POD Scheduling

KEEPCODING
Tech School

Kubernetes Workloads

- En esta sección:
  - Scheduling avanzado en Kubernetes
  - Reglas de afinidad / anti-afinidad
    - A nivel de nodo
    - A nivel de pod
  - Taints y Tolerations

Scheduling avanzado en Kubernetes

El scheduler (programador / planificador) por defecto de Kubernetes funciona generalmente bien en la mayoría de lo casos, por ejemplo, se asegura de que los PODs sólo vayan a nodos que tengan suficientes recursos libres, intenta distribuir los PODs del mismo controlador entre todos los nodos e intenta balancear la utilización de recursos de todos los nodos.

Sin embargo, a veces queremos controlar la manera o el lugar en la que esos PODs se ejecutan, por ejemplo, que se ejecuten en nodos con un hardware especial, o que unos PODs estén siempre cerca de otros si van a tener que comunicarse mucho.

Desde Kubernetes 1.6 disponemos de varias funcionalidades para scheduling avanzado que veremos a continuación.

Reglas de afinidad

- Affinity
  - En base a host (nodeAffinity spec)
  - En base a pods (podAffinity y podAntiAffinity specs)
- Desde el punto de vista del **POD**, ayuda a determinar a qué nodos se puede juntar o a qué pods se debe alejar o separar.

Node affinity

KEEPCODING
Tech School

Reglas de afinidad (node affinity rules)

- Determinan en qué nodos se puede lanzar el pod (sin tener en cuenta otros pods). Expande la funcionalidad de nodeSelector.
- Se aplican a pods sobre los labels de los propios nodos.
  - Las reglas de tipo **required** hacen que si ningún nodo las satisface, entonces el POD no se ejecutará (**hard affinity** / afinidad dura)
  - Las reglas de tipo **preferred** intenta que se ejecute siempre en el nodo que indique la regla, pero si por el contrario no se cumple, utilizará otro nodo disponible (**soft affinity** / afinidad blanda).

https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity

Reglas de afinidad (node affinity rules)

```
~$ kubectl describe node
~$ kubectl label node key=value
```

spec.affinity.nodeAffinity dentro del POD:

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
          - key: "failure-domain.beta.kubernetes.io/zone"
            operator: In
            values: ["europe-west1-d”]
```

```
affinity:
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - weight: 1
       preference:
         matchExpressions:
         - key: another-node-label-key
           operator: In
           values:
           - another-node-label-value
```

Operadores válidos: In, NotIn, Exists, DoesNotExist, Gt, Lt

Labels por defecto

Por defecto Kubernetes pre-configurará una serie de labels por defecto en los nodos:

- kubernetes.io/hostname
- failure-domain.beta.kubernetes.io/zone
- failure-domain.beta.kubernetes.io/region
- beta.kubernetes.io/instance-type
- kubernetes.io/os
- kubernetes.io/arch

```
~$ kubectl describe node
```

Pod affinity

KEEPCODING
Tech School

POD affinity / anti-affinity

Objetivo: determinar en qué nodo(s) se puede ejecutar un pod en función de otros pods.

¿Y si quisiéramos que se ejecutaran los PODs en función de otros PODs y donde se estén ejecutando? Para ello podemos usar **pod affinity / anti-affinity**.

Por ejemplo, tenemos un Pod con label app=P1 que se comunica frecuentemente con otro pod con label app=P2 (un patrón llamado “north-south”). Queremos que estén en la misma zona dentro de nuestro clúster por si alguna de las zonas sufre indisponibilidad que no lo sufra nuestro servicio.

1. Añadimos a los Pods de P2 una etiqueta que diga: app=P2
2. Añadimos la siguiente regla de podAffinity:

```
affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values: ["P1"]
        topologyKey: failure-domain.beta.kubernetes.io/zone
```

POD affinity / anti-affinity

Podemos tener **podAffinity** y **podAntiAffinity**.

Acciones:

- `requiredDuringSchedulingIgnoredDuringExecution`
- `preferredDuringSchedulingIgnoredDuringExecution`

Operadores permitidos: **In, NotIn, Exists, DoesNotExist**

¿Si queremos que esté en el mismo nodo?
¿Si queremos que se distribuya entre todos los nodos de forma forzada?

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-domain.beta.kubernetes.io/zone
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
      image: k8s.gcr.io/pause:2.0
```

Affinity demo

- Demo!

