Chapter 10

Best practices for developing apps

실습1: 애플리케이션 개발 주의할 항목

작업1: crash를 계속하는 포드

1. 계속 crash를 발생시키며 사실상 애플리케이션 서비스를 하지 못하는 파드를 컨트롤러가 어떻게 처리하는지 알아본다.

```
# cat > crashingpods.yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: crashing-pods
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: crashing-pods
    spec:
      containers:
      - image: busybox
        name: main
        command:
        - sh
        - -c
        - 'exit 1'
```

2. 위의 예제는 포드가 계속 종료되는 상태이다.
replicaset 컨트롤러는 이러한 파드를 카운트하고 3개의 서비스가 동작중인 것으로 관리한다.

```
# kubectl get pod
NAME                 READY   STATUS             RESTARTS   AGE
crashing-pods-2lljq  0/1     CrashLoopBackOff   2          43s
crashing-pods-6fqw8  0/1     CrashLoopBackOff   2          43s
crashing-pods-xdfrd  0/1     CrashLoopBackOff   2          43s
```

3. ReplicaSet이 어떤 조치를 취했는지 알아본다. 아래 결과에서 볼 수 있듯이 RS는 아무것도 처리하지 않는다.

```
# kubectl describe rs crashing-pods
```

```
Name:         crashing-pods
Namespace:    default
Selector:     app=crashing-pods
Labels:       app=crashing-pods
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=crashing-pods
  Containers:
   main:
    Image:      busybox
    Port:       <none>
    Host Port:  <none>
    Command:
      sh
      -c
      exit 1
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                   Message
  ----    ------            ----   ----                   -------
  Normal  SuccessfulCreate  2m12s  replicaset-controller  Created pod: crashing-pods-2lljq
  Normal  SuccessfulCreate  2m12s  replicaset-controller  Created pod: crashing-pods-6fqw8
  Normal  SuccessfulCreate  2m12s  replicaset-controller  Created pod: crashing-pods-xdfrd
```

4. 이 상황을 쿠버네티스는 어떻게 보고 있을까?

```
# kubectl describe pod crashing-pods-xdfrd
```

```
Name:           crashing-pods-xdfrd
Namespace:      default
Priority:       0
Node:           node22.example.com/10.100.0.22
Start Time:     Thu, 11 Jul 2019 07:54:15 -0400
Labels:         app=crashing-pods
Annotations:    <none>
Status:         Running
IP:             10.42.0.3
Controlled By:  ReplicaSet/crashing-pods
```

위의 Status를 보면 Running 이라고 표시된다. 쿠버네티스는 정상 실행으로 확인하는 것이다.

5. 실습에서 사용한 RS를 삭제한다.

```
# kubectl delete rs crashing-pods
```

```
replicaset.extensions "crashing-pods" deleted
```

작업 2: 초기화 컨테이너 운영을 통해 포드 순서 설정

1. 초기화 컨테이너는 앱 컨테이너들이 실행되기 전에 실행되는 특수한 컨테이너이며, 앱 이미지에는 없는 유틸리티 또는 설정 스크립트 등을 포함할 수 있다.
다음의 yaml 파일을 생성하여 초기화 컨테이너를 구성한다.

```
# cat > fortune-client.yaml
apiVersion: v1
kind: Pod
metadata:
  name: fortune-client
spec:
  initContainers:
  - name: init
    image: busybox
    command:
    - sh
    - -c
    - 'while true; do echo "Waiting for fortune service to come up..."; wget http://fortune -q -T 1 -O /dev/null >/dev/null 2>/dev/null && break; sleep 1; done; echo "Service is up! Starting main container."'
  containers:
  - image: busybox
    name: main
    command:
    - sh
    - -c
    - 'echo "Main container started. Reading fortune very 10 seconds."; while true; do echo "-------------"; wget -q -O - http://fortune; sleep 10; done'
```

2. 생성한 init pod를 실행하고 결과를 확인해보자.

```
# kubectl create -f fortune-client.yaml
pod/fortune-client created
```

두 개의 컨테이너중 하나만 완료되었음을 알려준다.

```
# kubectl get pod
NAME             READY   STATUS     RESTARTS   AGE
fortune-client   0/1     Init:0/1   0          4m1s
```

3. fortune-client 로그를 보면 init 컨테이너 계속 실행중임을 알 수 있다. init 컨테이너가 완료되기 전까지 주 컨테이너는 실행되지 않는다.

