K8S (3) K8S Object

The basic model of K8S objects type, i.e. Nodes, Namespaces, Pods, ReplicaSets, Deployments, DaemonSets, etc.
- apiVersion: api version
- kind: obj type
- metadata: accounting purpose
- spec: Desired state of this object

Node:

Hosting a container runtime, managed by kubelet and kube-proxy
(1) Control plane node (2) Worker node

Namespace:

Virtual partition in cluster

```
$ kubectl get namespaces // $kubectl get ns
NAME STATUS AGE
default Active 136m
kube-node-lease Active 136m
kube-public Active 136m
kube-system Active 136m
kubernetes-dashboard Active 135m

$ kubectl create namespace [new-namespace-name]
```

Pod:

Represents a single instance of the application. One pod can represent collection of containers.
Pod is ephemeral = doesn't last a lone time
Following yaml file create a pod. With name=nginx-pod, one single container inside the pos is running a nginx:1.22.1 image. The image is pulled from your container image registry. In our case: Docker Hub.

```
# nginx.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    run: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.22.1
    ports:
    - containerPort: 80
```

```
$ kubectl apply -f nginx.yaml
pod/nginx-pod created

$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
nginx-pod 1/1 Running 0 15s 10.244.120.73 minikube <none> <none>
```

Using apply yaml file is known as declarative method.
Another approach to create a pod is imperative method.

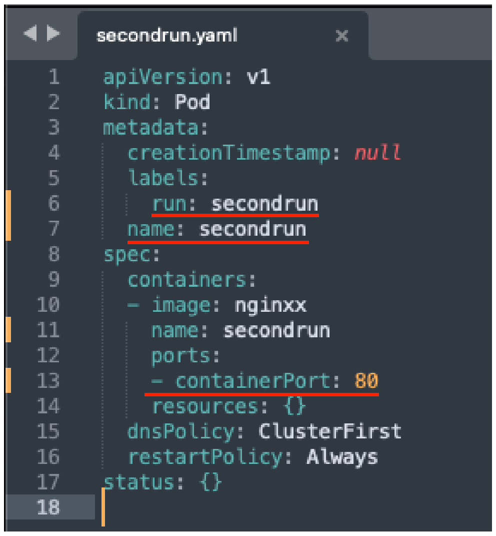
```
$ kubectl run firstrun --image=nginx
pod/firstrun created

$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
firstrun 1/1 Running 0 18s 10.244.120.75 minikube <none> <none>
nginx-pod 1/1 Running 0 123m 10.244.120.73 minikube <none> <none>
```

The third approach to create pod is mix method.

```
$ kubectl run firstrun --image=nginxx --port=88 --dry-run=client -o yaml >
secondrun.yaml
// We use dry-run flag so nothing will actually be created, dump the definition to
secondrun.yaml
```

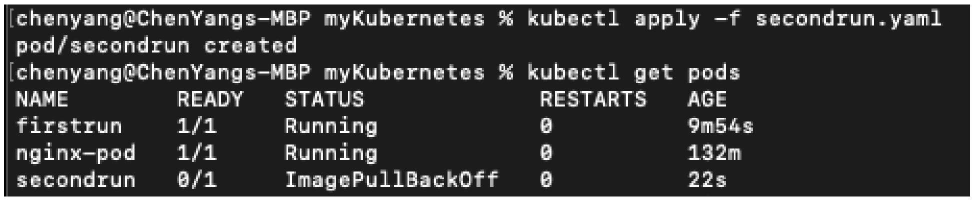
```
$ cat secondrun.yaml // notice the resource name is still firstrun, the resource
name needs to be unique in a namespace
apiVersion: v1
kind: Pod
metadata:
creationTimestamp: null
labels:
run: firstrun
name: firstrun
spec:
containers:
- image: nginxx
name: firstrun
ports:
- containerPort: 88
resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}
```

Update the name and port and $ kubectl apply -f second.yaml



```
secondrun.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: secondrun
  name: secondrun
spec:
  containers:
  - image: nginxx
    name: secondrun
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

If this is the only change you made, you should see there is an ImagePullBackOff when you check the secondrun pod we just created.



```
[chenyang@ChenYangs-MBP myKubernetes % kubectl apply -f secondrun.yaml
pod/secondrun created
[chenyang@ChenYangs-MBP myKubernetes % kubectl get pods
NAME        READY   STATUS             RESTARTS   AGE
firstrun    1/1     Running            0          9m54s
nginx-pod   1/1     Running            0          132m
secondrun   0/1     ImagePullBackOff   0          22s
```

Let's check the detail of the pod. You can see the Event section record what happened to our secondrun pod. The container image registry failed to pull the nginxx image. This is because we have a typo. There is no image called nginxx.

