Kubernetes Deep Dive — Practices and Further Notes

Last Edited	@June 28, 2023 10:24 PM
Tags	
study	

Start K8S

- https://labs.play-with-k8s.com/

  login, create a session, and use remote server for 4 hours

Create a cluster

```
# master node (need a new instance)
kubeadm init --apiserver-advertise-address ...
# use printout second command
kubectl apply -f https://raw.githubusercon ...
# verify master node ready
kubectl get nodes

# worker node (need a new instance)
# copy and paste the output from the last kubeadm init command
kubeadm join ...
```

```
[node1 ~]$ kubectl get nodes
NAME    STATUS   ROLES           AGE     VERSION
node1   Ready    control-plane   3m12s   v1.27.2
node2   Ready    <none>          28s     v1.27.2
node3   Ready    <none>          8s      v1.27.2
```

- Google Kubernetes Engine (GKE)

  https://cloud.google.com/kubernetes-engine

Pod, Deployment, Service

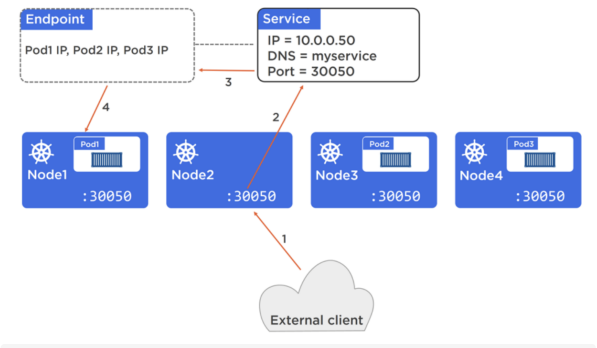
Deployment:

- features: scaling, self-healing, rolling update
- rolling update: provides 0 downtimes, iteratively updates pods

Service:

- typically in svc.yml, where states how the application is exposed to outside
- always has reliable DNS, IP, PORT
- maintain a dynamic list of healthy pods (called endpoints), which the network will be directed to
- `ServiceTypes` : `ClusterIP` vs `NodePortService`: Internal communication vs external communication

Figure: NodePortService Redirection



NodePort Service with 3 Pods exposed externally on port 30050 on every node in the cluster

Practices:

```
# ============ useful commands ===========
# get pods on one node
kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=<node>
```

```
# see pods status: spec is the desired status, status is observed status
kubectl get pods <pod-name> -o yaml
# get rolling update status
kubectl rollout status deployment <deployment-name>
# get list of endpoint object
kubectl get ep hello-svc
```

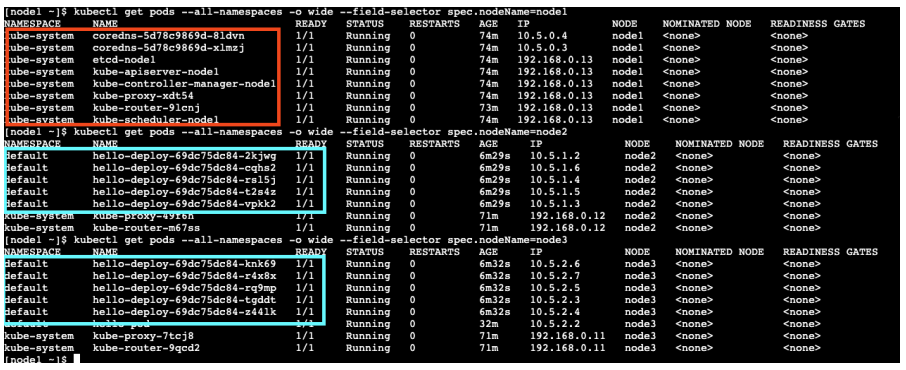


Figure: *I set 10 deploy object, which loads balance to two worker nodes, while the master node has most of the Kubernetes system running: etcd, API server, etc.*

