MultiContainerPods:

Multi-container pods provide an opportunity to enhance containers with helper containers that provide additional functionality. This lesson covers the basics of what multi-container pods are and how they are created. It also discusses the primary ways that containers can interact with each other within the same pod, as well as the three main multi-container pod design patterns: sidecar, ambassador, and adapter.

Be sure to check out the hands-on labs for this course (including the practice exam) to get some hands-on experience with implementing multi-container pods.

Container can interact with one other via the following method:

SharedNetwork, SharedStorageVolumes, SharedProcessNamespaces

Pod.

Three different design pattern:

1. Sidecar Pod
2. Ambassador Pod
3. Adaptor pod.

Liveness and Readiness Probe :

Probe: Allow you to customize how Kubernetes determines the status of you container.

LivenessProbe: indicate that container is running properly, and governs when the customer stop and restart the container.

Readiness Probe: indicate whether the container is ready to service requests, and govern whether govern request will be forward to the pod.

Liveness-probe:

apiVersion: v1

kind: Pod

metadata:

name: my-liveness-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "echo Hello, Kubernetes! && sleep 3600"]

livenessProbe:

exec:

command:

- echo

- testing

initialDelaySeconds: 5

periodSeconds: 5

ReadinessProbe:

my-readiness-pod.yml:

apiVersion: v1

kind: Pod

metadata:

name: my-readiness-pod

spec:

containers:

- name: myapp-container

image: nginx

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 5

periodSeconds: 5

Container Logging:

apiVersion: v1

kind: Pod

metadata:

name: counter

spec:

containers:

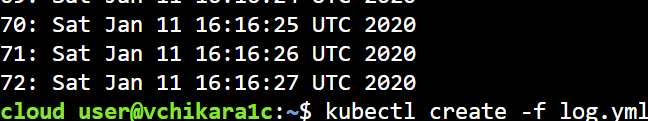
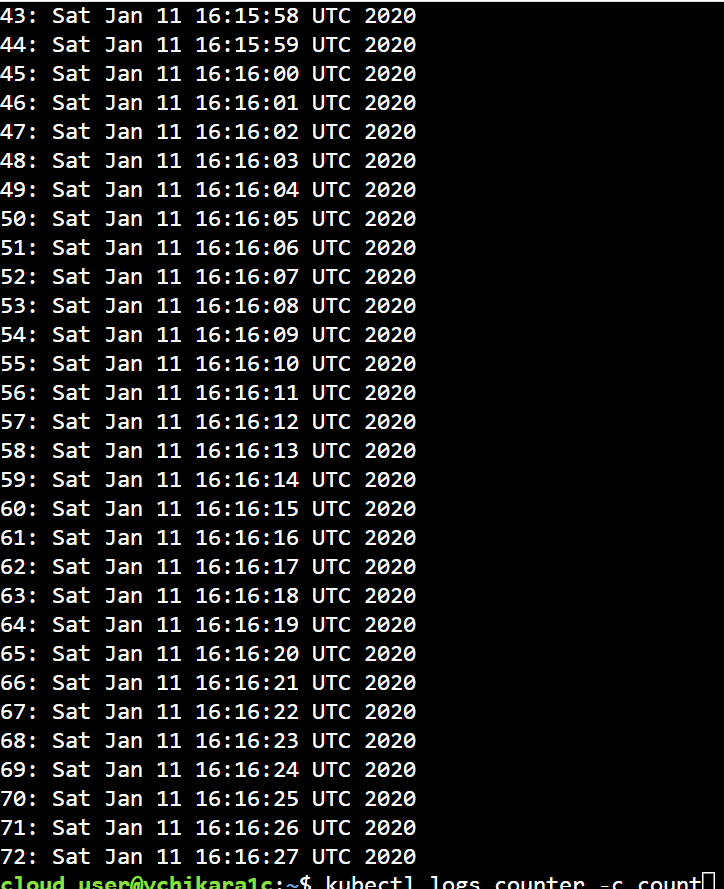
- name: count

image: busybox

args: [/bin/sh, -c, 'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']

kubectl logs counter

kubectl logs <pod name> -c <container name>



Installing metrics Server:

cd ~/

git clone https://github.com/linuxacademy/metrics-server

kubectl apply -f ~/metrics-server/deploy/1.8+/

kubectl get --raw /apis/metrics.k8s.io/

Monitoring Applications:

Kubectl top

apiVersion: v1

kind: Pod

metadata:

name: resource-consumer-big

spec:

containers:

- name: resource-consumer

image: gcr.io/kubernetes-e2e-test-images/resource-consumer:1.4

resources:

requests:

cpu: 500m

memory: 128Mi

- name: busybox-sidecar

image: radial/busyboxplus:curl

command: [/bin/sh, -c, 'until curl localhost:8080/ConsumeCPU -d "millicores=300&durationSec=3600"; do sleep 5; done && sleep 3700']

apiVersion: v1

kind: Pod

metadata:

name: resource-consumer-small

spec:

containers:

- name: resource-consumer

image: gcr.io/kubernetes-e2e-test-images/resource-consumer:1.4

resources:

requests:

cpu: 500m

memory: 128Mi

- name: busybox-sidecar

image: radial/busyboxplus:curl

command: [/bin/sh, -c, 'until curl localhost:8080/ConsumeCPU -d "millicores=100&durationSec=3600"; do sleep 5; done && sleep 3700']

kubectl top pods

kubectl top pod resource-consumer-big

kubectl top pods -n kube-system

kubectl top nodes

kubectl get and kubectl describe to locate a broken pod, and then explore various ways of editing Kubernetes objects to fix issues.

kubectl create namespace nginx-ns

apiVersion: v1

kind: Pod

metadata:

name: nginx

namespace: nginx-ns

spec:

containers:

- name: nginx

image: nginx:1.158

kubectl get pods

kubectl get namespace

kubectl get pods --all-namespaces

kubectl describe pod nginx -n nginx-ns

Pod design:

Label, Selector and Annotation

Kubernetes labels provide a way to attach custom, identifying information to your objects. Selectors can then be used to filter objects using label data as criteria. Annotations, on the other hand, offer a more freeform way to attach useful but non-identifying metadata. In this lesson, we will discuss labels, selectors, and annotations. We will also demonstrate how to use them in a cluster.

apiVersion: v1

kind: Pod

metadata:

name: my-production-label-pod

labels:

app: my-app

environment: production

spec:

containers:

- name: nginx

image: nginx

Pod with different labels:

kubectl get pods -l app=my-app

kubectl get pods -l environment=production

kubectl get pods -l environment=development

kubectl get pods -l environment!=production

kubectl get pods -l 'environment in (development,production)'

kubectl get pods -l app=my-app,environment=production

Pod with annotation:

apiVersion: v1

kind: Pod

metadata:

name: my-annotation-pod

annotations:

owner: terry@linuxacademy.com

git-commit: bdab0c6

spec:

containers:

- name: nginx

image: nginx

you can view labels, selector and annotation

kubectl describe pod my-annotation-pod

Deployments:

Deployment provide a way to declaratively manage a dynamic set of replica pods. They provide a powerful functionality such as scaling and rolling.

Deployment defines a desired state for the replica pods.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

kubectl get deployments

kubectl get deployment <deployment name>

kubectl describe deployment <deployment name>

kubectl edit deployment <deployment name>

kubectl delete deployment <deployment name>