Services

Service in Kubernetes is used to defines a logical set of Pods and a policy by which to access them. Services are used for communications among different pods. Service is defined using YAML or JSON file. The set of Pods targeted by a Service is usually determined by a LabelSelector. We will learn more about Labels and Selectors later.

Services are used to expose the pods outside the cluster. Services are exposed in different ways using different ServiceSpecs such as

ClusterIP

NodePort

Load Balancer

ExternalName

Kube-proxy

Every Worker node runs a daemon process called kube-proxy

The kube - proxy keeps track of API server of the master node for addition and deletion of Service endpoints.

For every Service created in a worker node, the kube-proxy configures the IP tables rules to handle the traffic for its ClusterIP and forwards it to service endpoints

When the service is removed, kube-proxy removes the iptables rules on all worker nodes as well.

Service Discovery

Service discovery at runtime is an important concept because all communications from external world to Kubernetes occurs through Services.

Environment Variables: When the Pod starts on a worker node, the kubelet adds a set of variables called Environment Variables in the Pod for all active Services. For example, if we have an active Service called redis-master, having Cluster-IP 172.10.0.12 and exposing Port -7307, the environment variables created will be as shown

REDIS\_MASTER\_SERVICE\_HOST=172.10.0.12

REDIS\_MASTER\_SERVICE\_PORT=7307

REDIS\_MASTER\_PORT=tcp://172.10.0.12:7307

REDIS\_MASTER\_PORT\_7307\_TCP=tcp://172.10.0.12:7307

REDIS\_MASTER\_PORT\_7307\_TCP\_PROTO=tcp

REDIS\_MASTER\_PORT\_7307\_TCP\_PORT=7307

REDIS\_MASTER\_PORT\_7307\_TCP\_ADDR=172.10.0.12

DNS : There are addon components in Kubernetes called DNS which creates and manages the DNS names for all the Services created. The format looks like my-svc.my-namespace.svc.cluster.local.

Types of Services

In the previous card the different type of specs for Service creation has been discussed. In this card the types of Services are defined.

Cluster IP: Cluster IP is the default Service type, in this type of Service there is virtual IP that can be used for communication only within the cluster.

NodePort: In NodePort Service there is a virtual IP address along with a port number in the range of 30000-32767, that maps to the respective Service from all the worker nodes.

A default port number is selected by the Kubernetes while creating this type of Service. NodePort type Service is used when we want to communicate with the external world.

Types of Services...

Load Balancer: This type of Service uses external load balancer, the Cluster IP and NodePort for this type of Service is automatically created and the load balancing is done by external load balancer.

External IP: This type of Services are mapped to External IP address if it can route worker nodes. The External IPs are not managed by Kubernetes.

Labels

**Labels** are key-value pair, that are used to organise and select attributes of an object. Ex

"metadata":{

"labels":{

"key1":"value1"

"key2":"value2"

}}

Labels can be used to map organisation structure onto the Kubernetes Object. Ex:

"environment":"dev","environment":"qa"

##### Selectors

**Selectors** are used to identify a sub-set of Objects. The kubernetes API supports two types of selectors.

* equality based
* set based

Empty label selectors select all the objects in the collection. A null label selector selects no objects

**Equality-based requirements** are used for identifying sub-sets of objects based on keys and values The =,==,!= are used for equality based selectors.

Ex: environment = production

**Set based Requirments** are used for filtering objects based on a set of values. There are three kinds of operators used in, not in and exists

Ex: environment in (production, dev)

9 of 15

Replication Controller

* Replication controller is a part of the controller- manager and it is used to produce replicas of the pods because pods are transient in nature and cannot replicate them selves.
* If the number of Replicas are less they generate more replicas of the pods to maintain the desired state.
* If the number of replicas are more they are destroyed

##### Deployment Objects and Deployment controller

#### Deployment Object

* A deployment object is used to define the desired state of an application.
* Deployments can create new replica sets, delete other deployments and update deployments.

#### Deployment Controller

* Deployment controller is a a part controller-manager of the Master node.
* It is used to create and update pods and replica sets.
* Deployment Controller converts the actual state to desired state

##### Names and Namespaces

**Names** are the client provided name given to a kubernetes Object in resource URL.

Ex; /api/v1/pods/example-pods/

**UID**: A system-generated string uniquely identifies a kubernetes Object.

**Namespaces**: Namespaces are virtual clusters within physical clusters. They are used when users are spread across multiple teams or projects. The names inside a namespace must be unique but can be similar across namespaces.

##### Creating and Deleting Namespaces

Namepaces are created using YAML files. Ex:

apiVersion: v1

kind: Namespace

metadata:

name: <insert-namespace-name-here>

the $kubectl create -f <namespace -name-here> command is run in CLI to create namespace. \*

##### What are Volumes?

Volumes are the persistent state data that can be stored on-disk or on other containers.

