KUB 16: Запуск подов на определенных узлах

Описание:

NodeSelector
Совсем недавно мы с вами обсуждали тему Labels (меток) и называли, что они есть у всех ресурсов Kubernetes и используются для группировки ресурсов. Эта группировка применяется, как правило, другими сущностями — к примеру, метки у подов используются у Services для того, чтобы определять, на какие поды можно направлять нагрузку. В этом задании мы с вами разберем еще один способ использования меток, но для Node — NodeSelector.
nodeSelector разрешает назначить на под (или на шаблон пода в рамках Deployment) условие, которое определит, на каких нодах он должен запускаться. Определяется этот параметр на уровне spec пода.
Раз уж мы подняли вопрос меток у нод, стоит вспомнить, что у всех нод есть свой набор меток, который абсолютно также можно использовать в nodeSelector. Так, к примеру, существует метка kubernetes.io/hostname, которая содержит hostname и, соответственно, единствена для каждой ноды. Она позволяет при необходимости назначить приложение на конкретную ноду.
Пример:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  nodeSelector:
    kubernetes.io/hostname: cl1ihia34v9pbjll4oic-yhes
```

NodeAffinity
nodeAffinity похож на nodeSelector, но работает немного иначе. Этот алгоритм более гибкий и использует метки узла для выбора. Например, у вас есть узлы с меткой gpu=true, а также с автоматическими метками failure-domain.beta.kubernetes.io/region, failure-domain.beta.kubernetes.io/zone, kubernetes.io/hostname (в облаках они добавляются автоматически). Простой nodeSelector: gpu=true на Affinity будет выглядеть так:

