

# Kubernetes Deep Dive — Practices and Further Notes

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

Start K8S

[Pod](#), [Deployment](#), [Service](#)

[Deployment](#):

[Service](#):

[Practices](#):

[Service Discovery](#)

[Kubernetes Storage](#)

[ConfigMap](#), [StatefulSet](#), [DaemonSet](#)

[ConfigMap](#)

[StatefulSet](#)

[DaemonSet](#)

[Reference](#)

## 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.githubusercontent.com ...
# 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

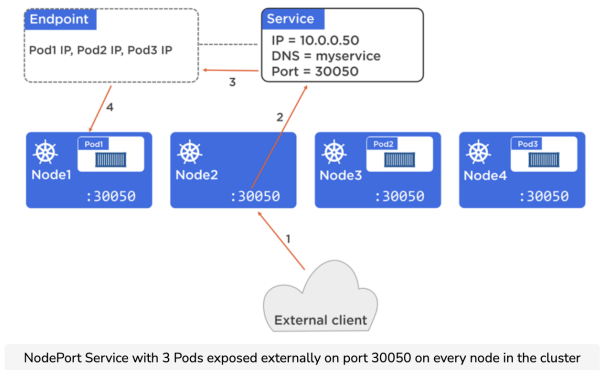
### Deployment:

- 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



### 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
```

```
[node1 ~]$ kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=node1
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
kube-system	coredns-5d78c9869d-81dvn	1/1	Running	0	74m	10.5.0.4	node1	<none>	<none>
kube-system	coredns-5d78c9869d-ximsj	1/1	Running	0	74m	10.5.0.3	node1	<none>	<none>
kube-system	etcd-node1	1/1	Running	0	74m	192.168.0.13	node1	<none>	<none>
kube-system	kube-apiserver-node1	1/1	Running	0	74m	192.168.0.13	node1	<none>	<none>
kube-system	kube-controller-manager-node1	1/1	Running	0	74m	192.168.0.13	node1	<none>	<none>
kube-system	kube-proxy-xdt54	1/1	Running	0	74m	192.168.0.13	node1	<none>	<none>
kube-system	kube-router-91cnj	1/1	Running	0	73m	192.168.0.13	node1	<none>	<none>
kube-system	kube-scheduler-node1	1/1	Running	0	74m	192.168.0.13	node1	<none>	<none>

```
[node1 ~]$ kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=node2
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
default	hello-deploy-69dc75dc84-zk3wg	1/1	Running	0	6m29s	10.5.1.2	node2	<none>	<none>
default	hello-deploy-69dc75dc84-qghu2	1/1	Running	0	6m29s	10.5.1.6	node2	<none>	<none>
default	hello-deploy-69dc75dc84-rs15j	1/1	Running	0	6m29s	10.5.1.4	node2	<none>	<none>
default	hello-deploy-69dc75dc84-t2s4z	1/1	Running	0	6m29s	10.5.1.5	node2	<none>	<none>
default	hello-deploy-69dc75dc84-vpkk2	1/1	Running	0	6m29s	10.5.1.3	node2	<none>	<none>
kube-system	kube-proxy-xvzss	1/1	Running	0	71m	192.168.0.12	node2	<none>	<none>
kube-system	kube-router-m57ss	1/1	Running	0	71m	192.168.0.12	node2	<none>	<none>

```
[node1 ~]$ kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=node3
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
default	hello-deploy-69dc75dc84-knk69	1/1	Running	0	6m32s	10.5.2.6	node3	<none>	<none>
default	hello-deploy-69dc75dc84-r4x8x	1/1	Running	0	6m32s	10.5.2.7	node3	<none>	<none>
default	hello-deploy-69dc75dc84-rq9mp	1/1	Running	0	6m32s	10.5.2.5	node3	<none>	<none>
default	hello-deploy-69dc75dc84-tqddt	1/1	Running	0	6m32s	10.5.2.3	node3	<none>	<none>
default	hello-deploy-69dc75dc84-z44lk	1/1	Running	0	6m32s	10.5.2.4	node3	<none>	<none>
kube-system	kube-proxy-7tcj8	1/1	Running	0	32m	10.5.2.2	node3	<none>	<none>
kube-system	kube-router-9qcd2	1/1	Running	0	71m	192.168.0.11	node3	<none>	<none>

```
[node1 ~]$
```

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.

```
node1 ~$ kubectl rollout status deployment hello-deploy
Waiting for deployment "hello-deploy" rollout to finish: 4 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 4 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 4 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 4 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "hello-deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "hello-deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "hello-deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "hello-deploy" rollout to finish: 1 old replicas are pending termination...
deployment "hello-deploy" successfully rolled out
node1 ~$
```

