and have many business services functions can run independently.

****

Based on the requirement we choose the best suited application architecture. When we deploy applications using Microservices Architecture the service component can run on separate Physical server, Virtual Server, or even in the container. We might be familiar with the Physical and Virtual compute environment; Let us talk about Container Compute Environment in the next Chapter.

**What is a Container?**

Container is the smallest unit of software that can run applications and its dependencies so that application runs quickly and reliably on that container.

It is even possible to create Image from the running container so that the Images can be built, Ship, and run anywhere in another container compute platform.

Container Engine which is installed and configured on Host OS is responsible for creating containers and maintaining isolation, compute resources provisioning to ensure containers are running healthy.

**What is Docker?**

Docker is the container engine that handles create, managing containers.

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker containers characteristics:

Standard: Docker created the industry standard for containers, so they could be portable anywhere

Lightweight: Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs

Secure: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

**Container orchestration**

Container orchestration is the process of automating, creating, managing containers that run in the pool of cluster nodes.

We have the below challenge to manage containers deployed with Microservices applications.

1. What if the container dies?
2. What if the container crashed?
3. What if the host node dies?
4. What if you would need to increase compute resources to meet the business demands?

Container Orchestration process automates the above said problems to create new containers automatically when container dies or crashes and even can scale the number of containers behind load balancer to avoid performance hiccups during high load.

### Advantages:

* Application Deployment is easy as the Orchestration process takes care of creating and maintaining the right number of containers.
* Easy to scale the number of containers
* Easy to maintain Availability and redundancy

**Container Orchestration Tools**

Container Orchestration Tools are taking the responsibility of creating containers and deploying the Microservices into containers. There are many orchestration tools available to handle the similar functionality with different capabilities.

### Kubernetes (K8s) Engine

* Google cloud platform offers Kubernetes Engine that provides container and cluster management services. Kubernetes Engine provides Kubernetes functions like deployment, scaling, and management of containerized applications.

### Docker Swarm

* Groups of Physical or virtual servers are grouped as clusters to run the containers on these clustered servers. Docker Swarm is the Container Orchestration Tool to manage container deployments in the cluster servers.

### Mesos

* Apache Mesos is the open-source project to manage a group of Physical or virtual servers to manage the container deployments. Mesos works with the same principles of Linux kernel.

**Kubernetes Managed Services**

Kubernetes services can be used as managed services from cloud providers. Service Providers set up and manage the Kubernetes Orchestration process in the backend and customers can just deploy the applications to get benefit of scalability, deployment, availability of the containers in the cluster.

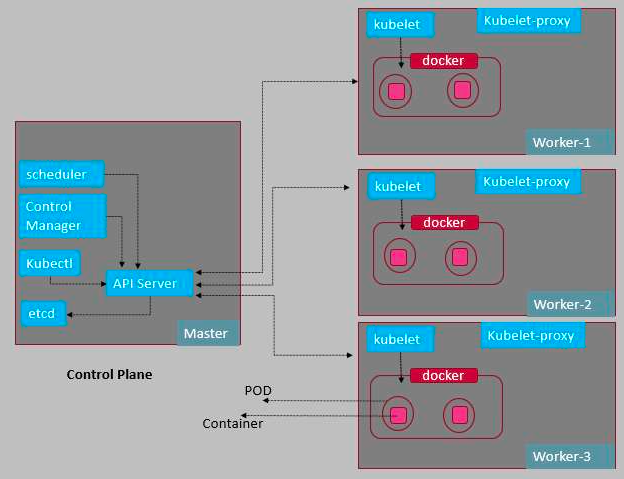
Below are the topmost Managed Kubernetes Services:

* [Amazon Elastic Kubernetes Service (EKS)](https://aws.amazon.com/eks/)
* [Azure Kubernetes Service (AKS)](https://azure.microsoft.com/en-us/services/kubernetes-service/)
* [IBM Cloud Kubernetes Service](https://www.ibm.com/cloud/container-service/)

**Kubernetes Architecture**

* Kubernetes follows client-server architecture. One server is configured as Kubernetes Master node and other servers are configured as Kubernetes worker nodes.
* Kubernetes Master node is installed and configured with Kubernetes Engine that controls the cluster operations and deployments.
* Kubernetes Worker node is where containers will be scheduled and run.