Taints y Tolerations

KEEPCODING
Tech School

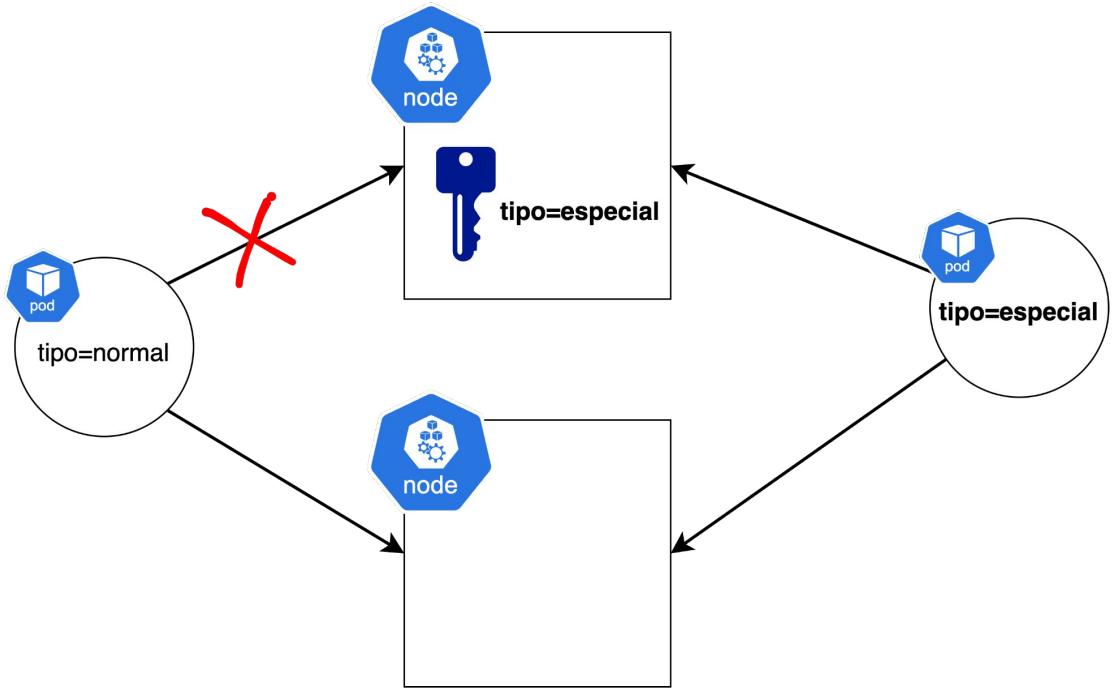
Taints & Tolerations

- Scheduling desde el punto de vista del **nodo**. Permite a los nodos **rechazar** pods.
  - nodeAffinity permite atraer a los pods a nodos.
  - **Taints** son lo opuesto, permiten a nodos repeler pods.
- Cuando un nodo tiene un taint, hace falta que un pod tenga un toleration para aceptar el taint concreto.
- Los **Tolerations** se aplican a pods y permiten ser asignados (pero no requieren) en nodos con taints coincidentes.

- Casos de uso:
  - Que en el Master no se ejecute ningún POD, excepto aquellos que sean de sistema.
  - Un grupo de nodos dedicados a una aplicación o a un grupo de usuarios
  - Que PODs normales no hagan uso de nodos que tengan un hardware especial

https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Taints & Tolerations



https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Trabajando con Taints

Podemos añadir un taint a un nodo.
El formato de un taint es **`key=val:effect`**

```
$ kubectl taint node NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N
```

Los taints effects (cómo reaccionar a pods sin toleration) pueden ser:

- **NoSchedule** → no se podrá asignar a nodo
- **PreferNoSchedule** → preferiblemente no se asignará
- **NoExecute** → si hay pods en ejecución se desalojan (evict)

```
$ kubectl taint node node1 app=machineLearning:NoSchedule
```

https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#taint

Añadiendo tolerations

Podemos un toleration a un POD.
Para que pueda ejecutarse en un nodo con un taint debe tener exactamente la misma especificación el toleration

```
tolerations:
 -    key: "key"
      operator: "Equal"
      value: "value"
      Effect: "NoSchedule"
```

Los taints effects pueden ser:

- NoSchedule
- PreferNoSchedule
- NoExecute

Casos especiales:

```
tolerations:
 -    operator: Exists
```

Tolera todo

```
tolerations:
 -    key: "key"
      operator: Exists
```

Tolera todos los values

https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#taint

Ejemplo Taints y Tolerations

Si creamos los siguientes taints a un nodo y un pod con 2 tolerations:

```
~$ kubectl taint nodes node1 key1=value1:NoSchedule
~$ kubectl taint nodes node1 key1=value1:NoExecute
~$ kubectl taint nodes node1 key1=value2:NoSchedule
```

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

En este caso el POD no se podrá ejecutar en el nodo, porque no existe ningún toleration que haga match en el tercer taint.

Podrá seguir ejecutándose si ya está en ejecución en el nodo, pero no si está recién creado.

¡Ojo! Si añadimos un taint de tipo NoExecute a un nodo, todos los pods que no lo toleren van a ser desahuciados (evited) de forma inmediata.

Podemos añadir una 5ª propiedad, **tolerationSeconds** que indica por cuánto tiempo el POD seguirá ejecutándose en el nodo en el que está.

Casos de uso

- **Nodos dedicados**: si quieres dedicar un grupo de nodos para un grupo particular de usuarios, añade un taint (dedicated=groupName:NoSchedule) y la correspondiente toleration en sus PODs.

- **Nodos con hardware específico**: en este caso no van a estar dedicados únicamente a los PODs que lo requieran, querremos que tengan preferencia, pero si no hay hueco libre en el resto permitamos que se puedan usar. Podemos crear un taint de tipo (special=true:PreferNoSchedule).

Custom schedulers

Si todas las funcionalidades que acabamos de ver que están integradas de forma nativa en Kubernetes aún así no son suficiente para nosotros, siempre podemos desarrollar nuestro propio Scheduler y delegar la responsabilidad de elegir el nodo en él.

Desde Kubernetes 1.6 podemos tener múltiples schedulers en el mismo clúster de Kubernetes, sólo tendremos que elegir cuál es el que queremos usar en la definición del POD bajo **spec.schedulerName**.

No hay ningún requisito para ejecutar un custom scheduler, pueden estar escritos en cualquier lenguaje!

Ejercicios Scheduler

KEEPCODING
Tech School

Scheduling - 1

A) Crea un pod con que requiera un nodo con label “keepcoding=true”. Crea un nodo con esa label

A) Igual pero preferred.

Scheduling - 2

Crea un nodo con un taint de NoSchedule
Crea pods que se ejecuten en ese nodo

Scheduling - 3

Crea dos deployments D1 y D2
Haz que los pods de D2 se ejecuten siempre en nodos donde haya algún pod de D1.

Scheduling - 4

Crea un deployment que todos sus pods se ejecuten distribuidos en distintos nodos siempre. Comprueba que eso funciona así.

KEEPCODING

Tech School

Madrid | Barcelona | Bogotá