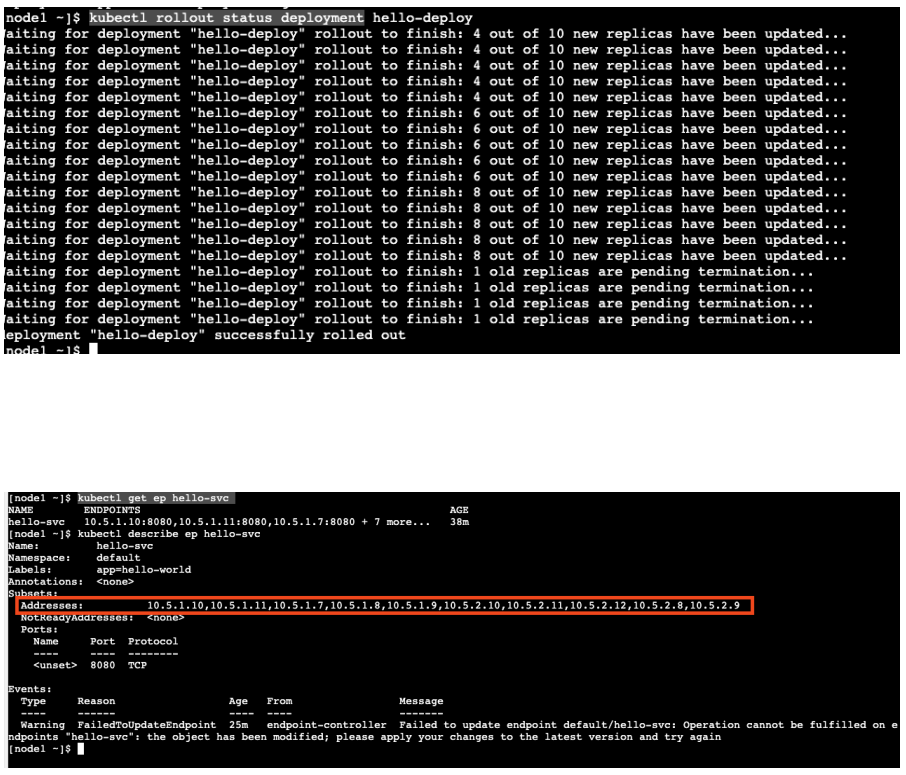
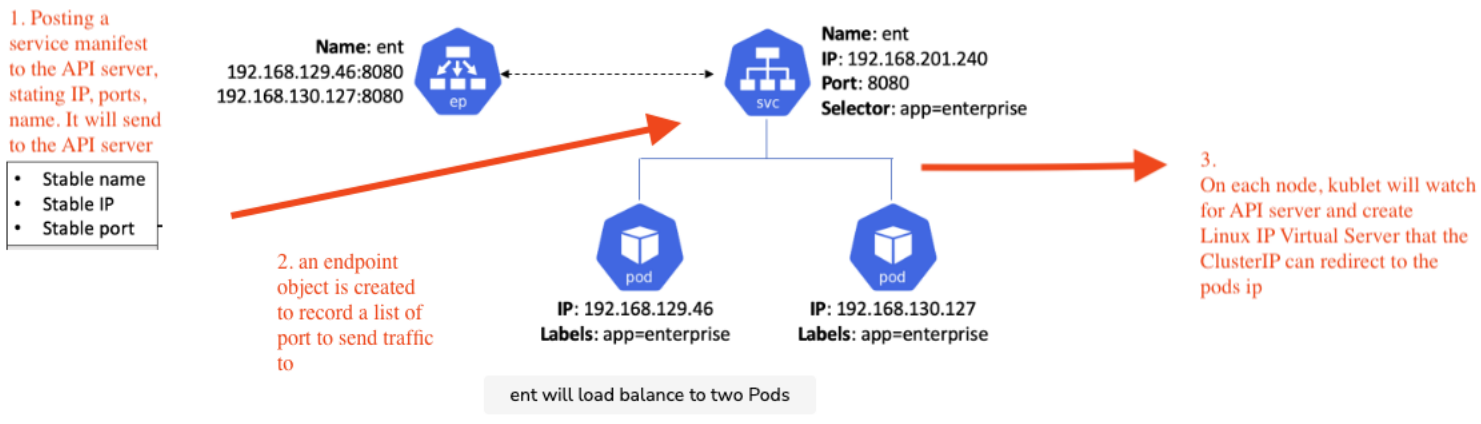