```
apiVersion: v1
kind: Pod
```

```yaml
metadata:
  name: kubia-gpu
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: gpu
            operator: In
            values:
            - "true"
```

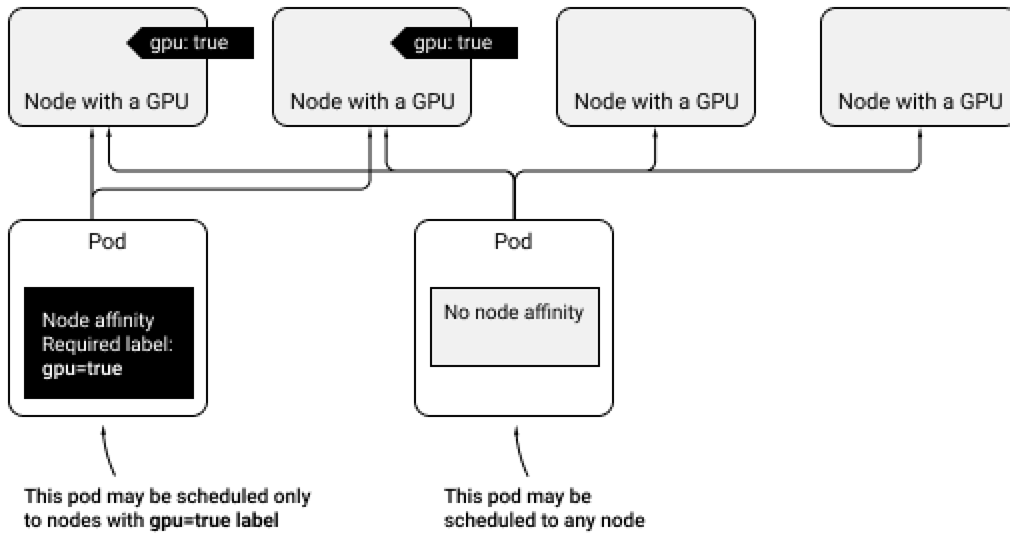
Принципы построения affinity:

1. Когда будет обрабатываться правило: DuringScheduling - отрабатывает в случае, если pod еще не запущен. Scheduler учитывает условия и выбирает ноду для размещения. DuringExecution - если pod уже запущен и у него поменялись affinity, scheduler повторно просчитывает все affinity и если текущая нода не удовлетворяет требованиям, он будет перезапущен на другой
2. Строгость условия: required (требуемый "must-have") preferred (желаемый) результат. В случае required, если не будут удовлетворены условия, то pod не запустится. Preferred - указывает предпочтения, которые scheduler пытается удовлетворить, если требования не выполнимы, то под запустится на любой ноде. Второй частью выражения является указание того, в какой момент мы должны применить правила

В настоящий момент наиболее популярные affinity:

- requiredDuringSchedulingIgnoredDuringExecution — требовать какие-то условия при запуске и не запускать под, если требования не удовлетворены. Запущенные поды остаются без изменений
- preferredDuringSchedulingIgnoredDuringExecution — желаемые условия, запустить где угодно, если не требования не удовлетворены. Запущенные поды остаются без изменений
- requiredDuringSchedulingrequiredDuringExecution - Pod запустится при соблюдениии всех условий, запущенные поды будут рестартованы.
- preferredDuringSchedulingpreferredDuringExecution - Попытаться найти лучшее расположение, при первичном размещении и, если для запущенных pods будет более выгодные условия, пенести их на новые ноды

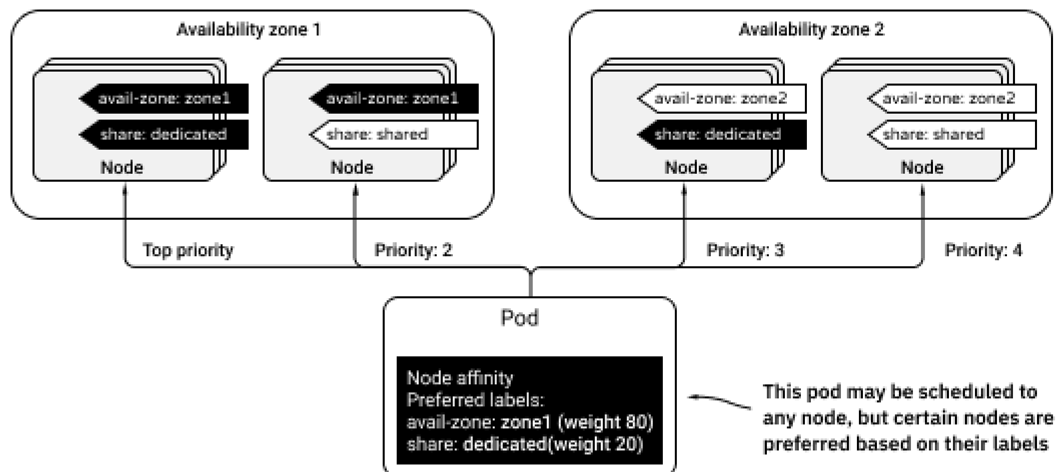
Практически всегда используется ...IgnoredDuringExecution, чтобы в случае ошибки не спровоцировать массовую миграцию pods или же не задеплоив pod не туда "убить production" :)



И даже более того, можно дополнительно управлять «весом» узла в алгоритме балансировки. Например:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: pref
spec:
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 80
            preference:
              matchExpressions:
              - key: availability-zone
                operator: In
                values:
                - zone1
          - weight: 20
            preference:
              matchExpressions:
              - key: share-type
                operator: In
                values:
                - dedicated
```

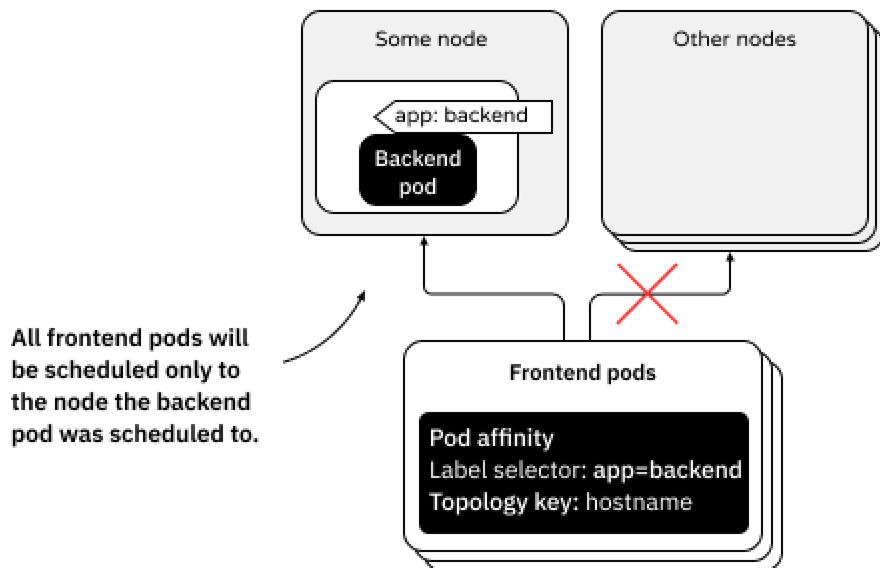
Это описание означает, что приоритет получают узлы с меткой zone1 и меткой dedicated.