```
$ kubectl describe pods secondrun
Name: secondrun
Namespace: default
...
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Normal Scheduled 44s default-scheduler Successfully assigned default/secondrun to
minikube
Normal Pulling 28s (x2 over 44s) kubelet Pulling image "nginxx"
Warning Failed 24s (x2 over 42s) kubelet Failed to pull image "nginxx": Error
response from daemon: pull access denied for nginxx, repository does not exist or
may require 'docker login': denied: requested access to the resource is denied
Warning Failed 24s (x2 over 42s) kubelet Error: ErrImagePull
Normal BackOff 11s (x2 over 41s) kubelet Back-off pulling image "nginxx"
Warning Failed 11s (x2 over 41s) kubelet Error: ImagePullBackOff
```

After you update the image name to nginx, you can delete the pod and re-apply. Or force replace the pod

```
$ kubectl replace --force -f secondrun.yaml
pod "secondrun" deleted
pod/secondrun replaced

$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
firstrun 1/1 Running 0 22m 10.244.120.75 minikube <none> <none>
nginx-pod 1/1 Running 0 145m 10.244.120.73 minikube <none> <none>
secondrun 1/1 Running 0 27s 10.244.120.77 minikube <none> <none>
```

Use delete to clean up the pod

```
$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
firstrun 1/1 Running 0 22m 10.244.120.75 minikube <none> <none>
nginx-pod 1/1 Running 0 145m 10.244.120.73 minikube <none> <none>
secondrun 1/1 Running 0 27s 10.244.120.77 minikube <none> <none>

$ kubectl delete -f nginx.yaml
pod "nginx-pod" deleted

$ kubectl delete pods firstrun secondrun
pod "firstrun" deleted
pod "secondrun" deleted

$ kubectl get pods -o wide
```

```
No resources found in default namespace.
```

Label and Selector:

You can add label to K8S objects and controller can select specific group. Two selectors are supported in K8S:
(1) Equality-Based Selectors (**run==fristrun** or **run!=fristrun**)
(2) Set-Based Selectors (**run in (fristrun, secondrun) or !run)**

ReplicationControllers:

Setup the pod replicas. Create new pod if actual count < desired count. Delete pod if actual count > desired count.
This is for old K8S, the new version recommend to use Deployment+ReplicaSet Controller. The biggest difference between RC and RS is RS support Set-Based Selectors.

ReplicaSet:

Watch the lifecycle of application. Scale if needed. tool: autoscaler

```
# replicaSet.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: guestbook
  template:
    metadata:
      labels:
        app: guestbook
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

You can use RS to manage pod. However, RS is subset of Deployment and also creating Deployment will automatically create RS. Just use Deployment.

Deployment:

DeploymentController is in Control Manager in Control Plane Node. (Check the high level architecture and review the Control Manager)
As a controller, Deployment also ensures that the current state (Pods & ReplicaSet) always matches the desired state of our running containerized application.
Using RollingUpdate strategy (rollouts and rollbacks) to update/rollback applications.

```
# deployment.yaml
apiVersion: apps/v1
# kind shows the object type, could be Pod, ReplicaSet, Namespace, Service, etc.
kind: Deployment
# metadata holds the basic info of this object
metadata:
  name: nginx-deployment
  labels:
    app: nginx-deployment
# spec defines the desired state of this object
# We request 3 replicas = always have 3 pods running. The pods are following the
template we define here
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-deployment
  template:
    metadata:
      labels:
        app: nginx-deployment
# spec defines the desired state of this object(template)
# We request single container running nginx:1.20.2 image
    spec:
      containers:
      - name: nginx
        image: nginx:1.20.2
        ports:
        - containerPort: 80
```

This is an interesting sample. Two spec (spec and spec.template.spec) serves different purposes. First spec defines the desired state of Deployment, Second spec defines the desired state of Pod. This is nested object (Pod is included in Deployment). The inner object can define its metadata and spec, the apiVersion and kind are inherited from outer object.

Deployment + ReplicaSet:

Deployment will create ReplicaSet, ReplicaSet will create pods and maintain the replica count. This particular state will be record as Revision 1. When the deployment is update, a new ReplicaSet will be created (also new pods are created) and rollout Revision.
Notice that the revision update only happens when deployment itself is update, i.e. Changing container image/port/volume/mount. Changing label or scaling will not create new revision. This is called Rolling Update.
The reason K8S use revision to manage state is, the previous state can be store through revision. Allowing us to Rollback to previous revision.

