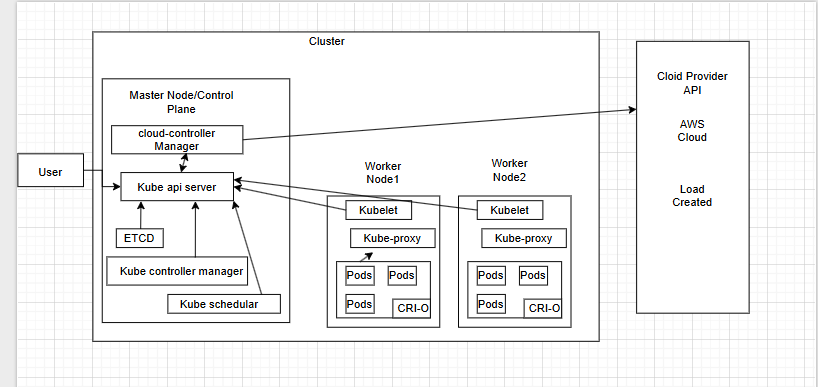
Kubernetes Cluster Architecture



Pod.yml

1.apiVersion: V1

Specify the api version for Kubernetes resource file.

2.kind

Specify the type of resource file you want to write.

3. metadata

Contain descriptive data for the pod file.

3.1 name-> Define unique name for the pod.

3.2 namespace-> Namespace where the pod is created.

3.3 labels

app-> Key value pair for grouping and selecting the pod.

4.spec -> Define the desired state and content for the pod.

4.1 containers -> List of one or more containers to run inside the pod.

4.2 name-> Name of container that run inside the pod

4.3 image-> Docker image to use for the container.

5. ports -> List of port that container will listen on.

5.1 conatinerPort ->expose port 80 which is default for nginx webserver.

Commands

kubectl get pods

kubectl get pods -o wide

kubectl get pods -n <namespace-name>

kubectl get pods --all-namespaces

kubectl get pods -l app=my-app

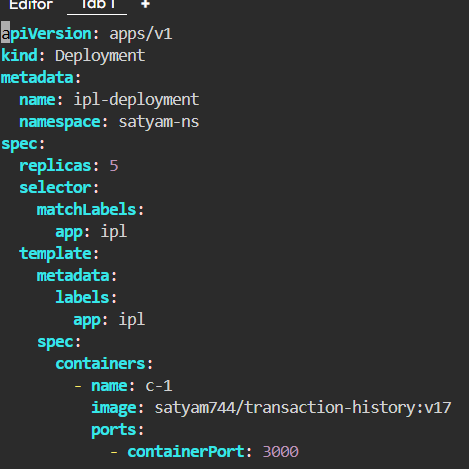
kubectl describe pod <pod-name>

kubectl delete pod <pod-name>

kubectl logs <pod-name>

kubectl exec -it <pod-name> -- /bin/bash

kubectl apply -f <file-name.yaml>



Deployment.yml

1.apiVersion: apps/v1

Specify the api version for Kubernetes deployment file.

2.kind

Specify the type of resource file you want to write.

3. metadata

Contain descriptive data for the deployment file.

3.1 name-> Define unique name for the deployment file.

3.2 namespace-> Namespace where the deployment is created.

4. spec-> Define the desired state and content for the deployment.

4.1 replicas -> It will tell the deployment controller that you have to maintain 5 replicas so that if any one pod fails, it will takeover that.

4.2 selector ->This is how the deployment finds and manages the pods it controls.

4.2.1 matchLabels -> This is a label selector that must match the labels in the pod template.

4.2.1.1 app:ipl-> Deployment will manage any pod that has the label app with the value ipl.

4.3 template -> Blueprint for the pods that the deployment will create.

4.3.1 metadata-> Label selector that must match the labels in the pod template.

4.3.2 labels->Blueprint for the pods that the deployment will create.

4.3.3 app ->Label applied to each pod created by this deployment.

5. spec ->Defines the containers to run inside the pods.

5.1 container-> List of containers within the pod.

5.2 name:c-1 -> Name of container.

5.3 image ->Docker image to use for the container

5.4 ports -> List of ports to expose from the container.

5.5containerPort-> The cluster can access this container on this port.

Commands

kubectl apply -f <deployment-file.yaml>

kubectl get deployments

kubectl get deployments -o wide

kubectl describe deployment <deployment-name>

kubectl scale deployment <deployment-name> --replicas=<number>

kubectl set image deployment/<deployment-name> <container-name>=<new-image>

kubectl rollout status deployment/<deployment-name>

kubectl rollout history deployment/<deployment-name>

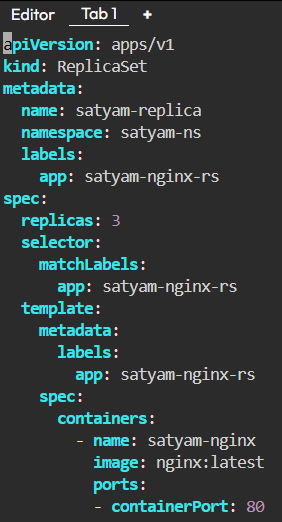
kubectl rollout undo deployment/<deployment-name>

kubectl rollout undo deployment/<deployment-name> --to-revision=<revision-number>

kubectl delete deployment <deployment-name>

kubectl edit deployment <deployment-name>

kubectl create deployment <deployment-name> --image=<image-name>



Replica.yml

1.apiVersion: apps/v1 -> Specify the api version for Kubernetes replica file.

