Kubernetes Workloads

KEEPCODING
Tech School

Kubernetes Workloads

- En esta sección:
  - Controladores de carga de trabajo
    - Deployment
    - DaemonSet
    - StatefulSet
    - Job & CronJob
  - Recomendaciones de diseño
  - Ejercicios

Kubernetes Workloads

- Controladores de PODs
  - Como hemos comentado, generalmente nunca crearemos pods directamente, sino que crearemos tipos de objetos capaces de manejar los pods y cuidar de ellos.

  - En inglés no se llaman realmente "controllers" sino que se habla directamente de "workload resources"

  - https://kubernetes.io/es/docs/concepts/workloads/controllers/

    Kubernetes Documentation / Concepts / Workloads / Workload Resources / Deployments

Deployment

KEEPCODING
Tech School

Deployment

- Un deployment es un tipo de controlador que nos permite de manera declarativa manejar Pods y ReplicaSets.
- Se establece un estado deseado, y el controlador de Deployment se encarga de que los Pods que “están a su cargo” alcancen dicho estado.

```
$ kubectl get pods
$ kubectl delete pod # para reiniciar pods del deployment

# Crea servicio via kubectl expose
$ kubectl expose deploy nginx --port=80 --target-port=80
$ kubectl get svc

/ # curl nginx
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Deployment

- Características:
  - Típicamente usado para aplicaciones "stateless", efímeras.
  - No tienen identidad individual.
  - Pods con nombre dinámico, no pueden ser accedidos directamente por otros pods (ya que se desconocen los nombres)
  - Aceptan volúmenes persistentes, **pero**:
    - El mismo volumen para todas las réplicas
    - https://akomljen.com/kubernetes-persistent-volumes-with-deployment-and-statefulset/
  - Scaling up & down, rollbacks, rolling-updates, HPA, affinity rules y toda la funcionalidad de los pods.
- Importante
  - **.spec.template → pod definition (metadata.labels obligatorio)**
  - **.spec.selector → Obligatorio, ha de apuntar a los pods.**

Deployment Spec

- minReadySeconds
- paused
- progressDeadlineSeconds
- **replicas**
- revisionHistoryLimit
- **selector**
  - matchExpressions
  - matchLabels
- strategy
  - rollingUpdate
    - maxSurge
    - maxUnavailable
  - type
- **template → podTemplateSpec (metadata + spec del pod)**

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#deployment-v1-apps

Deployment - Demo

- Demo!

DaemonSet

KEEPCODING
Tech School

DaemonSet

- La declaración es parecida a la del Deployment, pero mantiene **1 réplica por nodo** de Kubernetes.
- Son usados sobre todo para desplegar servicios que queremos que estén presentes en todos los nodos del clúster, por ejemplo: agentes de monitorización, agentes de logs, etc

```
$ kubectl get ds
$ kubectl describe ds
```

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

DaemonSet

- Características:
  - Generalmente asociados a los nodos del cluster.
  - Útiles para funcionalidad a nivel de host, como monitorización, gestión de logs, etc.
  - Nos aseguran que todos los nodos de nuestro clúster (o algunos, en función de lo que necesitemos), tengan corriendo una copia de un pod
  - El ciclo de vida de estos pods está unido al ciclo de vida de los nodos.
  - Por lo demás son como deployments, pero con **réplicas "1 por nodo"**
  - **Por lo general nunca necesitaremos un DaemonSet excepto si somos los administradores del sistema y es para algo relacionado con los propios nodos de Kubernetes.**

DaemonSet Spec

- minReadySeconds
- revisionHistoryLimit
- selector
  - matchExpressions
  - matchLabels
- updateStrategy
  - rollingUpdate
    - maxSurge
    - maxUnavailable
  - type
- **template → podTemplateSpec (metadata + spec del pod)**

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#daemonset-v1-apps

DaemonSet - Demo

- Demo!
- Otros ejemplos:
  - Fluentd DaemonSet (ejemplo básico doc kubernetes)
  - Filebeat DaemonSet
  - Metricbeat DaemonSet

Estos ejemplos son más complicados de entender al utilizar un montón de funcionalidades. Los analizaremos más adelante.

StatefulSet

KEEPCODING
Tech School

StatefulSet

- Al igual que el Deployments, es un tipo de controlador que nos permite de manera declarativa manejar Pods.
- Los vamos a usar **para aplicaciones o sistemas con estado, donde cada pod del grupo sea único y necesite manejar sus propios datos para funcionar**.

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

StatefulSet

- Características:
  - Típicamente usados para aplicaciones con estado, que necesitan persistencia de datos para mantener su estado.
  - Nombres de pods estáticos / conocidos. Cada pod tiene identidad e individualidad!
  - Rolling updates y escalados controlados.
  - Garantiza el orden y la unicidad de los pods con identificadores persistentes.
  - Permiten storage persistente, **dedicado por pod**
    - .spec.volumeClaimTemplates
    - Cada réplica tendrá su propio volumen persistente

- Importante
  - **Requiere un servicio headless** (* clusterIP: None *)

StatefulSet