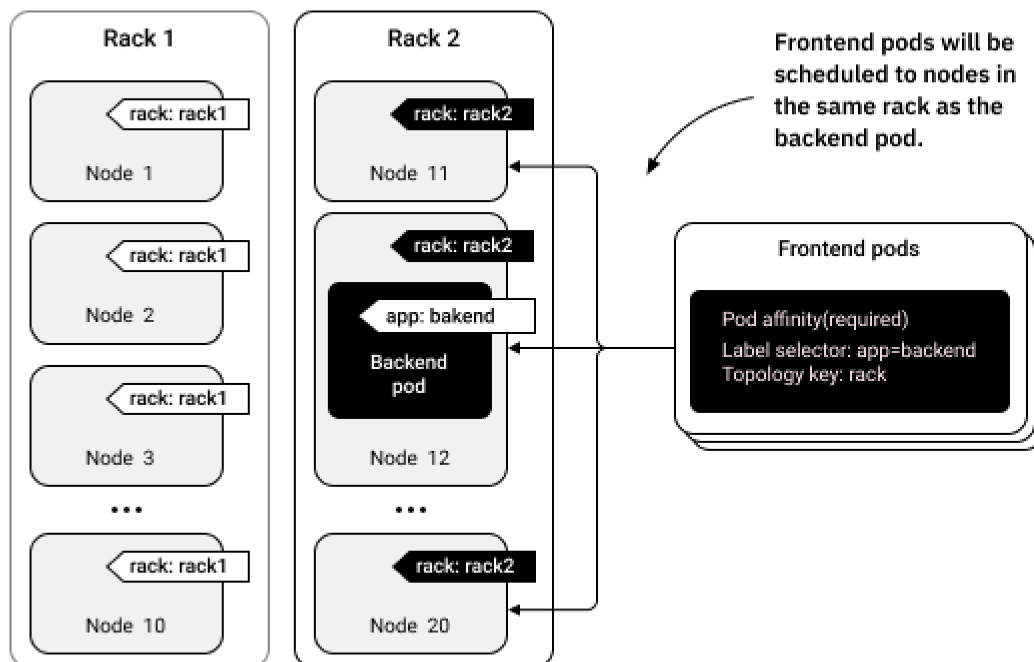
Так можно влиять не только на описание nodeSelector, но и на распределение Pod:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 5
  template:
    ...
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: backend
```

Разберемся в topologyKey. Pod попадает на узел, на котором уже есть pod с меткой app=backend. Scheduler сначала выбирает ноды с заданным topologyKey, затем фильтрует их по признаку наличия Pod с app=backend.



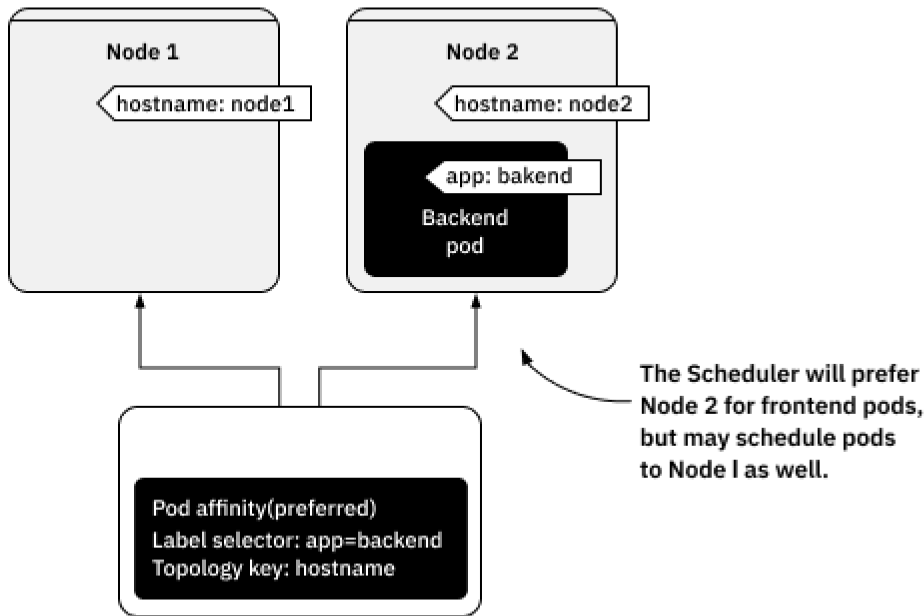
matchLabels можно заменить на matchExpression и написать более мощное выражение для выбора узла. Например, по стойкам в дата-центре.



Или прописать приоритет в выборе узла с app=backend, но позволить Scheduler запускать Pod на другом узле, при нехватке ресурсов:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
```

```
  replicas: 5
  template:
    ...
    spec:
      affinity:
        podAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 80
            podAffinityTerm:
              topologyKey: kubernetes.io/hostname
              labelSelector:
                matchLabels:
                  app: backend
      containers: ...
```