Below diagram shows Kubernetes Architecture and its components for cluster operations.



**Kubernetes Objects**

**Container Image**

* A container image is a ready-to-run software package, containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.

**Containers**

* Container is the smallest unit of software that can run applications and its dependencies so that application runs quickly and reliably on that container.

**POD**

* A Pod is the basic execution unit of a Microservice application. Kubernetes manages running the containers in the Pod abstract layer. Usually a single Pod contains only one Container, but Pod can have multiple Containers when those are having dependency on each other to run.

**POD Characteristics**

* Containers within a Pod share an IP address and port space and can find each other via localhost.

**Kubernetes Components**

A Kubernetes cluster consists of the components that represent the control plane and a set of machines called nodes.

**Control Plane Components**

**etcd:**

etcd is the consistent and highly available key value database used by Kubernetes as backing store for all cluster data.

**API Server:**

API Server exposes the Kubernetes API. API Server is the front-end component in the Kubernetes cluster. All cluster operational requests are receiving by API server to control the cluster operations.

**Kube-Controller Manager**

Manages controller processes in the cluster, below are the controller processes responsible for various operations.

* Node controller: Responsible for noticing and responding when nodes go down.
* Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
* Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
* Service Account & Token controllers: Create default accounts and API access tokens for new namespaces

**Kube-Scheduler**

Scheduler is responsible to monitor newly created pod and assigns the node where the pod can run. Kube-scheduler consider various factors to select the node where the pod can run like,

* Individual and collective resource requirements
* Hardware/software/policy constraints
* Affinity and anti-affinity specifications
* Data locality
* Inter-workload interference, and deadlines.

**Worker Node Components**

**Kubelet**

An agent service that runs on each worker node in the cluster. Kubelet ensures that the containers are running in the Pod.

Kubelet manages Pods that are created by Kubernetes. Kubelet takes the Pod Specification defined in the manifest and ensures that the containers are deployed accordingly and the healthy in the Pods.

**Kubelet-proxy**

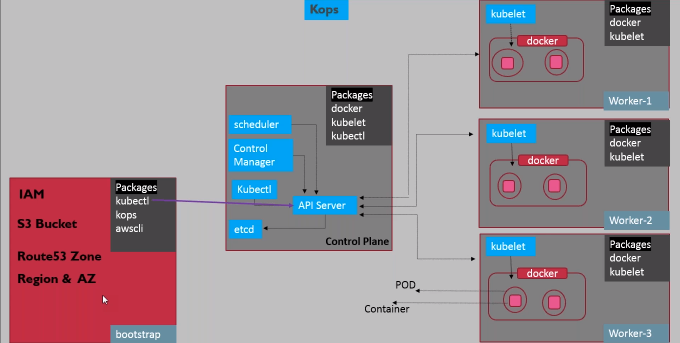
Kubelet-proxy is the network proxy that runs on each worker node to expose the services to the internet or public network.

**Container runtime**

Container runtime is the software that runs containers in the Pods. Kubernetes supports below container runtimes:

* Docker
* Containerd
* CRI-O

Among these, Docker is the most widely used container runtime to deploy containers managed within the Pods Kubernetes.



### kops

#### Create bootstrap server

Bootstrap server can be your local system or an EC2 instance created in AWS. Bootstrap server is only needed to setup the K8s cluster by running the kops commands to initialize the cluster.

1. Install AWS CLI
   * pip install awscli
2. Install Kubectl
   * curl -LO “https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl”
   * chmod +x kubectl
   * mv kubectl /usr/local/bin/kubectl
3. Install Kops
   * curl -Lo kops https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d ‘”‘ -f 4)/kops-linux-amd64
   * chmod +x kops
   * mv kops /usr/local/bin/kops

#### Create IAM Role

Create an IAM Role and attach the role to the bootstrap server. IAM Role must have below policies attached.

1. AmazonEC2FullAccess
2. AmazonRoute53FullAccess
3. AmazonS3FullAccess
4. IAMFullAccess
5. AmazonVPCFullAccess
6. AmazonEventBridgeFullAccess
7. AmazonSQSFullAccess

#### Configure Route53

Custom domain is required to set up the Kubernetes cluster using Kops, so that Kops can create required resource records in that domain in Route 53. example –  “dptcluster.devopsrealtime.com”

Gossip is an alternate option if you want to deploy a cluster without a custom domain in Route 53. To use gossip-based DNS, configure the cluster domain name to end with .k8s.local. example – “dptcluster.k8s.local”

#### Create Cluster State Storage (S3 Bucket)

To store the state of your cluster, and the representation of your cluster, we need to create a dedicated S3 bucket for kops to use. This bucket will become the source of truth for our cluster configuration.