```yaml
# HEADLESS SERVICE
apiVersion: v1
kind: Service
metadata:
  name: nginx-headless
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx-headless"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

StatefulSet Spec

- podManagementPolicy
- replicas
- revisionHistoryLimit
- selector
  - matchExpressions
  - matchLabels
- serviceName → **Importante!!** → **obligatorio**
- updateStrategy
  - rollingUpdate
    - partition
  - type
- template → **podTemplateSpec (metadata + spec del pod)**
- volumeClaimTemplates: → Solicitud de storage
  - metadata
  - spec (volumeClaimSpec)

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#statefulset-v1-apps

StatefulSets - Demo

- Demo!

Job & CronJob

KEEPCODING
Tech School

Job & CronJob

- Para procesos batch, aplicaciones o scripts que han de terminar en algún momento.
- Los CronJob permiten ejecución periódica.
- Jobs:
  - Crean uno o más pods y se reintentan hasta que un número determinado de ellos termine correctamente.
  - Caso simple: Job que lanza un pod hasta que se complete correctamente.
- CronJob: Job + schedule, para ejecución periódica.

Job & CronJob

- Casos de uso típicos:
  - Trabajo simple sin paralelización (se arranca un pod).
  - Múltiples pods trabajando en paralelo
    - Hasta que '.spec.completions' hayan terminado
    - Cola de trabajo (sin especificar '.spec.completions').
      - Se tienen que coordinar entre ellos
      - Se supone que todos tendrán que acabar.
      - Cuando un pod termina no se crea otro nuevo.

Job Spec

- `activeDeadlineSeconds` → Tiempo que se le permite estar al job en ejecución.
- `backoffLimit` → Número de fallos permitidos antes de considerar el job fallado.
- `completions` → Número de pods que han de finalizar correctamente para considerar el job finalizado.
- `parallelism` → Número de pods a lanzar en paralelo
- `template` → **podTemplateSpec (metadata + spec del pod)**
- `ttlSecondsAfterFInished` → Segundos tras finalización antes de ser eliminado
- `suspend` → Para suspender el job

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#jobspec-v1-batch

CronJob Spec

- `concurrencyPolicy —>` Permitir concurrencia
- `failedJobHistoryLimit —>` Número de ejecuciones fallidas a guardar
- `jobTemplate —>` JobTemplateSpec (metadata + job spec)
- `schedule —>` Planificación "cron" del job.
- `startingDeadlineSeconds —>` Tiempo extra por si se pasa la hora-
- `successfulJobHistoryLimit—>` Número de ejecuciones success a guardar
- `suspend—>` Para suspender el CronJob

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#cronjobspec-v1-batch

Jobs & CronJobs - Demo

- Demo!