 # kubectl logs fortune-client -c init

```
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
```

4. fortune-server를 동작시켜서 init 컨테이너가 종료되고, 주 컨테이너가 동작하도록 구성해보자.

 # kubectl create –f fortune-server.yaml

 # kubectl logs fortune-client -c init

```
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Waiting for fortune service to come up...
Service is up! Starting main container.
```

 # kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
fortune-client	1/1	Running	0	12m
fortune-server	2/2	Running	0	3m31s

5. 실습에 사용한 파드를 모두 삭제한다.

 # kubectl delete pod --all

pod "fortune-client" deleted

pod "fortune-server" deleted

작업 3: Lifecycle hook 사용

post-start, pre-sop hook을 사용하는 nginx 웹서버를 운영한다.

1. 컨테이너 동작과 동시에 poststart hook을 실행하여 /usr/share/message 파일에 메시지를 기록하고, 컨테이너가 종료되기 전에 kill 명령을 통해 nginx를 종료하는 작업을 진행한다.

```
# cat > lifecycle-event.yaml
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo Hello from the postStart handler > /usr/share/message"]
      preStop:
        exec:
          command: ["/bin/sh","-c","nginx -s quit; while killall -0 nginx; do sleep 1; done"]
```

2. 포드를 실행하고 설정한 lifecycle 동작 상태를 확인해보자.

```
# kubectl create -f lifecycle-event.yaml
pod/lifecycle-demo created

# kubectl get pod lifecycle-demo
NAME             READY   STATUS    RESTARTS   AGE
lifecycle-demo   1/1     Running   0          53s

# kubectl exec  lifecycle-demo -- cat /usr/share/message
Hello from the postStart handler

# kubectl delete pod lifecycle-demo
pod "lifecycle-demo" deleted
```

작업 4: 종료 메시지를 작성하는 포드 만들기

컨테이너 종료 시 메시지를 기록하는 포드를 만들어서 운영해본다.

1. 다음과 같이 즉시로 종료하는 포드를 만들고 종료메시지가 기록되도록 해보자.

```
# cat > termination-message.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-termination-message
spec:
  containers:
  - image: busybox
    name: main
    command:
    - sh
    - -c
    - 'echo "I''ve had enough" > /var/termination-reason ; exit 1'
    terminationMessagePath: /var/termination-reason
```

2. 포드를 실행하고 종료메시지를 확인해보자.

```
# kubectl create -f termination-message.yaml
# kubectl describe po pod-with-termination-message
```

```
....
NAME                          READY   STATUS             RESTARTS   AGE
pod-with-termination-message   0/1     CrashLoopBackOff   2          44s
[root@master11 10]# kubectl describe po pod-with-termination-message
Name:         pod-with-termination-message
Namespace:    default
Priority:     0
Node:         node23.example.com/10.100.0.23
Start Time:   Thu, 11 Jul 2019 11:39:58 -0400
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.45.0.5
Containers:
  main:
    Container ID:  docker://ba18852dfed800563b3c449f789abb038710b53865998
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:c94cf1b87ccb80f2e6414
2b8d0fce39f11b9
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      echo "I've had enough" > /var/termination-reason ; exit 1
    State:       Waiting
      Reason:    CrashLoopBackOff
    Last State:  Terminated
      Reason:    Error
      Message:   I've had enough
      Exit Code:    1
```