* Login to AWS console and create S3 bucket. example bucket – “dpt-k8s-state-store”

#### Setup SSH Keys

Create SSH Keys by running “#ssh-keygen -t rsa”, which creates keys in default location $HOME/.ssh

#### **Create cluster configuration**

Congratulations!! You have set up all prerequisites on the bootstrap server.

We’re ready to start creating our k8s cluster! setup below environment variables to start creating cluster configuration.

**Custom Domain**

1. export NAME=dptcluster.devopsrealtime.com
2. export KOPS\_STATE\_STORE=s3://dpt-k8s-state-store

**Gossip Domain**

1. export NAME=dptcluster.k8s.local
2. export KOPS\_STATE\_STORE=s3://dpt-k8s-state-store

Run below command to create cluster configuration

1. kops create cluster --zones=us-east-1a ${NAME}

#### Modify cluster configuration (Optional)

Kops created default cluster configuration and now you can edit the configuration if needed to customize it by running below command

1. kops edit cluster ${NAME}

#### Build cluster

Run the below command to build the cluster with the configuration that was created in the previous step. Cluster build takes a while and hence waits for longer until the cluster is ready for you.

1. kops update cluster ${NAME} –yes –admin

Note that the configuration for your cluster was automatically generated and written to $HOME/.kube/config for you!  If config was not created then can export the config using below command

1. kops export kubecfg –admin

#### Validate cluster

Below commands help you to check your cluster status. If the cluster is not ready, wait for a few more minutes and check again!!

1. kops validate cluster
2. kubectl get nodes

#### Delete cluster

IMPORTANT! to note that cluster resources created by Kops are not in Free Tier, hence you may need to destroy resources once you are done with the setup.

You can preview all the AWS resources that will be destroyed when the cluster is deleted by issuing the following command.

1. kops delete cluster –name ${NAME}

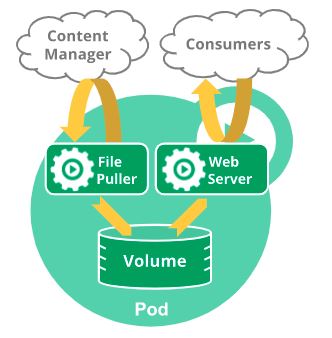
Below command to delete resources

1. kops delete cluster –name ${NAME} –yes

===============================================================

What is POD?

POD is the group of one or more containers with shared storage space and network. kubelet service is responsible for managing, deploying and running the containers and PODs within the kubernetes cluster.



This diagram depicts POD diagram having two containers which are dependent each other. Web Server is the container to server web traffic and File Puller is the another container which is helper container to the main container. Both containers are running in same POD sharing same volume space.

Source: [Kubernetes.io](https://kubernetes.io/)

Verify Cluster worker nodes

[root@ ~]$kubectl **get** nodes

NAME STATUS ROLES AGE VERSION

ip-172-31-66-17.ec2.**internal** Ready <**none**> 17h v1.18.5

ip-172-31-72-37.ec2.**internal** Ready <**none**> 17h v1.18.5

ip-172-31-76-168.ec2.**internal** Ready <**none**> 17h v1.18.5

ip-172-31-77-56.ec2.**internal** Ready master 18h v1.18.5

[iwayQ@ ~]$

Create POD Manifest file

Create "createpod.yaml" file with below manifest definitions

#Deploy nginx-pod with latest nginx version

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

env: sandbox

spec:

containers:

- name: ngginx-container

image: nginx

Deploy POD

[root@ ~]$kubectl apply -f createpod.yaml

pod/nginx-pod created

[iwayQ@ ~]$

Get PODs running on the k8s cluster

Below are the various commands to list running PODs with different properties

#Get Running PODs

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-pod 1/1 Running 0 40s

#Get Running PODs with wide info

[root@ ~]$kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

nginx-pod 1/1 Running 0 48s 172.31.140.65 ip-172-31-72-37.ec2.internal <none> <none>