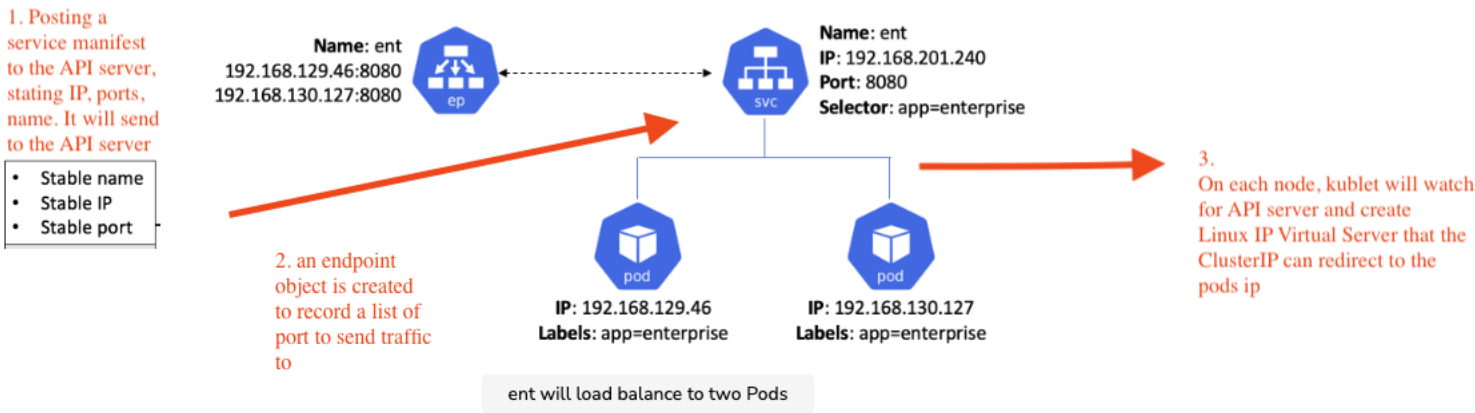
Figure: rolling update status

```
[node1 ~]$ kubectl get ep hello-svc
NAME                                ENDPOINTS                                AGE
hello-svc                          10.5.1.10:8080,10.5.1.11:8080,10.5.1.7:8080 + 7 more... 38m
(node1 ~)$ kubectl describe ep hello-svc
Name:                               hello-svc
Namespace:                          default
Labels:                              app=hello-world
Annotations:                         <none>
Subsets:
Addresses:                          10.5.1.10,10.5.1.11,10.5.1.7,10.5.1.8,10.5.1.9,10.5.2.10,10.5.2.11,10.5.2.12,10.5.2.8,10.5.2.9
NotReadyAddresses:                  <none>
Ports:
  Name    Port    Protocol
  ----    -
  <unset> 8080    TCP
Events:
Type     Reason              Age   From      Message
----     -
Warning  FailedToUpdateEndpoint 25m   endpoint-controller  Failed to update endpoint default/hello-svc: Operation cannot be fulfilled on endpoints "hello-svc": the object has been modified; please apply your changes to the latest version and try again
(node1 ~)$
```

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

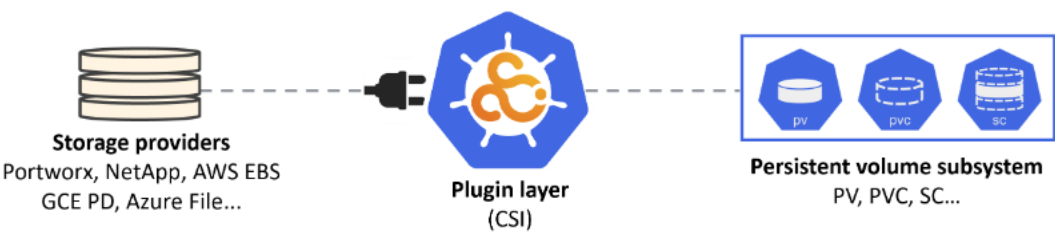
## Service Discovery

- “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

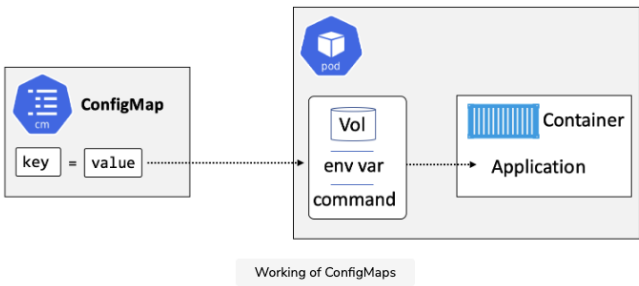
- PersistentVolumes (PV) are how you map external storage onto the cluster
- PersistentVolumeClaims (PVC) are like tickets that authorize applications (Pods) to use a PV.
- Storage class: external storage provider, such as GCP, AWS, Azure, etc
  - “Storage Classes take things to the next level by allowing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



## ConfigMap, StatefulSet, DaemonSet

### ConfigMap

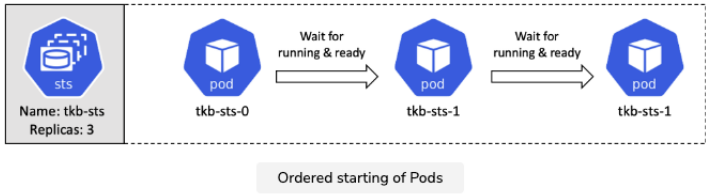
- ConfigMap: separate configuration for different environments and application images, so that the application can be deployed in different environments (e.g. dev, test, production)
- ConfigMaps are typically used to store non-sensitive configuration data such as:
  - Environment variable values
  - 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



- 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



- 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](https://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) <https://www.educative.io/courses/the-kubernetes-course>