15Oct2022
Day09
Kubernetes Workloads V
Kubernetes Resource Manager

Workloads
https://kubernetes.io/docs/concepts/workloads/
A workload is an application running on Kubernetes.
1-Pod
2-ReplicaSet
3-Deployment
5-DaemonSet
6-Init Containers
7-Staticp Pod
8-Job
9-CronJob

Jobs
https://kubernetes.io/docs/concepts/workloads/controllers/job/
-A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. when a specified number of successful completions is reached, the task is complete.
-Deleting a Job will clean up the Pods it created.
ex

```
# kubectl api-resources | grep -i "job"
jobs                          batch/v1                    true       Job
# vim alpinejob.yaml
kind: Job
apiVersion: batch/v1
metadata:
  name: alpinejob
spec:
  template:
    spec:
      containers:
      - name: alpinecnt
        image: alpine
        command: ["/bin/sh","-c"]
        args: ["echo 'Hellooooo!';sleep 15"]
      restartPolicy: Never
:wq!
# kubectl apply -f alpinejob.yaml
# watch kubectl get job
# kubectl get pods -o wide
# kubectl logs alpinejob-sjbjt
```

Parallel execution for Jobs
1-Controlling parallelism
2-Completion mode

1-Controlling parallelism
-all pods in job will start at the same time with enable option ‘parallelism’
-The requested parallelism can be set to any non-negative value.
ex

```
# vim alpinejob.yaml
kind: Job
apiVersion: batch/v1
metadata:
  name: alpinejob
spec:
  parallelism: 4
  template:
    spec:
      containers:
      - name: alpinecnt
        image: alpine
        command: ["/bin/sh","-c"]
        args: ["echo 'Hellooooo!';sleep 15"]
      restartPolicy: Never
:wq!
# kubectl apply -f alpinejob.yaml
# watch kubectl get pods
# watch kubectl get jobs
```

Mohammad Ali Naghval
ali@prodevans.com

2-Completion mode

-with enable ‘completions’ option first, pod1 in job will start and complete then pod2 and3 and 4 and ….

ex

```
# vim alpinejob.yaml
kind: Job
apiVersion: batch/v1
metadata:
  name: alpinejob
spec:
  completions: 4
  template:
    spec:
      containers:
      - name: alpinecnt
        image: alpine
        command: ["/bin/sh","-c"]
        args: ["echo 'Hellooooo!';sleep 15"]
      restartPolicy: Never
:wq!
# kubectl apply -f alpinejob.yaml
# watch kubectl get pods
# watch kubectl get jobs
```

delete Job

```
# kubectl delete jobs.batch alpinejob --force --grace-period=0
```

CronJob

https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

-A CronJob creates Jobs on a repeating schedule.

-One CronJob object is like one line of a crontab file.

```
# cat /etc/crontab
┌───────────── minute (0 - 59)
│ ┌───────────── hour (0 - 23)
│ │ ┌───────────── day of the month (1 - 31)
│ │ │ ┌───────────── month (1 - 12)
│ │ │ │ ┌───────────── day of the week (0 - 6) ‘0 and 7 are Sunday’
│ │ │ │ │
│ │ │ │ │
│ │ │ │ │
* * * * * <command to execute>
```

ex

```
# kubectl api-resources | grep -i "cron"
cronjobs          cj      batch/v1                true      CronJob
# vim rediscronjob.yaml
kind: CronJob
apiVersion: batch/v1
metadata:
  name: rediscj
spec:
  schedule: "51 14 15 10 6"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: rediscnt
            image: redis
          restartPolicy: Never
:wq!
# kubectl apply -f ubuntucronjob.yaml --dry-run=client
# kubectl apply -f ubuntucronjob.yaml
# watch kubectl get cronjobs.batch
# kubectl get pod -o wide
```

Resource Management for Pods and Containers

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

-When you specify a Pod, you can optionally specify how much of each resource a container need.
-The most common resources to specify are:

1-CPU
2-memory (RAM)

Resource types

CPU and memory are each a resource type
A resource type has a base unit.

CPU represents compute processing and is specified in units of Kubernetes CPUs.
CPU measurement is in millicores. with this measure format each **1CPU** is splinted into **1000units(millicores)**
Memory is specified in units of **bytes**.
Memory can express as plain integer in B, K, M, G, T, P

Monitoring resources through Metrix-Server

https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/#metrics-server
https://github.com/kubernetes-sigs/metrics-server

Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

```
# kubectl top nodes
error: Metrics API not available
# kubectl top pod
error: Metrics API not available
```

Installation

https://github.com/kubernetes-sigs/metrics-server

```
# kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
# kubectl get deployments.apps -A
# kubectl get deployments.apps --namespace kube-system
# kubectl top nodes
Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)
# kubectl get deployments.apps --namespace kube-system
# kubectl get pods --namespace kube-system
# kubectl logs pods/metrics-server-678f4bf65b-jnm2f --namespace kube-system
```

to disable tls certification request,

```
# kubectl edit deployments.apps --namespace kube-system metrics-server
42      - --kubelet-preferred-address-types=InternalIP          ->modify to 'InternalIP' remove others
43      - --kubelet-insecure-tls                                ->append this line
:wq!
# kubectl top nodes
Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)
# kubectl rollout restart deployment --namespace kube-system metrics-server
# kubectl top nodes
NAME                 CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master1.example.com  152m         7%     1684Mi          43%
node1.example.com    60m          6%     543Mi           28%
```

Requests and Limits

Requests ->minimum requirement to start

when you specify the resource request for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on.

Limits ->maximum access to resources

when you specify a resource limit for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set.

ex

```
# kubectl run nginx --image nginx -o yaml --dry-run=client >nginxpod.yaml
# vim nginxpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources:
      requests:                                  ->to start, minimum
        cpu: "4m"
        memory: "2Mi"
      limits:                                    ->to make limit, maximum
        cpu: "8m"
        memory: "4Mi"
:wq!
# kubectl top pods
NAME                   CPU(cores)   MEMORY(bytes)
rediscj-27764091-tfq6k 2m           2Mi
```

Mohammad Ali Naghval
ali@prodevans.com