#Get Running PODs

[r@oot ~]$kubectl get po

NAME READY STATUS RESTARTS AGE

nginx-pod 1/1 Running 0 55s

#Get Running PODs with POD label

[root@ ~]$kubectl get pods nginx-pod

NAME READY STATUS RESTARTS AGE

nginx-pod 1/1 Running 0 4m13s

[root@ ~]$

Describe POD manifest definition

[root@ ~]$kubectl **get** pods nginx-pod -o yaml

Describe POD description and events related to POD life cycle

[root@ ~]$kubectl describe pod nginx-pod

Name: nginx-pod

Namespace: default

Priority: 0

Node: ip-172-31-72-37.ec2.internal/172.31.72.37

Start Time: Sat, 04 Jul 2020 04:20:19 +0000

Labels: app=nginx

env=sandbox

Annotations: cni.projectcalico.org/podIP: 172.31.140.65/32

cni.projectcalico.org/podIPs: 172.31.140.65/32

Status: Running

IP: 172.31.140.65

IPs:

IP: 172.31.140.65

Containers:

ngginx-container:

Container ID: docker://9caa9edd297589f9a4a1966be5994885ebb216e3212c351a45516abdd1b9ca95

Image: nginx

Image ID: docker-pullable://nginx@sha256:21f32f6c08406306d822a0e6e8b7dc81f53f336570e852e25fbe1e3e3d0d0133

Port: <none>

Host Port: <none>

State: Running

Started: Sat, 04 Jul 2020 04:20:24 +0000

Ready: True

Restart Count: 0

Environment: <none>

Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-p8d6t (ro)

Conditions:

Type Status

Initialized True

Ready True

ContainersReady True

PodScheduled True

Volumes:

default-token-p8d6t:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-p8d6t

Optional: false

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal Scheduled 6m1s default-scheduler Successfully assigned default/nginx-pod to ip-172-31-72-37.ec2.internal

Normal Pulling 5m59s kubelet, ip-172-31-72-37.ec2.internal Pulling image "nginx"

Normal Pulled 5m56s kubelet, ip-172-31-72-37.ec2.internal Successfully pulled image "nginx"

Normal Created 5m56s kubelet, ip-172-31-72-37.ec2.internal Created container ngginx-container

Normal Started 5m56s kubelet, ip-172-31-72-37.ec2.internal Started container ngginx-container

[root@ ~]$

Run commands in POD

[root@ ~]$kubectl **exec** nginx-pod -- /bin/bash

[r@oot ~]$

Delete POD

[root@ ~]$kubectl delete pod nginx-pod

pod "nginx-pod" deleted

[root ~]$kubectl get pods

No resources found in default namespace.

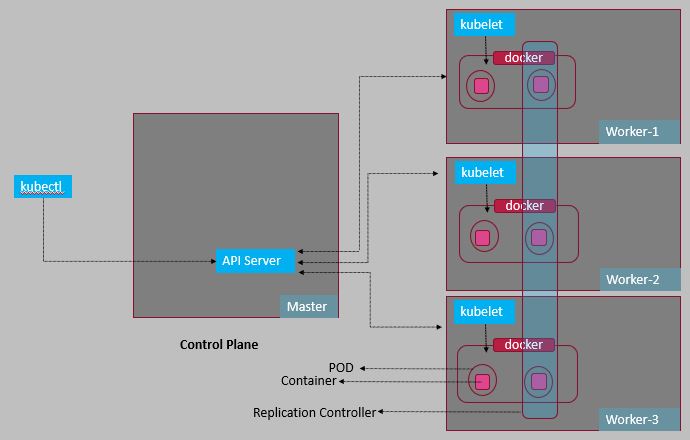
[root@ ~]$

=================================================================================

What is a Replication Controller?

The Replication Controller ensures that the specified number of pod replicas are running at any one time within the kubernetes cluster. The Replication Controller controls the creation of PODs automatically if the number of running POD replicas does not match the specified number.

When Deployed Replication Controller it takes care of creating specified numbers if POD replicas automatically as per the Manifest definitions.



Replication Controller

Replication Controller is also a more convenient way to scale out and scale in the number of POD replicas within the K8s cluster.