Conclusiones Workloads

KEEPCODING
Tech School

Recomendaciones de diseño

- **No utilices pods** directamente, ya que no serán reprogramados
- Usa nombres DNS para acceder a los servicios (no variables de entorno).
- **No** utilices **hostPort** ni **hostNetwork** salvo absoluta necesidad.
- Utiliza bien las etiquetas (labels), son muy útiles para seleccionar y filtrar y se pueden actualizar dinámicamente (`kubectl label`).
- Generalmente un pod ejecuta un solo contenedor y un solo proceso (microservicios). En los casos en los que hay mútiples contenedores, uno suele ser el contenedor principal y el resto son auxiliares (complementan al principal de alguna forma, a veces se les llama 'sidecars').

Recomendaciones de diseño

- Deployment vs StatefulSet
  - ¿Tiene cada réplica un estado individual e independiente persistente? → StatefulSet
  - ¿Necesito conocer en todo momento el nombre de las réplicas? → StatefulSet
- Deployment vs DaemonSet
  - ¿Necesito un pod por cada nodo y además tienen que ver con los propios nodos de Kubernetes? → DaemonSet
  - ¿No quiero que haya más de un pod en el mismo nodo? → No implica DaemonSet!
- PersistentVolume vs Volume
- Probes (readiness / liveness)
- Considera el uso de Init containers
- Patrones de diseño:
  - Sidecar containers → Complementan la funcionalidad
  - Embajadores → Proveen el acceso a los contenedores
  - Adaptadores → Adaptan protocolos

Ejercicios workloads

KEEPCODING
Tech School

2.4.1 Ejercicios con Deployments

1. Crea un deployment con la imagen **“nginx:1.7.8”**, con 2 réplicas, y que defina el puerto 80.
2. Muestra el YAML del deployment creado
3. Muestra el YAML del replica set que ha creado este deployment.
4. Muestra el YAML de alguno de los Pods.
5. Mira a ver el estado de rollout del deployment.
6. Ahora actualiza el deployment a **“nginx:1.7.9”**
7. Muestra el historial de rollout, y comprueba que las réplicas están OK.
8. Haz un rollback a la versión anterior.
9. Actualiza a una versión que no existe, por ejemplo “nginx:1.91”
10. Verifica que algo va mal.

2.4.1 Ejercicios con Deployments

11. Vuelve a la revisión nº 2 y verifica que la imagen es la 1.7.9
12. Comprueba los detalles de la revisión nº3
13. Escala el deployment a 5 réplicas.
14. Autoescala el deployment, entre 5 y 10 réplicas, y CPU al 80% (*)
15. Pausa el rollout del deployment
16. Actualiza la imagen a nginx:1.9.1 y comprueba que no pasa nada (hemos pausado)
17. Continúa el rollout, y comprueba que se aplica la versión 1.9.1.
18. Elimina el deployment y el HPA.

2.4.2 Ejercicios con Jobs

1. Crea un job con la imagen busybox, y que ejecute el comando 'echo Dev;sleep 30;echo Ops'
2. Muestra los logs en tiempo real (hasta que acabe)
3. Comprueba el estado del job, haz un describe y luego mira los logs.
4. Elimina el job
5. Crea el mismo job, pero que se ejecute 5 veces una después de la otra. Verifica el estado y elimínalo.
6. Igual que (7) pero que se ejecuten en paralelo.

