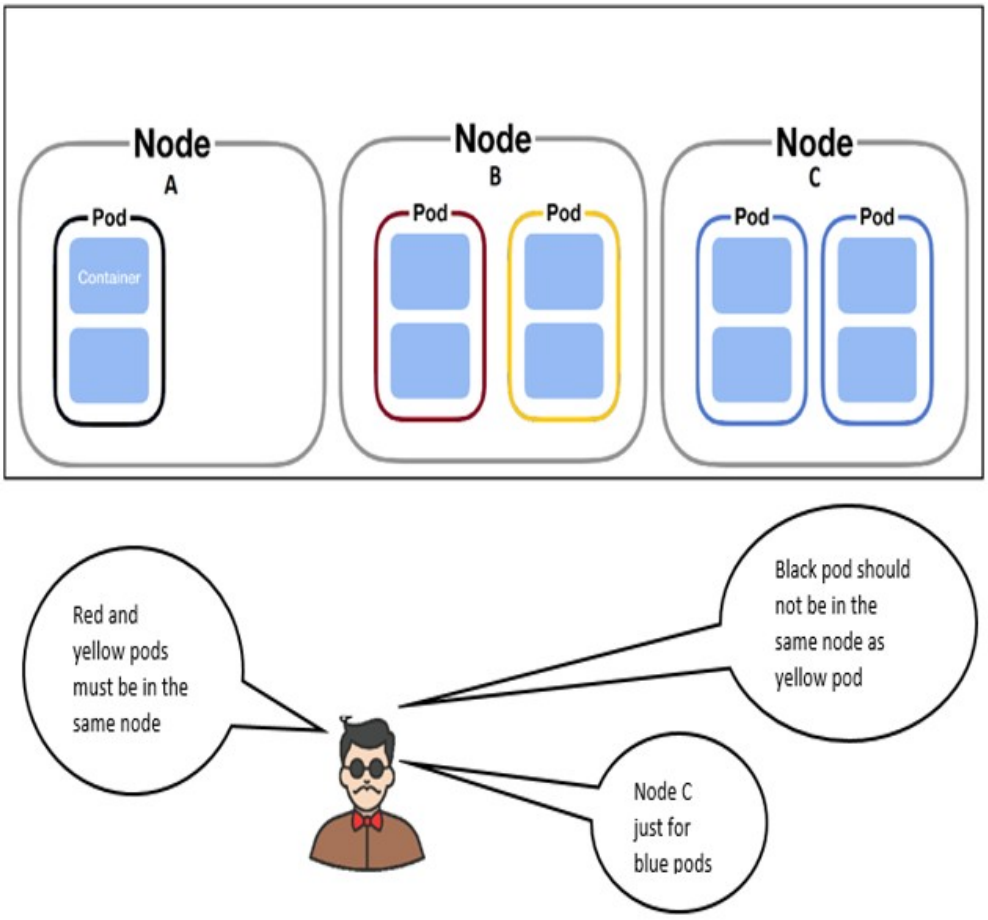


KUBERNETES: THE ART OF ADVANCED SCHDULING

MOHAMMADALI KAMALIAN

Jan 2020

Contents

مقدمه

در کوبرنتیز اصلی ترین بازیگر ما کسی نیست جز pod.از pod به عنوان واحد اجرایی یاد میشود که میتواند از یک یا چند کانتینر تشکل شده باشد. به زبان ساده تر هر آنچه سرویس یا اپلیکیشن ما نیاز دارد در دل این pod و کانتینر هایش قرار دارد.

با بزرگ شدن کلاستر ممکن است سناریو هایی برایتان پیش بیاید که محل استقرار pod ها برای شما مهم باشد و نیازمند این باشید که حتما کنترل برنامه ریزی pod را به دست بگیرید زیرا در حالت عادی kubernetes خود برای محل میزبانی pod ها تصمیم میگیرد.منظور از محل میزبانی همان نود های worker ما میباشد که pod بر روی آنها قرار میگیرد.