Create manifest file for Replication Controller

Create 'createrd.yaml' file with below manifest definitions

#Replication Controller

apiVersion: v1

kind: ReplicationController

metadata:

name: nginx-rc

spec:

replicas: 3

selector:

app: nginx-app

template:

metadata:

name: nginx-pod

labels:

app: nginx-app

env: sandbox

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

Deploy Replication Controller

[root@ ~]$kubectl apply -f createrc.yaml

replicationcontroller/nginx-rc created

[root@ ~]$

Get running PODs in the cluster

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rc-497w9 1/1 Running 0 26s

nginx-rc-6h7vp 1/1 Running 0 26s

nginx-rc-lbv5c 1/1 Running 0 26s

[root@ ~]$

Get running Replication Controllers in the cluster

root@ ~]$kubectl get rc

NAME DESIRED CURRENT READY AGE

nginx-rc 3 3 3 49s

root@ ~]$kubectl get rc nginx-rc

NAME DESIRED CURRENT READY AGE

nginx-rc 3 3 3 72s

[root@ ~]$

Describe Replication Controllers properties with events

[root@ ~]$kubectl describe rc nginx-rc

Name: nginx-rc

Namespace: default

Selector: app=nginx-app

Labels: app=nginx-app

env=sandbox

Annotations: Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

Pod Template:

Labels: app=nginx-app

env=sandbox

Containers:

nginx:

Image: nginx

Port: 80/TCP

Host Port: 0/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal SuccessfulCreate 2m5s replication-controller Created pod: nginx-rc-lbv5c

Normal SuccessfulCreate 2m5s replication-controller Created pod: nginx-rc-497w9

Normal SuccessfulCreate 2m5s replication-controller Created pod: nginx-rc-6h7vp

[root@ ~]$

Scale Out number of PODs managing by Replication Controller

[root@ ~]$kubectl scale rc nginx-rc --replicas=5

replicationcontroller/nginx-rc scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rc-497w9 1/1 Running 0 7m9s

nginx-rc-6fp9q 1/1 Running 0 14s

nginx-rc-6h7vp 1/1 Running 0 7m9s

nginx-rc-b4p6w 1/1 Running 0 14s

nginx-rc-lbv5c 1/1 Running 0 7m9s

[root@ ~]$

Scale In number of PODs managing by Replication Controller

[root@ ~]$kubectl scale rc nginx-rc --replicas=3

replicationcontroller/nginx-rc scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rc-497w9 1/1 Running 0 7m40s

nginx-rc-6h7vp 1/1 Running 0 7m40s

nginx-rc-b4p6w 0/1 Terminating 0 45s

nginx-rc-lbv5c 1/1 Running 0 7m40s

[root@ ~]$

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rc-497w9 1/1 Running 0 7m54s

nginx-rc-6h7vp 1/1 Running 0 7m54s

nginx-rc-lbv5c 1/1 Running 0 7m54s

[root@ ~]$

Delete Replication Controller

[root@ ~]$kubectl **delete** rc nginx-rc

replicationcontroller "nginx-rc" deleted

[root@ ~]$kubectl **get** rc

**No** resources **found** **in the default** namespace.

[root@ ~]$kubectl **get** pods

**No** resources **found** **in the default** namespace.

[root@ ~]$

**Replica Set Controller**

Create manifest file for Replica Set

Create "CreateReplicaSet.yaml" file with below manifest definitions

#Replica Set with Set-Based Selectors

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-rs

spec:

replicas: 5

selector:

matchLabels:

app: nginx-app

matchExpressions:

- {key: tier, operator: IN, values: [frontend]}

template:

name: nginx-pod

labels:

app: nginx-app

tier: frontend

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

Deploy Replica Set

[root@ ~]$kubectl apply -f CreateReplicaSet.yaml

replicaset.apps/nginx-rs created

[root@ ~]$

Verify running PODs in the cluster

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rs-6p274 1/1 Running 0 33s

nginx-rs-pfbfd 1/1 Running 0 33s

nginx-rs-w6xv6 1/1 Running 0 33s

[root@ ~]$kubectl get po

NAME READY STATUS RESTARTS AGE

nginx-rs-6p274 1/1 Running 0 39s

nginx-rs-pfbfd 1/1 Running 0 39s

nginx-rs-w6xv6 1/1 Running 0 39s

[iwayQ@ ~]$

Verify running Replica Set in the cluster

[root@ ~]$kubectl get rs

NAME DESIRED CURRENT READY AGE

nginx-rs 3 3 3 60s

[root@ ~]$kubectl get rs -o wide

NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR

nginx-rs 3 3 3 64s nginx-container nginx app=nginx-app,tier in (frontend)