2.4.3 Ejercicios con CronJobs

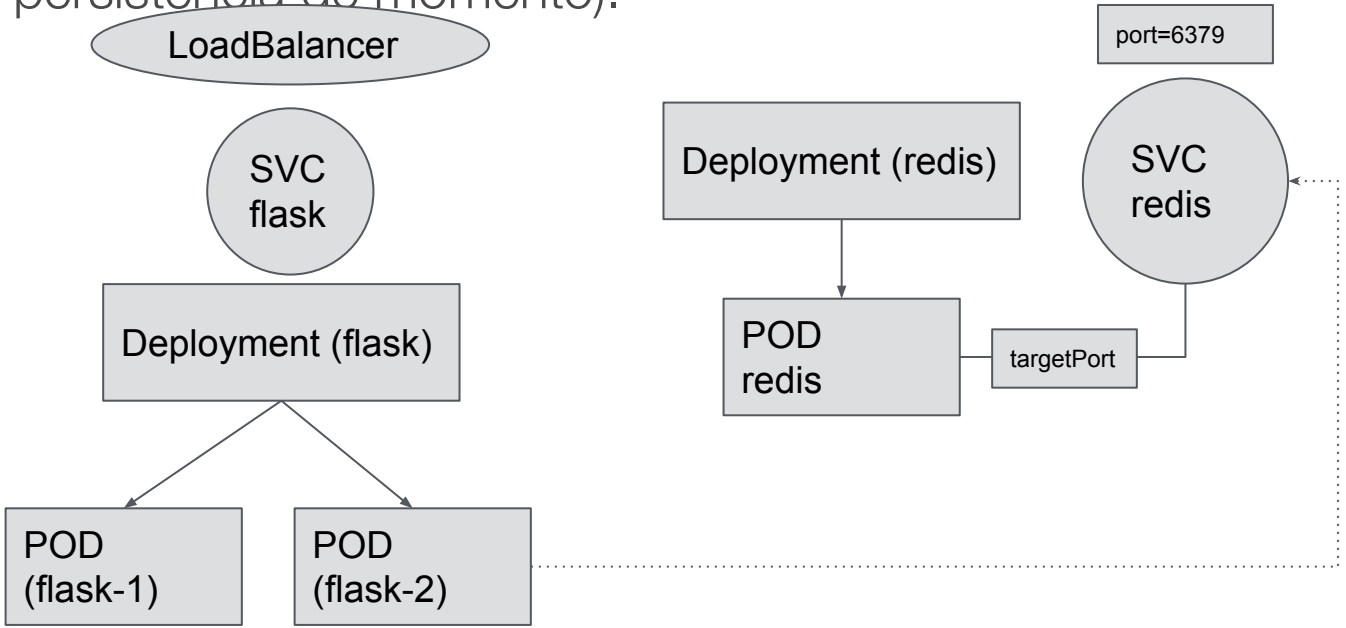
1. Crea un CronJob con la imagen busybox, que se ejecute cada hora a y 45 y muestre por pantalla la fecha con la hora.
2. Muestra los logs y luego lo eliminas.

2.4.4 Ejercicios con Servicios

1. Crea un Deployment con la imagen Nginx y un servicio **ClusterIP** en el puerto 80
2. Chequea el servicio y los endpoints
3. Obtén la IP del servicio, crea un POD con busybox y haz un curl usando tanto la IP como el nombre DNS.
4. Convierte el servicio a NodePort, observa las diferencias. Intenta acceder ahora desde el exterior del cluster.
5. Crea un deployment llamado “foo” con la imagen “dgkanatsios/simpleapp” y 3 réplicas. Etiquétalo con “app=foo”. Declara que los contenedores del port pueden aceptar tráfico en el puerto 8080.
6. Obtén las direcciones IP de los PODs.
7. Crea un servicio "foo" que escuche en el puerto 6262 y apunte a los pods del deployment foo.
8. Conecta desde otro POD al DNS del servicio "foo".

2.4.5

Despliega la aplicación de flask con contador y redis en Kubernetes (sin persistencia de momento).



2.4.6

- Crea un StatefulSet de nginx con 2 réplicas y PersistentVolumeClaim de disco de 1G.
- Actualiza a 3 réplicas.
- Desde un pod con busybox haz un ping a la réplica 1 directamente.
- Identifica el servicio headless asociado al StatefulSet

2.4.7

- Despliega un DaemonSet con la imagen de nginx en todos los nodos.
- Despliega únicamente el DaemonSet anterior en los nodos con label disk=ssd.

KEEPCODING

Tech School

Madrid | Barcelona | Bogotá