قبل از ورود به مباحث به مثال زیر توجه کنید:

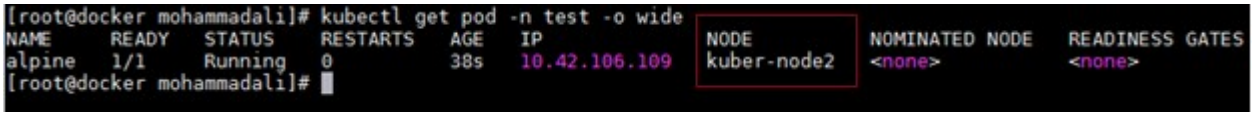
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: test
spec:
  containers:
  - name: alpine
    image: alpine
    args:
    - /bin/sh
    - -c
    - sleep 30000
  nodeSelector:
    kubernetes.io/hostname: kuber-node2
```

همانطور که مشاهده میکنید یک pod alpine درست میکنیم و با استفاده از nodeSelector و استفاده از label نود مربوطه صراحتا اعلام میکنیم که میخواهیم pod ما بر روی kuber-node2 اجرا شود. این بدان معناست که محل میزبانی pod را میخواهیم خود تعیین کنیم.

دستورات بالا را در یک فایل ذخیره میکنیم و بعد با دستور زیر آن را اجرا میکنیم:

```
[root@docker mohammadali]# kubectl apply -f alpine.yaml
pod/alpine created
[root@docker mohammadali]#
```

و حال چک میکنیم pod درخواستی ما بر روی کدام نود افتاده است:



همانطور که مشاهده میکنید Pod درخواستی ما بر روی kuber-node2 در حال کار کردن است.

در ابتدا شاید به نظر برسد که خوب همه ی خواسته های ما جواب داده شده است و خواسته ی دیگری در این زمینه نخواهیم داشت اما با بزرگ تر شدن کلاستر و بالا رفتن تعداد سرویس ها با خواسته های متفاوتی روبه رو میشویم به عنوان مثال:

- میخواهم POD A بر روی NODE A اجرا شود ولی اگر نشد بر روی نود های دیگر
- میخواهم POD A و POD B حتما هر دو بر روی NOD C اجرا شوند و کنار هم باشند
- NODE A فقط باید میزبان POD های خاصی باشد.
- به جای POD میخواهم محدودیت هایی را برای NODE تعریف کنم که از هر کسی میزبانی نکند.

تمامی این خواسته ها در کلاستر کوبرنتیز قابل پیاده سازی است.کوبرنتیز با بهره گیری از توابع بسیار پیشرفته و مختلف قدرت بسیار زیادی به ما میدهد تا Pod های خود را مدیریت کینم.از این قابلیت در خیلی از سایت ها و کتاب ها به عنوان Advance scheduling یاد میشود.

در این داکیومنت سعی داریم با مثال های متعدد به روش های مختلف در Advance scheduling بپردازیم.کلاستر کوبرنتیز استفاده شده در این داکیومنت یک کلاستر 3 نوده به شماره ورژن v.1.16.3 میباشد

Node Name	Role
Kuber-master	**Master**
Kuber-node1	**Worker**
Kuber-node2	**Worker**

Taint and Toleration

در مثالی که در مقدمه زدیم، با استفاده از Node Selector یک pod را مجبور کردیم که بر روی node خاصی بنشیند و برای این کار اطلاعاتی را در دل manifest نوشتیم و آن را اجرا کردیم (در مورد node Selector حتما بیشتر صحبت خواهیم کرد) حال میخواهیم با روشی آشنا شویم که node ها را با آن مجبور میکنیم که فقط میزبان Pod های خاصی باشند و یا به زبان ساده تر از هر کسی میزبانی نکند.

Taint در زبان انگلیسی به معنای رنگ و یا لکه میباشد که ما همان رنگ معنیش میکنیم.هر نودی بنا به شرایطی که دارد دارای taint خاص خود میباشد.به عنوان مثال taint بر روی نود های kuber-master و kuber-node1 کلاستر ما به شرح زیر است:

```
[root@docker mohammadali]# kubectl describe node kuber-master | grep Taints -A 2
Taints:             node-role.kubernetes.io/etcd=true:NoExecute
                    node-role.kubernetes.io/controlplane=true:NoSchedule
```

```
[root@docker mohammadali]# kubectl describe node kuber-node1 | grep Taints
Taints:             <none>
```

همانطور که مشاهده میکنید kuber-master ما دارای taint خاصی میباشد و در kuber-node1 هیچ taint نداریم.

حال نود kuber-node1 را خاموش میکنیم و بعد از خاموش شدن نود، taint آن را دوباره چک میکنیم:

```
[root@docker mohammadali]# kubectl describe node kuber-node1 | grep Taints -A 1
Taints:             node.kubernetes.io/unreachable:NoExecute
                    node.kubernetes.io/unreachable:NoSchedule
```

همانطور که مشاهده میکنید kuber-node1 دارای taint جدیدی شد که به به راحتی میتوان معنای آن را فهمید:در دسترس نیست.

Taint در حقیقت با رنگ آمیزی کردن نود وضعیت میزبانی از پاد ها را مشخص میکند. به سوالات زیر توجه کنیم تا کار اصلی taint را بهمیم:

- آیا نود من میتواند میزبان pod باشد؟
- آیا نود من که توانایی میزبانی را دارد باید منتظر pod خاصی باشد؟

هر نود میتواند یک یا چند taint داشته باشد.به عنوان مثال به رنگ kuber-master توجه کنیم.زمانی که ما کلاستر کوبرنتیز را نصب میکنیم در حالت پیش فرض نود یا نود های مستر دارای رنگ خاص خود هستند که این اجازه رو نمیدهند هر Pod غیر از pod های سیستمی کلاستر بر روی آن ها اجرا شوند. این بدان معناست که pod های سیستمی کوبرنتیز دارای خصوصیت های خاصی هستند که میتوانند بر روی kuber-master بنشینند که از این خصوصیت به عنوان toleration یاد میشود.

برای درک هر چه بهتر به مثال زیر توجه کنیم:

1-ابتدا رنگ خاصی بر روی kuber-node2 میزنیم تا مجوز اجرا شدن هر pod ای را بر روی kuber-node2 خود بگیریم:

```
[root@docker mohammadali]# kubectl taint node kuber-node2 node-type=production:NoSchedule
```

```
[root@docker mohammadali]# kubectl describe node kuber-node2 | grep Taints
Taints:             node-type=production:NoSchedule
[root@docker mohammadali]#
```

2-pod زیر را میسازیم و به صراحت اعلام میکنیم میخواهیم روی kuber-node2 باشد:

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: test
spec:
  containers:
  - name: alpine
    image: alpine
    args:
    - /bin/sh
    - -c
    - sleep 30000
  nodeSelector:
    kubernetes.io/hostname: kuber-node2
```

```
[root@docker mohammadali]# kubectl apply -f alpine.yaml
pod/alpine created
[root@docker mohammadali]#
```

3-pod را با دستورات زیر چک میکنیم:

```
[root@docker mohammadali]# kubectl get pods -o wide -n test
NAME     READY   STATUS    RESTARTS   AGE     IP       NODE     NOMINATED NODE   READINESS GATES
alpine   0/1     Pending   0          9m11s   <none>   <none>   <none>           <none>
[root@docker mohammadali]#
```

```
[root@docker mohammadali]# kubectl describe  pod/alpine -n test | grep -i event -A 5
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  30s (x15 over 12m)  default-scheduler  0/3 nodes are available: 1 node(s) had taints that the pod did
n't tolerate, 2 node(s) didn't match node selector.
[root@docker mohammadali]#
```

همانطور که مشاهده میکنیم pod در حالت pending مانده و خروجی دستور describe این را اعلام میکند که node درخواستی node selector دارای رنگ (taint) خاصی است که pod شما نمیتواند آنرا تحمل (tolerate) بکند یا به زبان ساده تر مجوز آن را ندارد .برای حل این مشکل باید tolerations را در پاد تعریف کنیم:

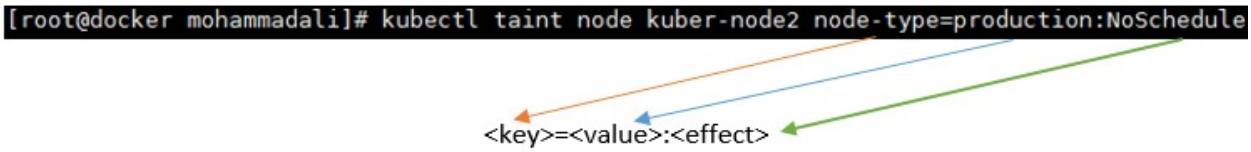
```yaml
kind: Pod
metadata:
  name: alpine
  labels:
    app: alpine
  namespace: test
spec:
  containers:
  - name: alpine
    image: alpine
    args:
    - /bin/sh
    - -c
    - sleep 30000
  nodeSelector:
    kubernetes.io/hostname: kuber-node2
  tolerations:
  - key: node-type
    operator: Equal
    value: production
    effect: NoSchedule
```