[root@ ~]$kubectl get rs nginx-rs

NAME DESIRED CURRENT READY AGE

nginx-rs 3 3 3 73s

[root@ ~]$

Describe running Replica Set properties and events

[root@ ~]$kubectl describe rs nginx-rs

Name: nginx-rs

Namespace: default

Selector: app=nginx-app,tier in (frontend)

Labels: <none>

Annotations: Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

Pod Template:

Labels: app=nginx-app

tier=frontend

Containers:

nginx-container:

Image: nginx

Port: 80/TCP

Host Port: 0/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal SuccessfulCreate 104s replicaset-controller Created pod: nginx-rs-w6xv6

Normal SuccessfulCreate 104s replicaset-controller Created pod: nginx-rs-pfbfd

Normal SuccessfulCreate 104s replicaset-controller Created pod: nginx-rs-6p274

[root@ ~]$

Scale Out POD replicas manged by Replica Set

[root@ ~]$kubectl scale rs nginx-rs --replicas=5

replicaset.apps/nginx-rs scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rs-6p274 1/1 Running 0 2m38s

nginx-rs-lqdcg 1/1 Running 0 6s

nginx-rs-pfbfd 1/1 Running 0 2m38s

nginx-rs-vbmqg 1/1 Running 0 6s

nginx-rs-w6xv6 1/1 Running 0 2m38s

[root@ ~]$

Scale In POD replicas managed by Replica Set

[root@ ~]$kubectl scale rs nginx-rs --replicas=3

replicaset.apps/nginx-rs scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rs-6p274 1/1 Running 0 2m59s

nginx-rs-lqdcg 0/1 Terminating 0 27s

nginx-rs-pfbfd 1/1 Running 0 2m59s

nginx-rs-vbmqg 0/1 Terminating 0 27s

nginx-rs-w6xv6 1/1 Running 0 2m59s

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-rs-6p274 1/1 Running 0 3m9s

nginx-rs-pfbfd 1/1 Running 0 3m9s

nginx-rs-w6xv6 1/1 Running 0 3m9s

[iwayQ@ ~]$

Delete Replica Set

[root@ ~]$kubectl **delete** rs nginx-rs

replicaset.apps "nginx-rs" deleted

[root@ ~]$kubectl **get** rs

**No** resources **found** **in** **default** namespace.

[root@ ~]$kubectl **get** pods

**No** resources **found** **in** **default** namespace.

[root~]$

===============================================================================

**Deployment Controller**

**What is Deployment?**

Deployment is the process of running the application work load on the POD.

Create manifest file for Deployment in Kubernetes cluster

Create "createDeployment.yaml" file with the below manifest file definitions

#Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deploy

labels:

app: nginx-app

spec:

replicas: 3

selector:

matchLabels:

app: nginx-app

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

Create Deployment in K8s cluster

[root@ ~]$kubectl apply -f createDeployment.yaml

deployment.apps/nginx-deploy created

[root@ ~]$

Verify Deployments

[root@ ~]$kubectl get deploy

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deploy 3/3 3 3 36s

[root@ ~]$kubectl get deploy -o wide

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

nginx-deploy 3/3 3 3 39s nginx-container nginx app=nginx-app

[root@ ~]$

Verify Replica Sets

[root@ ~]$kubectl get rs

NAME DESIRED CURRENT READY AGE

nginx-deploy-7dcc9fb9c4 3 3 3 2m19s

[root@ ~]$kubectl get rs -o wide

NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR

nginx-deploy-7dcc9fb9c4 3 3 3 2m21s nginx-container nginx app=nginx-app,pod-template-hash=7dcc9fb9c4