2.kind -> Specify the type of resource file you want to write.

3. metadata -> Contain descriptive data for the replicaset file.

3.1 name-> Define unique name for the replicasetfile file.

3.2 namespace-> Namespace where the deployment is created.

4. spec-> Define the desired state and content for the deployment.

4.1 replicas -> If a pod is deleted, the ReplicaSet will immediately create a new one to replace it.

4.2 selector ->The mechanism by which the replicaset identifies which pods it should manage..

4.2.1 matchLabels -> This is a label selector that must match the labels in the pod template.

4.2.1.1 app:ipl-> Deployment will manage any pod that has the label app with the value ipl.

4.3 template -> Blueprint for the pods that the deployment will create.

4.3.1 metadata-> Label selector that must match the labels in the pod template.

4.3.2 labels->Blueprint for the pods that the deployment will create.

4.3.3 app ->Label applied to each pod created by this deployment.

5. spec ->Defines the containers to run inside the pods.

5.1 container-> List of containers within the pod.

5.2 name:c-1 -> Name of container.

5.3 image ->Docker image to use for the container

5.4 ports -> List of ports to expose from the container.

5.5containerPort-> The cluster can access this container on this port.

Commands

kubectl apply -f <replicaset-file.yaml>

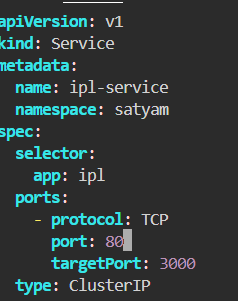
kubectl get rs

kubectl get rs -o wide

kubectl describe rs <replicaset-name>

kubectl scale rs <replicaset-name> --replicas=<number>

kubectl delete rs <replicaset-name>



Services

**1.ClusterIP**

apiVersion -> Specify the api version for Kubernetes api resource file.

kind-> Define the type of resource file you want to write.

metadata ->contains the metadata for service file.

name -> Unique name for the Service within its namespace. Allowing other applications inside the cluster to find and communicate with this Service using its name.

namespace -> specifies the namespace where the Service will be created.

Spec -> Specifying the service file desired state and behavior.

Selector -> Service uses this label selector to find and group all the pods that it should send traffic to. Any pod that has the label app:ipl will be automatically included as an endpoint for this Service.

ports -> Defines the ports that the Service will expose.

Protocol -> Specifies the network protocol.

Port -> port number that other applications inside the Kubernetes cluster will use to access this Service.

targetport -> Specifies the port number on the container inside the pod that the Service will forward traffic to.

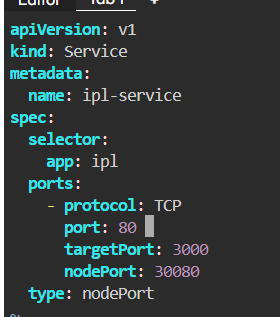
type:ClusterIP -> Defines the type of Service. ClusterIP is the default Service type and creates a stable, internal-only IP address.

**Commands**

kubectl apply -f service.yml -n satyam

kubectl get svc -n satyam

kubectl describe svc ipl-service -n satyam



2. NodePort

apiVersion -> Specify the api version for kubernetes api resource file.

kind-> Define the type of resource file you want to write.

metadata ->contains the metadata for service file.

name -> Unique name for the Service within its namespace. Allowing other applications inside the cluster to find and communicate with this Service using its name.

namespace -> specifies the namespace where the Service will be created.

Spec -> Specifying the service file desired state and behavior.

Selector -> Service uses this label selector to find and group all the pods that it should send traffic to. Any pod that has the label api-ipl will be automatically included as an endpoint for this Service.

ports -> Defines the ports that the Service will expose.

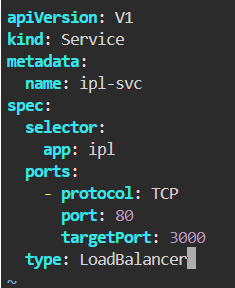
Protocol -> Specifies the network protocol.

Port -> port number that other applications inside the Kubernetes cluster will use to access this Service.

targetport -> Specifies the port number on the container inside the pod that the Service will forward traffic to.

type:nodePort -> Defines the type of service that allow expose of your application to external traffic.

**commands**

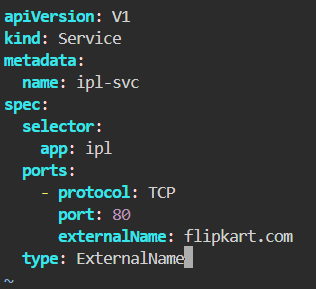


3.LoadBalancer

It will expose the application outside cluster.

It will give one externe lip address for accessing your app outside the cluster.

It is when you want to use your application without hostname for temporary purpose.



Externel Name

It maps the service to an external hostame.

When to use -> when you want to access application via dns or host name.

Commands

kubectl get services

kubectl get svc -o wide

kubectl get svc -n <namespace-name>

kubectl describe service <service-name>

kubectl apply -f <service-file.yaml>

kubectl apply -f <service-file.yaml>

kubectl expose deployment <deployment-name> --name=<service-name> --port=<service-port> --target-port=<container-port> --type=<service-type>

kubectl delete service <service-name>

kubectl delete -f <service-file.yaml>

kubectl edit service <service-name>

kubectl get endpoints <service-name>