حال با ساخت این pod میتوانیم به راحتی مجوز اجرا شدن بر روی kuber-node2 که دارای taint خود است را بدهیم.

```
[root@docker mohammadali]# kubectl get pods -n test -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP             NODE          NOMINATED NODE   READINESS GATES
alpine   1/1     Running   0          29m   10.42.106.125  kuber-node2   <none>           <none>
[root@docker mohammadali]#
```

```
[root@docker mohammadali]# kubectl describe  pod/alpine -n test | grep -i toleration -A 5
Tolerations:     node-type=production:NoSchedule
                 node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason     Age   From               Message
  ----     ------     ----  ----               -------
```

همانطور که مشاهده میکنید pod ساخته شده دارای toleration مربوط به kuber-node2 هست برای همین مجوز اجرا شدن بر روی kuber-node2 را داراست. اما سوال اینجاست دستور taint که بر روی kuber-node2 زدیم به چه معناست و چه کاری انجام میدهد؟



معنای دستور بالا:هیچ پادی اجازه ی اجرا شدن بر روی kuber-node2 را ندارد مگر اینکه بتواند taint آن را تحمل کند یا به زبان ساده تر مجوز آن را که همان key value effect است را داشته باشد اما اگر قبل از زدن این رنگ pod هایی روی این node بود که مجوز نشستن بر روی این نود را نداشتند مشکلی نیست و بگذار به کارشان ادامه دهند.

Effect مهم ترین بخش کار میباشد که در حال حاضر 3 مقدار مختلف را میتواند بگیرد:

PreferNoSchedule	تلاش کن پادی که tolerate نمیکنن اینجا اجرا نشه ولی اگر انتخاب دیگه ای نداشتی مشکلی نیست(فقط برای پاد های جدید)
NoSchedule	جلوی پاد هایی که tolerate نمیکنن رو بگیر و اجازه نده روی نود بنشینن(فقط برای پاد های جدید)
NoExecute	جلوی پاد هایی که tolerate نمیکنن رو بگیر و اجازه نده روی نود بنشینن (چه پاد های جدید و پاد های که از قبل بوده اند)

NoExecute روشی سخت در قبال پاد ها در نظر میگیرد به این ترتیب که اگر kuber-node2 ما هیچ گونه taint نداشته باشد و پاد ها هم بدونه مشکل بر روی آن اجرا شوند،اگر taint جدیدی زده شود همین پاد های فعلی هم طبق مکانیزمی تحت تاثیر قرار میگرند و باید از نود بیرون کشیده شوند.در مقابل PrefeNoShedule رویکرده مهربانان تری نسبت به pod ها دارد بدین صورت که تلاش خود را برای اجازه ندادن به Pod هایی که مجوز ندارند را اعمال میکند اما اگر به هر دلیلی انتخابی دیگر جز همین نود فعلی را نداشت اجازه اجرا شدن را میدهد.

به مثال زیر توجه کنید:

kuber-node1-1 را خاموش میکنیم:

```
[root@docker mohammadali]# kubectl get nodes
NAME           STATUS     ROLES                  AGE   VERSION
kuber-master   Ready      controlplane,etcd      21d   v1.16.3
kuber-node1    NotReady   worker                 21d   v1.16.3
kuber-node2    Ready      worker                 21d   v1.16.3
```

2- taint های قبلی را از kuber-node2 پاک میکنیم و taint جدید با effect از نوع PreferNoSchedule بر روی نود میزنیم:

نکته:چون taint را قبلا پاک کردم به این error خوردم وگرنه دستور پاک کردن taint درست است.

```
[root@docker mohammadali]# kubectl taint node kuber-node2 node-type:NoSchedule-
error: taint "node-type:NoSchedule" not found
[root@docker mohammadali]# kubectl taint node kuber-node2 node-type=production:PreferNoSchedule
node/kuber-node2 tainted
[root@docker mohammadali]#
```