[root@ ~]$

Verify running PODs

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-2zkwg 1/1 Running 0 2m54s

nginx-deploy-7dcc9fb9c4-5pl8l 1/1 Running 0 2m54s

nginx-deploy-7dcc9fb9c4-pvflw 1/1 Running 0 2m54s

[root@ ~]$kubectl get po

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-2zkwg 1/1 Running 0 2m57s

nginx-deploy-7dcc9fb9c4-5pl8l 1/1 Running 0 2m57s

nginx-deploy-7dcc9fb9c4-pvflw 1/1 Running 0 2m57s

[root@ ~]$

Describe Replica Set configuration

[root@ ~]$kubectl describe rs

Name: nginx-deploy-7dcc9fb9c4

Namespace: default

Selector: app=nginx-app,pod-template-hash=7dcc9fb9c4

Labels: app=nginx-app

pod-template-hash=7dcc9fb9c4

Annotations: deployment.kubernetes.io/desired-replicas: 3

deployment.kubernetes.io/max-replicas: 4

deployment.kubernetes.io/revision: 1

Controlled By: Deployment/nginx-deploy

Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

Pod Template:

Labels: app=nginx-app

pod-template-hash=7dcc9fb9c4

Containers:

nginx-container:

Image: nginx

Port: 80/TCP

Host Port: 0/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal SuccessfulCreate 3m24s replicaset-controller Created pod: nginx-deploy-7dcc9fb9c4-5pl8l

Normal SuccessfulCreate 3m24s replicaset-controller Created pod: nginx-deploy-7dcc9fb9c4-2zkwg

Normal SuccessfulCreate 3m24s replicaset-controller Created pod: nginx-deploy-7dcc9fb9c4-pvflw

[root@ ~]$

Describe Deployment configuration

[root@ ~]$kubectl describe deploy

Name: nginx-deploy

Namespace: default

CreationTimestamp: Sat, 04 Jul 2020 06:32:36 +0000

Labels: app=nginx-app

Annotations: deployment.kubernetes.io/revision: 1

Selector: app=nginx-app

Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable

StrategyType: RollingUpdate

MinReadySeconds: 0

RollingUpdateStrategy: 25% max unavailable, 25% max surge

Pod Template:

Labels: app=nginx-app

Containers:

nginx-container:

Image: nginx

Port: 80/TCP

Host Port: 0/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>

Conditions:

Type Status Reason

---- ------ ------

Available True MinimumReplicasAvailable

Progressing True NewReplicaSetAvailable

OldReplicaSets: <none>

NewReplicaSet: nginx-deploy-7dcc9fb9c4 (3/3 replicas created)

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal ScalingReplicaSet 3m53s deployment-controller Scaled up replica set nginx-deploy-7dcc9fb9c4 to 3

[root@ ~]$

Scale Out POD Replicas in the Replica Set managed by Deployment

[root@ ~]$kubectl scale deployment nginx-deploy --replicas=5

deployment.apps/nginx-deploy scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-8dtc2 1/1 Running 0 43s

nginx-deploy-7dcc9fb9c4-c5tg2 1/1 Running 0 43s

nginx-deploy-7dcc9fb9c4-tt2fc 1/1 Running 0 43s

nginx-deploy-7dcc9fb9c4-z29qq 1/1 Running 0 4s

nginx-deploy-7dcc9fb9c4-zjf78 1/1 Running 0 4s

[root@ ~]$kubectl get rs

NAME DESIRED CURRENT READY AGE

nginx-deploy-7dcc9fb9c4 5 5 5 50s

[root@ ~]

POD Replicas in the Replica Set managed by Deployment

[root@ ~]$kubectl scale deployment nginx-deploy --replicas=3

deployment.apps/nginx-deploy scaled

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-8dtc2 1/1 Running 0 2m3s

nginx-deploy-7dcc9fb9c4-c5tg2 1/1 Running 0 2m3s

nginx-deploy-7dcc9fb9c4-tt2fc 1/1 Running 0 2m3s

nginx-deploy-7dcc9fb9c4-z29qq 0/1 Terminating 0 84s

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-8dtc2 1/1 Running 0 2m6s

nginx-deploy-7dcc9fb9c4-c5tg2 1/1 Running 0 2m6s

nginx-deploy-7dcc9fb9c4-tt2fc 1/1 Running 0 2m6s

nginx-deploy-7dcc9fb9c4-z29qq 0/1 Terminating 0 87s

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-8dtc2 1/1 Running 0 2m9s

nginx-deploy-7dcc9fb9c4-c5tg2 1/1 Running 0 2m9s

nginx-deploy-7dcc9fb9c4-tt2fc 1/1 Running 0 2m9s

[root@ ~]$kubectl get rs

NAME DESIRED CURRENT READY AGE

nginx-deploy-7dcc9fb9c4 3 3 3 2m14s

[root@ ~]$

Delete Deployment

[root@ ~]$kubectl delete deploy nginx-deploy

deployment.apps "nginx-deploy" deleted

[root@ ~]$kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-deploy-7dcc9fb9c4-pvflw 0/1 Terminating 0 6m30s

[root@ ~]$kubectl get pods

No resources found in default namespace.

