Advance Deploy

K8s part 3

Env

ซอฟต์แวร์ยุคใหม่นิยมใช้ Environment Variables (Env) สำหรับการตั้งค่าคอนฟิกต่าง ๆ ซึ่ง Kubernetes รองรับการใช้งานรูปแบบนี้อย่างเต็มที่

โดยสามารถกำหนดค่า Env เข้าไปในแต่ละ Pod ได้ผ่านพารามิเตอร์ env และ envFrom

โดย envFrom มักใช้ร่วมกับ ConfigMap หรือ Secret เพื่อให้การจัดการค่าคอนฟิกมีความปลอดภัยและยืดหยุ่นมากขึ้น

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-env-pod
spec:
  containers:
    - name: my-container
      image: nginx
      env:
        - name: APP_MODE
          value: "production"
        - name: LOG_LEVEL
          value: "info"
```

Env

```
controlplane:~$ k get po
NAME   READY   STATUS    RESTARTS   AGE
test   1/1     Running   0          5s
controlplane:~$ k logs test
My env is PRD
controlplane:~$
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args: ["-c", "echo My env is $MY_ENV && sleep
600"]
    env:
    - name: MY_ENV
      value: "PRD"
```

ConfigMaps

คือกลไกของ Kubernetes ที่ใช้เก็บข้อมูลการตั้งค่า (configuration) เช่นไฟล์ config, key-value pairs หรือ environment variables เพื่อให้ container ภายใน pod สามารถนำไปใช้ได้โดยไม่ต้อง hard-code ค่าไว้ใน image

โดยค่าที่เก็บใน ConfigMap จะเป็น **clear text** ซึ่งหมายถึงสามารถอ่านได้ตรง ๆ ไม่ถูกเข้ารหัส

วิธีนำ ConfigMap เข้าไปใช้ใน Pod

1. **Volume Mount**
   - ใช้กรณีที่ต้องการ mount ไฟล์ config เข้าไปใน container เช่น nginx.conf หรือ application.properties
   - ConfigMap จะถูก mount เป็นไฟล์ใน path ที่กำหนด
2. **envFrom**
   - ใช้กรณีที่ต้องการโหลด key-value ทั้งหมดจาก ConfigMap เข้าไปเป็น environment variables
   - เหมาะกับการตั้งค่าหลายตัวพร้อมกัน
3. **env**
   - ใช้กรณีที่ต้องการเลือกบาง key จาก ConfigMap มาเป็น environment variable

ConfigMaps

kubectl create cm my-cm --from-literal=MY_VALUE1=1 --from-literal=MY_VALUE2=2

```
controlplane:~$ k describe cm my-cm
Name:         my-cm
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
MY_VALUE2:
----
2

MY_VALUE1:
----
1


BinaryData
====
```

Volume Mount

```yaml
apiVersion: v1
data:
  MY_VALUE1: "1"
  MY_VALUE2: "2"
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: my-cm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args: ["-c", "echo Value 1 is $MY_VALUE1 and Value 2 is $MY_VALUE2 && sleep 300"]
    volumeMounts:
    - name: my-vol
      mountPath: "/mnt/vol"
      readOnly: true
  volumes:
  - name: my-vol
    configMap:
      name: my-cm
```

```
controlplane:~$ k logs test
Value 1 is and Value 2 is
```

envFrom

```yaml
apiVersion: v1
data:
  MY_VALUE1: "1"
  MY_VALUE2: "2"
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: my-cm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args: ["-c", "echo VALUE 1 is $MY_VALUE1 and VALUE 2 is $MY_VALUE2"]
    envFrom:
      - configMapRef:
          name: my-cm
```

env

```yaml
apiVersion: v1
data:
  MY_VALUE1: "1"
  MY_VALUE2: "2"
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: my-cm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args: ["-c", "echo VALUE 1 is $MY_VALUE1 and VALUE 2 is $MY_VALUE2"]
    env:
    - name: my-cm
      valueFrom:
        configMapKeyRef:
          name: my-cm
          key: MY_VALUE1
          key: MY_VALUE2
```

Secret

ใช้เก็บข้อมูลที่มีความอ่อนไหวเช่น Password, Token or Key โดยไม่ต้องใส่ไปใน Pod

kubectl create secret generic my-secret --from-literal=key1=supersecret

kubectl create secret tls NAME --cert=path/to/cert/file --key=path/to/key/file

kubectl create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL

```
controlplane:~$ kubectl create secret generic my-sec --from-literal=MY_USER=admin --from-literal=MY_PASS=P@ssword
secret/my-sec created
controlplane:~$ k describe secret my-sec
Name:         my-sec
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
MY_PASS:  8 bytes
MY_USER:  5 bytes
```

Volume Mount

```
controlplane:~$ k describe secret my-sec
Name:         my-sec
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
MY_PASS:  8 bytes
MY_USER:  5 bytes
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: test
  name: test
spec:
  volumes:
    - name: sec-volume
      secret:
        secretName: my-sec
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args:
      - -c
      - |
        echo "User is $(cat /mnt/sec/MY_USER)"
        echo "Pass is $(cat /mnt/sec/MY_PASS)"
        sleep 600
    volumeMounts:
      - name: sec-volume
        mountPath: "/mnt/sec"
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
controlplane:~$ k logs test
User is admin
Pass is P@ssword
controlplane:~$
```

envFrom

```
controlplane:~$ k describe secret my-sec
Name:         my-sec
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
MY_PASS:  8 bytes
MY_USER:  5 bytes
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args:
      - -c
      - |
        echo "User is $MY_USER"
        echo "Pass is $MY_PASS"
        sleep 600
    envFrom:
      - secretRef:
          name: my-sec
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
controlplane:~$ k logs test
User is admin
Pass is P@ssword
controlplane:~$
```

env