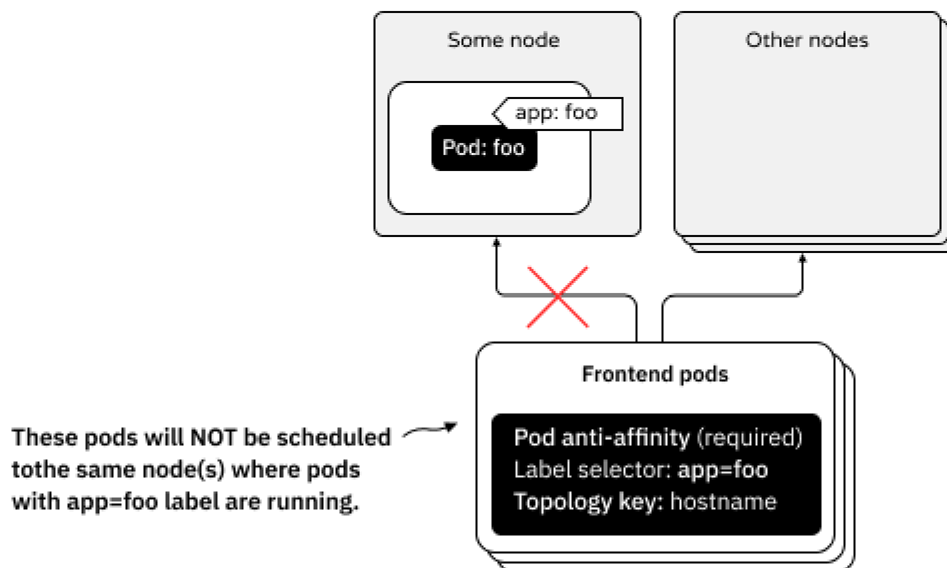


Node AntiAffinity

antiAffinity работает точно наоборот:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 5
  template:
    metadata:
```