[root@ ~]$kubectl get rs

No resources found in default namespace.

[root@ ~]$kubectl get deploy

No resources found in default namespace.

[root@ ~]$

**Certification Tip!**

Here's a tip!

As you might have seen already, it is a bit difficult to create and edit YAML files. Especially in the CLI. During the exam, you might find it difficult to copy and paste YAML files from browser to terminal. Using the kubectl run command can help in generating a YAML template. And sometimes, you can even get away with just the kubectl run command without having to create a YAML file at all. For example, if you were asked to create a pod or deployment with specific name and image you can simply run the kubectl run command.

Use the below set of commands and try the previous practice tests again, but this time try to use the below commands instead of YAML files. Try to use these as much as you can going forward in all exercises

Before we begin, familiarize with the two options that can come in handy while working with the below commands:

--dry-run: By default as soon as the command is run, the resource will be created. If you simply want to test your command , use the --dry-run=client option. This will not create the resource, instead, tell you whether the resource can be created and if your command is right.

-o yaml: This will output the resource definition in YAML format on screen.

Use the above two in combination to generate a resource definition file quickly, that you can then modify and create resources as required, instead of creating the files from scratch.

**Create an NGINX Pod**

kubectl run nginx --image=nginx

**Generate POD Manifest YAML file (-o yaml). Don't create it(--dry-run)**

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

**Create a deployment**

kubectl create deployment --image=nginx nginx

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run)**

kubectl create deployment --image=nginx nginx --dry-run=client -o yaml

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run) with 4 Replicas (--replicas=4)**

kubectl create deployment --image=nginx nginx --dry-run=client -o yaml > nginx-deployment.yaml

**You can also scale a deployment using the kubectl scale command.**

**kubectl scale deployment nginx --replicas=4**

**Save it to a file, make necessary changes to the file (for example, adding more replicas) and then create the deployment.**

kubectl create -f nginx-deployment.yaml

**OR**

**In k8s version 1.19+, we can specify the --replicas option to create a deployment with 4 replicas.**

kubectl create deployment --image=nginx nginx --replicas=4 --dry-run=client -o yaml > nginx-deployment.yaml

#### Service

**Create a Service named redis-service of type ClusterIP to expose pod redis on port 6379**

**kubectl expose pod redis --port=6379 --name redis-service --dry-run=client -o yaml**

**(This will automatically use the pod's labels as selectors)**

**Or**

**kubectl create service clusterip redis --tcp=6379:6379 --dry-run=client -o yaml (This will not use the pods labels as selectors, instead it will assume selectors as app=redis.** [**You cannot pass in selectors as an option.**](https://github.com/kubernetes/kubernetes/issues/46191) **So it does not work very well if your pod has a different label set. So generate the file and modify the selectors before creating the service)**

**Create a Service named nginx of type NodePort to expose pod nginx's port 80 on port 30080 on the nodes:**

**kubectl expose pod nginx --type=NodePort --port=80 --name=nginx-service --dry-run=client -o yaml**

**(This will automatically use the pod's labels as selectors,** [**but you cannot specify the node port**](https://github.com/kubernetes/kubernetes/issues/25478)**. You have to generate a definition file and then add the node port manually before creating the service with the pod.)**

**Or**

**kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml**

**(This will not use the pods labels as selectors)**

**Both the above commands have their own challenges. While one of them cannot accept a selector the other cannot accept a node port. I would recommend going with the kubectl expose command. If you need to specify a node port, generate a definition file using the same command and manually input the nodeport before creating the service.**

#### Reference:

[**https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands**](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands)

[**https://kubernetes.io/docs/reference/kubectl/conventions/**](https://kubernetes.io/docs/reference/kubectl/conventions/)