#### Why use Volumes in Kubernetes?

* You know that Pods are transient in nature and when the containers in a pod crash all the data inside themare deleted and new containers are created. The Data inside them is lost, to overcome this problem Volumes are used.
* Volumes can store data on a storage medium or in othe containers.
* Volumes are placed inside the Pods and are connected to the containers.

Persistent Volumes

* A PersistentVolume (PV) is storage space created in the cluster by the administrator.
* It is similar to other cluster resources like the nodes. PVs are volume plugins like Volumes, but unlike volumes inside a container, they have a lifecycle independent that of the lifecycle of the pod.
* The API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

**PersistentVolumeClaim (PVC)** are storage requests created by the user. Users request for Persistent Volumes based on their requirments. Once a suitable PersistentVolume is found, it is bound to a PersistentVolumeClaim.

Like Pods consume node resources, the PVCs consume Persistent Volume resources. Pods can request specific levels of resources (CPU and Memory).

Pod Lifecycle

Pods are transient, pods are created, assigned a unique ID (UID), and scheduled to nodes where they remain until termination or deletion. If a node dies, the pods scheduled to that node are scheduled for deletion, after a timeout period. A pod is never rescheduled to a node instead, a new pod with a new UID is assigned to the node. the following are the terms used to define a Pod Lifecycle.

* Pod Phase
* Pod Condition
* Container Probes
* Pod Lifetime

Pod Lifecycle Terms

**Pod Phase**: The Pod phase is a simple yet high-level summary of where the pod is in its lifecycle. The different values of Pod phase are Running, Succeeded, failed and Unknown.

**Pod Condition**: is an array through which the Pod has to pass. Each element in **Pod Condition** has six possible fields.

1. lastProbeTime:
2. lastTransitionTime:
3. message:
4. reason:
5. status:
6. type

* PodScheduled
* Ready
* Initialized
* Unscheduled
* ContainersReady

Pod Life Cycle Terms contd...

**Container Probe** is a health check performed by the kubelet on a container. Handler is used to perform the check. There are three types of handlers

1. ExecAction:
2. TCPSocketAction
3. HTTPGetAction

Visit [this link](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes) for more information.

**Pod Lifetime**: Pods are not deleted and destroyed automatically unless they are destroyed manually or by a controller but there is an exception to this when the pod with a phase succeeded or failed for more than a specific duration it will expire and automatically destroyed.

Deployment and Deployment Controller

A Deployment controller is used to create and update Pods and ReplicaSets.

Deployment controller changes the actual state to the desired state as described in the deployment object. It can create new RepilcaSets, remove existing deployments and create new deployments.

**Deployment Controller uses**:

* Creating deployments for new Replicasets
* Declaring desired states of the pods
* Rolling back deployment to an earlier version
* Scaling up the deployment
* To pause or stop deployments
* To check the status of deployment
* Clearing old ReplicaSets

**Creating Deployments**: A deployment is created using a YAML file an then running the following command on Terminal.

$kubectl create -f <file\_name>.yaml

To see the created deployments

$kubectl get deployments

5 of 10

ConfigMaps

* ConfigMaps are used to define the configuration of a container image.
* They are used to separate the configuration details from the container image, so that you do not have to create multiple images for similar applications, instead you can create multiple ConfigMaps.
* ConfigMaps can be used by Pods and replication controllers to create multiple containers.
* There are two ways in which configMaps can be created
  + from literals
  + from files

##### Types of ConfigMaps

#### Creating configMaps files using literals

To create configMaps using literals we use the kubectl CLI. Execute the following command to create a configMap.

$ kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2

configmap "my-config" created

You can get the configuration information by executing the following command

$ kubectl get configmaps my-config -o yaml

#### Creating the configMap using YAML file

This method of creating a configMap is more preferred because it gives you full control to change the different components of the configMap. The YAML file structure looks like this

apiVersion: v1

kind: ConfigMap

metadata:

name: imageName

data:

image data

To create the configMap file create a yaml file and run the kubectl create command Ex: $ kubectl create -f imageName-configmap.yaml

Secrets

* Secrets are volume spaces that are used to store sensitive information of the application.
* They are created using the kubectl create command

$ kubectl create secret generic userpassword --from-literal=password=loginpassword

This creates a secret called userpassword that stores the login password.

* Secrets can be created manually by using YAMl files and encoding the information. -Ex:

$ echo loginpassword | base64

bG9naW5wYXNzd29yZAo=

The YAML file

apiVersion: v1

kind: Secret

metadata:

name: userpassword

type: Opaque

data:

password: bXlzcWxwYXNzd29yZAo=