```
controlplane:~$ k describe secret my-sec
Name:         my-sec
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
MY_PASS:  8 bytes
MY_USER:  5 bytes
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: test
  name: test
spec:
  containers:
  - image: busybox
    name: test
    command: ["/bin/sh"]
    args:
      - -c
      - |
        echo "User is $MY_USER"
        echo "Pass is $MY_PASS"
        sleep 600
    env:
      - name: MY_USER
        valueFrom:
          secretKeyRef:
            name: my-sec
            key: MY_USER
      - name: MY_PASS
        valueFrom:
          secretKeyRef:
            name: my-sec
            key: MY_PASS
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
controlplane:~$ k logs test
User is admin
Pass is P@ssword
controlplane:~$
```

Labels

คือ key-value ที่ใช้ติดแท็กให้กับ Kubernetes object เช่น Pod, Service, Deployment ฯลฯ เพื่อระบุคุณสมบัติหรือกลุ่มของ object นั้น ๆ

ทำอะไรได้บ้าง?

- จัดกลุ่ม resource
- ใช้สำหรับค้นหา (query) object
- ใช้ร่วมกับ selector เพื่อควบคุมว่า resource ไหนจะถูกจัดการ

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: test
  name: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: test
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

Selector

คือเงื่อนไขที่ใช้เลือก object ที่มี label ตรงตามที่กำหนด เช่น Deployment จะใช้ selector เพื่อเลือก Pod ที่จะควบคุม

ประเภทของ selector:

- matchLabels: ต้องตรงทุก key-value

- matchExpressions: ใช้เงื่อนไขเชิงตรรกะ เช่น In, NotIn, Exists

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: test
  name: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: test
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

Strategy (Recreate)

Recreate Strategy

ลักษณะการทำงาน:

- Kubernetes จะ หยุด (terminate) Pod เก่าทั้งหมดก่อน

- แล้วจึง สร้าง Pod ใหม่ทั้งหมด ขึ้นมาแทน

ข้อดี:

- ง่ายและตรงไปตรงมา

- เหมาะกับ workload ที่ไม่ต้องการ availability ระหว่างการ deploy เช่น batch jobs

ข้อเสีย:

- มี downtime ชั่วคราว เพราะไม่มี Pod ให้บริการระหว่างการเปลี่ยน

- ไม่เหมาะกับระบบที่ต้องการ high availability

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: test
  name: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: test
    spec:
      containers:
      - image: nginx
        name: web
        resources: {}
status: {}
```

Output

```
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  39s   deployment-controller  Scaled up replica set test-cc7b7fdd9 from 0 to 2
  Normal  ScalingReplicaSet  8s    deployment-controller  Scaled down replica set test-cc7b7fdd9 from 2 to 0
  Normal  ScalingReplicaSet  7s    deployment-controller  Scaled up replica set test-7ff5d44c7 from 0 to 2
```

Strategy (Rolling)

RollingUpdate Strategy (ค่า default)

ลักษณะการทำงาน:

- Kubernetes จะค่อย ๆ สร้าง Pod ใหม่ทีละชุด
- แล้วค่อย ๆ ลบ Pod เก่าออก ตามเงื่อนไขที่กำหนด

มีสองพารามิเตอร์หลัก:

- maxUnavailable: จำนวน Pod ที่สามารถหยุดชั่วคราวได้ระหว่างการอัปเดต
- maxSurge: จำนวน Pod ใหม่ที่สามารถสร้างเกินจำนวนที่กำหนดได้

ข้อดี:

- ไม่มี downtime ถ้าค่าถูกตั้งอย่างเหมาะสม
- เหมาะกับระบบ production ที่ต้องการความต่อเนื่อง

ข้อเสีย:

- อาจใช้เวลานานกว่าการ deploy แบบ Recreate
- ต้องระวังเรื่อง resource ถ้า maxSurge สูงเกินไป

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: test
  name: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: test
    spec:
      containers:
      - image: nginx
        name: web
        resources: {}
status: {}
```

Output

```
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  32s   deployment-controller  Scaled up replica set test-cc7b7fdd9 from 0 to 2
  Normal  ScalingReplicaSet  13s   deployment-controller  Scaled up replica set test-7ff5d44c7 from 0 to 1
  Normal  ScalingReplicaSet  13s   deployment-controller  Scaled down replica set test-cc7b7fdd9 from 2 to 1
  Normal  ScalingReplicaSet  13s   deployment-controller  Scaled up replica set test-7ff5d44c7 from 1 to 2
  Normal  ScalingReplicaSet  5s    deployment-controller  Scaled down replica set test-cc7b7fdd9 from 1 to 0
```

HPA

คือ auto scale ของ k8s ที่ใช้งานร่วมกับ Metric server สามารถตั้ง scale up and scale down

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: test-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: test
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 30
      policies:
        - type: Percent
          value: 70
          periodSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
        - type: Percent
          value: 50
          periodSeconds: 60
```

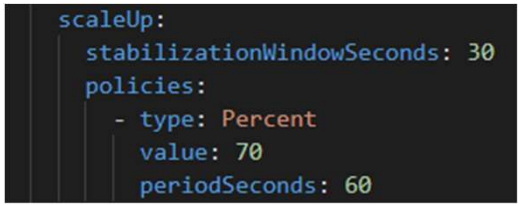
HPA

stabilizationWindowSeconds: รอ 30s ก่อนสั่ง scale

type: Percent

value: เพิ่มเมื่อ CPU 70%

periodSeconds: เพิ่ม pod ทุกๆ 60s



HPA

stabilizationWindowSeconds: รอ 5 นาทีก่อน scale down

type: Percent

value: เมื่อ CPU 50%

periodSeconds: scale ทุกๆ 60s