Kubernetes Part-2

By

Prince Khanna

03-Sep-2021

Agenda

+How to go inside a container

+Some more Concepts with Hands On

+Ingress and Egress(if we will have time)

+Exercise

How to go inside a container?

In the exercise, you created a Pod that has one container. The container runs the image provided.

Get a shell to the running container:

kubectl exec -it <PodName> [-c <container name>] -- /bin/bash

Running individual commands in a container

In an ordinary command window, not your shell, list the environment variables in the running container:

kubectl exec <Pod Name> env

Basic Concepts

+ Probes
+ Resources
+ Config Map
+ Secret

Probes

Probes are executed by kubelet to determine pods' health.

All three types of probes have common settings to configure.

+ **initialDelaySeconds**: How many seconds to wait after the container has started (default: 0)
+ **periodSeconds**: Wait time between probe executions (default: 10)
+ **timeoutSeconds**: Timeout of the probe (default: 1)
+ **successThreshold**: Threshold needed to mark the container healthy. (default: 1)
+ **failureThreshold**: Threshold needed to mark the container unhealthy. (default: 3)

Types of Probe

+Command

```
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
```

+Http

```
httpGet:
    path: /healthz
    port: 8080
```

+Tcp

```
tcpSocket:
    port: 8080
```

Sample liveliness probe yaml File

```
livenessProbe:
    httpGet:
      path: /healthz
      port: 8080
      httpxHeaders:
      - name: Custom-Header
        value: Awesome
    initialDelaySeconds: 3
    periodSeconds: 3
```

Resources

When you specify a Pod, you can optionally specify how much of each resource a Container needs. The most common resources to specify are CPU and memory (RAM),

Requests and limits

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its request for that resource specifies. However, a container is not allowed to use more than its resource limit.

Sample Pod with resources yaml File

```
apiVersion: v1
kind: Pod
metadata:
 name: frontend
spec:
 containers:
 - name: app
  image: images:v4
  resources:
   requests:
    memory: "64Mi"
    cpu: "250m"
   limits:
    memory: "128Mi"
    cpu: "500m"
```

Config Map

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

Sample config map yaml File

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: game-demo
data:
 # property-like keys; each key maps to a simple value
 player_initial_lives: "3"
 ui_properties_file_name: "user-interface.properties"

 # file-like keys
 game.properties: |
  enemy.types=aliens,monsters
  player.maximum-lives=5
 user-interface.properties: |
  color.good=purple
  color.bad=yellow
  allow.textmode=true
```

How to Use Config Map

```
apiVersion: v1
kind: Pod
metadata:
 name: configmap-demo-pod
spec:
 containers:
  - name: demo
    image: alpine
    command: ["sleep", "3600"]
    env:
     # Define the environment variable
     - name: PLAYER_INITIAL_LIVES # Notice that the case is different here
                                  # from the key name in the ConfigMap.
       valueFrom:
        configMapKeyRef:
          name: game-demo          # The ConfigMap this value comes from.
          key: player_initial_lives # The key to fetch.
```

Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.

Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

To use a Secret, a Pod needs to reference the Secret. A Secret can be used with a Pod in three ways:

+ As files in a volume mounted on one or more of its containers.
+ As container environment variable.
+ By the kubelet when pulling images for the Pod.

Types of Secret

Builtin Type	Usage
Opaque	arbitrary user-defined data
`kubernetes.io/service-account-token`	service account token
`kubernetes.io/dockercfg`	serialized `~/.dockercfg` file
`kubernetes.io/dockerconfigjson`	serialized `~/.docker/config.json` file
`kubernetes.io/basic-auth`	credentials for basic authentication
`kubernetes.io/ssh-auth`	credentials for SSH authentication
`kubernetes.io/tls`	data for a TLS client or server
`bootstrap.kubernetes.io/token`	bootstrap token data

Sample Secret yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

How to Use Secret

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: username
      - name: SECRET_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: password
  restartPolicy: Never
```

Q&A

References

Get a Shell to a Running Container | Kubernetes

Configure Liveness, Readiness and Startup Probes | Kubernetes

Managing Resources for Containers | Kubernetes

ConfigMaps | Kubernetes

Secrets | Kubernetes

Volumes | Kubernetes