Figure: rolling update status

Figure: a list of health pods. this list will be updated as pods are down or up

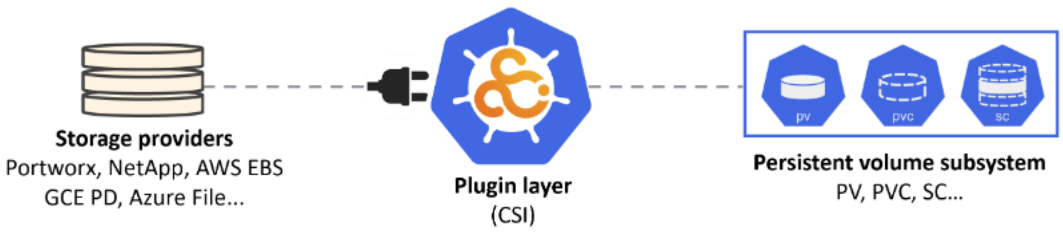
Service Discovery

- “Service registration is the process of a microservice registering its connection details in a *service registry* so that other microservices can discover it and connect to it.”
- Service discovery is a process one service communicates with the other
- Cluster DNS operates at an address known to every Pod and container in the cluster
  - it is a Kubernetes-native application
  - implements a controller that constantly watches the API server for a new service (a new app) object to be registered
- Service Registry Behind Scene (Brief, more complicated)
  - Cluster DNS helps the first step



Kubernetes Storage

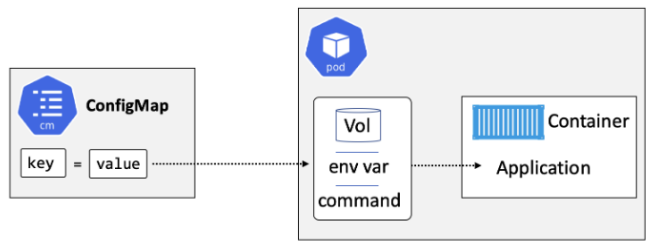
- PersistentVolumes (PV) are how you map external storage onto the cluster
- PersistentVolumeClaims (PVC) are like tickets that authorize applications (Pods) to use a PV.
- Storage class: external storage provider, such as GCP, AWS, Azure, etc
  - “Storage Classes take things to the next level by allowing applications to dynamically request storage. You create a Storage Class object that references a class, or tier, of stora



ConfigMap, StatefulSet, DaemonSet

ConfigMap

- ConfigMap: separate configuration for different environments and application images, so that the application can be deployed in different environments (e.g. dev, test, production)
- ConfigMaps are typically used to store non-sensitive configuration data such as:
  - Environment variable values
  - Entire configuration files (things like web server configs (like **nginx**) and database configs)
  - Hostnames
  - Service ports
  - Account names
- Internally, it's a key-value pair



Working of ConfigMaps

- Kubernetes-native applications can access ConfigMap data directly via the API without needing things like environment variables and volumes

StatefulSet

- used for stateful application: e.g. DB, storing users' authentication token
- some features:
  - Persistent Pod Names, DNS hostnames, volume bindings
    - Pod are named as `<StatefulSetName>-<Integer>`
    - volume binding: if a pod is gone, a new pod will be launched and attached to the same volume
  - "StatefulSets create one Pod at a time and always wait for previous Pods to be *running and ready* before creating the next."
    - same for scaling, one pod will wait for the other; not same for deleting pod

Ordered starting of Pods

- Working Example (using Google **GKE**)
  - https://github.com/nigelpoulton/TheK8sBook/tree/main/statefulsets
  - app.yml: have headless services, statefulset definition with volume (PVC)
  - gcp-sc.yml: use `pd.csi.storage.gke.io` as cloud provisioner

  each Statefulset will be bounded with one volume

```
harrisonliding@cloudshell:~ (kubernetes-391000)$ kubectl get pvc
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-tkb-sts-0   Bound    pvc-dbbe962b-7771-4cf9-b3bd-54a36bccdfcf   1Gi        RWO            flash          2d23h
www-tkb-sts-1   Bound    pvc-52341ff5-5e93-433e-ac1b-a648437a20e4   1Gi        RWO            flash          2d22h
www-tkb-sts-2   Bound    pvc-d2d79b11-badf-4165-a29c-b57f203d2036   1Gi        RWO            flash          2d22h
harrisonliding@cloudshell:~ (kubernetes-391000)$
```

DaemonSet

- guarantees to have a pod replica on every single node of a cluster

Reference

Notes mostly are summarized from or refer to *Learn Kubernetes: A Deep Dive* https://www.educative.io/courses/the-kubernetes-course