DaemonSets:

DS is like Deployment+ReplicaSet, managing Pod replica and application update. But DS will enforce Pod replica in every node. (Deployment+ReplicaSet might put multiple Pods on the same node). DS is essential when you need to monitor every node. Like the kube-proxy agent runs as a pod in every node, Calico networking node agent manages the Pod networking. Both are managed by DS.

Service:

In K8S, container application cannot connect to another container application. Because container doesn't expose itself's port.
We reply on Service to accomplish. A Service will access kube-proxy+IP table+Routing rule+cluster DNS, add micro Load Balancer on top and expose the container port.

Demo the Rolling Update and Rollback:

```
$ kubectl create deployment mynginx --image=nginx:1.15-alpine // Create a new
deployment
deployment.apps/mynginx created

$ kubectl get deploy,rs,po -l app=mynginx // List all deployment+replicaSet+pod with
specific label
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/mynginx 1/1 1 1 44s
```

```
NAME DESIRED CURRENT READY AGE
replicaset.apps/mynginx-7fbcf7bbfd 1 1 1 44s

NAME READY STATUS RESTARTS AGE
pod/mynginx-7fbcf7bbfd-ddp5v 1/1 Running 0 44s

$ kubectl scale deploy mynginx --replicas=3 // Scale up
deployment.apps/mynginx scaled

$ kubectl get deploy,rs,po -l app=mynginx
// List again to check the replica, notice the replicaSet is still the same. Record
this name (mynginx-7fbcf7bbfd) and observe the change in the following steps.
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/mynginx 3/3 3 3 102s

NAME DESIRED CURRENT READY AGE
replicaset.apps/mynginx-7fbcf7bbfd 3 3 3 102s

NAME READY STATUS RESTARTS AGE
pod/mynginx-7fbcf7bbfd-9kp28 1/1 Running 0 2s
pod/mynginx-7fbcf7bbfd-dgvbk 1/1 Running 0 59s
pod/mynginx-7fbcf7bbfd-h85hk 1/1 Running 0 2s
```

Check the detail of the mynginx deployment, especially the current image version (nginx:1.15-alpine)

```
$ kubectl describe deployment mynginx
Name: mynginx
Namespace: default
CreationTimestamp: Fri, 22 Dec 2023 21:42:00 -0800
Labels: app=mynginx
Annotations: deployment.kubernetes.io/revision: 1
Selector: app=mynginx
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=mynginx
Containers:
nginx:
Image: nginx:1.15-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>
```

```
Conditions:
Type Status Reason
---- ------ ------
Progressing True NewReplicaSetAvailable
Available True MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet: mynginx-7fbcf7bbfd (3/3 replicas created)
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Normal ScalingReplicaSet 80s deployment-controller Scaled up replica set mynginx-
7fbcf7bbfd to 1
Normal ScalingReplicaSet 23s deployment-controller Scaled up replica set mynginx-
7fbcf7bbfd to 3 from 1
```

Check the revision history

```
$ kubectl rollout history deployment mynginx // Check the revision history. The
current revision 1 is associated with the replicaSet mynginx-7fbcf7bbfd
deployment.apps/mynginx
REVISION CHANGE-CAUSE
1 <none>

$ kubectl rollout history deployment mynginx --revision=1 // Check detail of the
current revision, showing the image version
deployment.apps/mynginx with revision #1
Pod Template:
Labels: app=mynginx
pod-template-hash=7fbcf7bbfd
Containers:
nginx:
Image: nginx:1.15-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>
```

Update the deployment by rolling update. Don't get confused by the rolling update. It means changing the resource state, possibly a downgrade.

```
$ kubectl set image deployment mynginx nginx=nginx:1.16-alpine // Upgrade image
version to nginx:1.16-alpine
deployment.apps/mynginx image updated

$ kubectl rollout history deployment mynginx // Check the revision history, you can
see an additional revision is here
deployment.apps/mynginx
```

```
REVISION CHANGE-CAUSE
1 <none>
2 <none>

$ kubectl rollout history deployment mynginx --revision=1 // Revision 1: The same
deployment.apps/mynginx with revision #1
Pod Template:
Labels: app=mynginx
pod-template-hash=7fbcf7bbfd
Containers:
nginx:
Image: nginx:1.15-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>

$ kubectl rollout history deployment mynginx --revision=2 // Revision 2: The current
state with desired image version
deployment.apps/mynginx with revision #2
Pod Template:
Labels: app=mynginx
pod-template-hash=6fdbc5d54c
Containers:
nginx:
Image: nginx:1.16-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>

$ kubectl get deploy,rs,po -l app=mynginx
// List again. The previous replicaSet (mynginx-7fbcf7bbfd) is still here, but being
scaled down to zero-pod. There is a new replicaSet associated with new pods.
// The previous replicaSet is stil here so we can rollback any time.
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/mynginx 3/3 3 3 42m

NAME DESIRED CURRENT READY AGE
replicaset.apps/mynginx-6fdbc5d54c 3 3 3 79s
replicaset.apps/mynginx-7fbcf7bbfd 0 0 0 42m

NAME READY STATUS RESTARTS AGE
pod/mynginx-6fdbc5d54c-ctf76 1/1 Running 0 20s
pod/mynginx-6fdbc5d54c-d2nmb 1/1 Running 0 19s
```

```
pod/mynginx-6fdbc5d54c-p62zb 1/1 Running 0 21s
```