```yaml
    labels:
    app: frontend
  spec:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - topologyKey: kubernetes.io/hostname
          labelSelector:
            matchLabels:
              app: frontend
    containers: ...
```



Данный механизм позволяет распределять поды как угодно, в том числе сделать запуск подов в единичном экземпляре на каждой ноде (хотя для этого есть другой вид ресурсов, с которым мы уже познакомились - daemonset).

Taints

nodeAffinity позволяет повысить вероятность выбора узла для запуска pod. taint работает наоборот — позволяет исключить узел из возможных для запуска. taint и tolerations работают совместно, обеспечивая исключение определенных узлов из списка подходящих для запуска. Один или несколько taints вешаются на узел, чтобы убедиться, что pod, который не допускает этих ограничений, никогда не запускался на этом узле. tolerations применяются к Pod и разрешают (но не требуют) запуск на определенном узле с определенным taint. Посмотрите на taints мастера kubectl describe node master.

taint формируется по правилу <key>=<value>:<effect>.

Пример назначенного на node taint:

node-role.kubernetes.io/master:NoSchedule

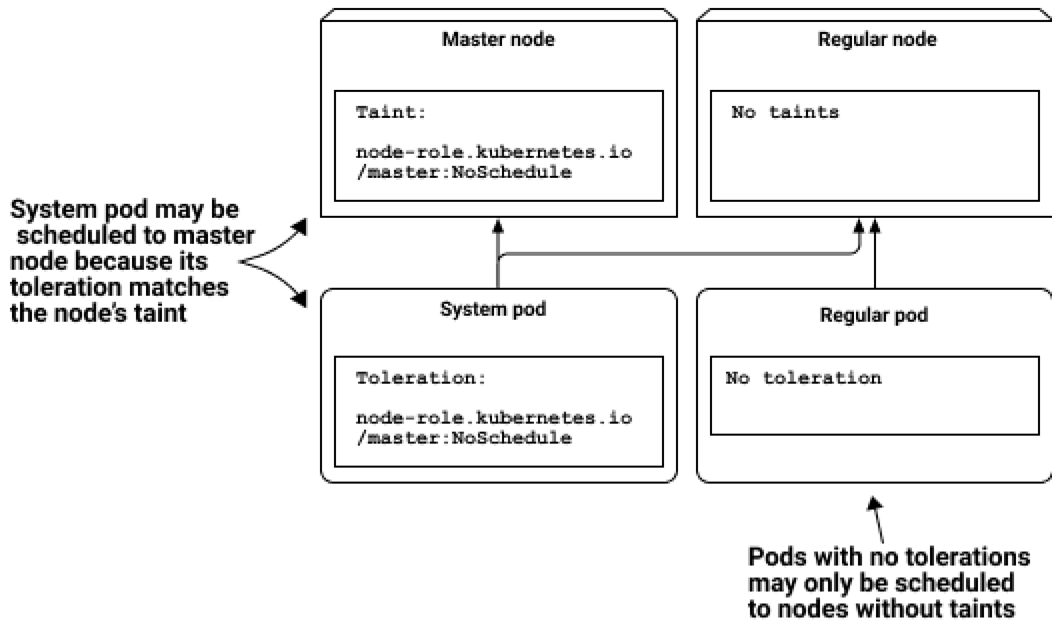
В этом примере value == null, effect == NoSchedule.

Это означает, что pod без описания tolerations для данного taint никогда не будет запущен на мастере. А это почти любой pod. Данное ограничение можно обойти:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prod
spec:
  replicas: 5
  template:
    spec:
    ...
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
```

В этом случае pod может попасть на узел мастера.
Назначение taint возможно как через kubectl edit, так и через kubectl taint. Пример:
kubectl taint node node1.k8s node-type=production:NoSchedule
А затем обходить эти ограничения для определенных pod:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prod
spec:
  replicas: 5
  template:
    spec:
    ...
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
```



Есть три значения для effect:

- NoSchedule — поды никогда не будут назначены на данный узел.
- PreferNoSchedule — scheduler попытается избежать этого узла, но, если других доступных для запуска узлов не останется, будет выбран данный.
- NoExecute — не похож на два предыдущих ограничения, влияющих только на планировщик. Если повесить это ограничение на узел, то все pod, не обходящие данное ограничение, будут перемещены с узла.

Дополнительный параметр tolerationSeconds позволяет добавить время, в течение которого pod может обходить это ограничение.

```
tolerations:
- effect: NoExecute
  key: node.alpha.kubernetes.io/notReady
  operator: Exists
  tolerationSeconds: 300
- effect: NoExecute
  key: node.alpha.kubernetes.io/unreachable
  operator: Exists
  tolerationSeconds: 300
```

В первом примере pod может 300 секунд находиться на узле с taint notReady, прежде чем будет перемещен на другой узел. Во втором примере это же правило применяется для узла с taint Unreachable.

Полезные ссылки:

- Taints and Tolerations (official docs)
- Herding pods: taints, tolerations and affinity in kubernetes
- Assigning Pods to Nodes (official docs)
- CKA Labs (14): Kubernetes Affinity and Anti-Affinity
- Assigning Pods to Nodes (official docs)

Задание:

1. Добавьте всем нодам в кластере label usercase=workload.
2. Добавьте на одну ноду taint dedicated=true noSchedule.
3. Создайте deployment dp-label, который будет запускать поды на нодах с label usercase=workload, и укажите количество реплик — 5 штук.
4. Создайте deployment dp-taint, который будет запускать поды на нодах с label usercase=workload, и toleration, который удовлетворяет условию dedicated=true.
5. Отправьте задание на проверку.