Try to rollback

```
$ kubectl rollout undo deployment mynginx --to-revision=1 // Rollback to revision 1
deployment.apps/mynginx rolled back

$ kubectl rollout history deployment mynginx // Check the revision history, there is
additional revision 3
deployment.apps/mynginx
REVISION CHANGE-CAUSE
2 <none>
3 <none>

$ kubectl rollout history deployment mynginx --revision=2 // The same
deployment.apps/mynginx with revision #2
Pod Template:
Labels: app=mynginx
pod-template-hash=6fdbc5d54c
Containers:
nginx:
Image: nginx:1.16-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>

$ kubectl rollout history deployment mynginx --revision=3 // This is the state after
we rollback, the same with revision 1
deployment.apps/mynginx with revision #3
Pod Template:
Labels: app=mynginx
pod-template-hash=7fbcf7bbfd
Containers:
nginx:
Image: nginx:1.15-alpine
Port: <none>
Host Port: <none>
Environment: <none>
Mounts: <none>
Volumes: <none>

$ kubectl get deploy,rs,po -l app=mynginx
// The revision 2 replicaSet (mynginx-6fdbc5d54c) being scaled down to zero.
// The replicaSet for revision 3=1 being scaled up to three pods.
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/mynginx 3/3 3 3 4m9s
```

```
NAME DESIRED CURRENT READY AGE
replicaset.apps/mynginx-6fdbc5d54c 0 0 0 2m20s
replicaset.apps/mynginx-7fbcf7bbfd 3 3 3 4m9s

NAME READY STATUS RESTARTS AGE
pod/mynginx-7fbcf7bbfd-htqd8 1/1 Running 0 76s
pod/mynginx-7fbcf7bbfd-jh2ll 1/1 Running 0 76s
pod/mynginx-7fbcf7bbfd-x72d6 1/1 Running 0 75s
```

One thing notice that K8S allow 10 consecutive rolling updates.

Demo the DS:

```
# fluentd-agent.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-agent
  labels:
    k8s-app: fluentd-agent
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-agent
  template:
    metadata:
      labels:
        k8s-app: fluentd-agent
    spec:
      containers:
      - name: fluentd-agent
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
      terminationGracePeriodSeconds: 30
```

Above is a sample of DaemonSet, which will ensure every node in the cluster will have a pod replica running. The pod defined by the template.

Notice we don't need to define the number of replica in DS, it will match the number of node in the cluster.
In our case, we have 3 nodes.

```
$ kubectl apply -f fluentd-agent.yaml
daemonset.apps/fluentd-agent created

$ kubectl get daemonSet.apps // Validate there are 3 replicas
NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
fluentd-agent 3 3 3 3 3 <none> 24s

$ kubectl get pods -o wide // Check the pod list, we can see every pod is in
different node
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
fluentd-agent-92jpd 1/1 Running 0 41s 10.244.120.113 minikube <none> <none>
fluentd-agent-skljs 1/1 Running 0 41s 10.244.205.195 minikube-m02 <none> <none>
fluentd-agent-vlbps 1/1 Running 0 41s 10.244.151.1 minikube-m03 <none> <none>
```

There are also other DS running in K8S

```
$ kubectl get ds -A
// The kube-proxy is part of K8S control plane
// The calico-node agent is one of the components of the calico CNI (container
network interface plugin)
NAMESPACE NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
default fluentd-agent 3 3 3 3 3 <none> 2m52s
kube-system calico-node 3 3 3 3 3 kubernetes.io/os=linux 14h
kube-system kube-proxy 3 3 3 3 3 kubernetes.io/os=linux 14h
```

Clean up. When you delete the DS, it will also delete the pods created by DS.

```
$kubectl delete daemonsets.apps fluentd-agent // or kubectl delete ds fluentd-agent
daemonset.apps "fluentd-agent" deleted
```