3-با استفاده از manifest زیر deployment alpine را با 2 replicas میسازیم.و به آن هیچ toleration نمیدهیم:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: test
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 2
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
```

```
[root@docker mohammadali]# kubectl apply -f alpine_deployment.yaml
deployment.apps/alpine created
[root@docker mohammadali]# kubectl get pods -n test -o wide
NAME                      READY   STATUS              RESTARTS   AGE   IP            NODE          NOMINATED NODE   READINESS GATES
alpine-87f7ff87c-gjss9    0/1     ContainerCreating   0          8s    <none>        kuber-node2   <none>           <none>
alpine-87f7ff87c-z7hpr    1/1     Running             0          8s    10.42.106.70  kuber-node2   <none>           <none>
[root@docker mohammadali]#
```

همانطور که میبینیم kubernetes بخاطره اینکه taint ساخته شده بر روی kuber-node2 از نوع PreferNoSchedule میباشد و هیچ گونه انتخاب دیگری جز kuber-node2 وجود ندارد اجازه ی اجرا بر روی این نود را صادر میکند.حال میتوانید 2 نوع effect دیگر را به راحتی تست کنید و رفتار کلاستر را بسنجید.

همانطور که هر نود میتواند چند taint داشته باشد، هر پاد هم میتواند چند toleration را داشته باشد.زمانی که Kubernetes با مجموعه ای از taint ها و toleration ها روبروست از مکانیزم filter کردن استفاده میکند بدین صورت که همه ی taint های نود را نظر میگیرد و اگر به ازای هر taint در نظر گرفته شده toleration آن وجود داشت آن را نادیده در نظر میگیرد در غیر این صورت بنا بر effect تعیین شده تصمیم گیری میکند.میتوانید در لینک زیر چگونگی رفتار kubernetes با مجموعه ای از taint ها و toleration ها را بخوانید.

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

چگونه toleration را بنویسیم؟

همانطور که در مثال بالا مشاهده کردیم toleration در قسمت podspec نوشته میشود و بنا به نوع taint تعریف شده برای node ،نوع تعریف toleration هم متفاوت میباشد.

یک toleration زمانی برابری با taint میکند که سه اصل را رعایت کند:

1. در هر دو طرف key ها با هم برابر باشد
2. در هر دو طرف(در صورت وجود) value ها با هم برابر باشد
3. در هر دو طرف effect ها با هم برابر باشد

در مثال های بالا toleration را برای taint ای که دارای value میباشد مورد بررسی قرار دادیم حال به سراغ taint بدون value میرویم:

```
kubectl taint node kuber-node1 node-type=:NoSchedule
```

در صورتی که بخواهیم toleration را در دل pod تعریف کنیم:

```
tolerations:
```

```
- key: node-type
  operator: Exists
  effect: NoSchedule
```

همانطور که مشاهده میکنید ، زمانی که در taint خود از value برای Key استفاده نمیکنیم برای تعریفش در Pod toleration باید از operator: Exists استفاده کنیم و هیچ گونه value دیگر تعریف نمیکنیم.در حالی که اگر taint ما value داشت، برای pod toleration باید operator: Equal تعریف میکردیم و صراحتا مقدار value را اعلام میکردیم.

رنگ هایی که به صورت اتوماتیک زده میشود

در مثال های بالا مشاهده کردیم که اگر نودی را خاموش کنیم یک یا چند taint جدید به صورت اتوماتیک بر روی node زده میشود.

```
[root@docker mohammadali]# kubectl describe node kuber-node1 | grep Taints -A 1
Taints:             node.kubernetes.io/unreachable:NoExecute
                    node.kubernetes.io/unreachable:NoSchedule
```

این نوع رنگ آمیزی ها برای نشان دادن مشکلات نود زده میشود و از آنجایی که pod toleration آن ها را به ندرت به صورت دستی تعریف میکنیم، pod نمیتواند بر روی آنها بنشیند.در لینک زیر و در اواخر صفحه لیستی از این نوع taint ها را میتوانیم مشاهده کنیم:

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

نکته بسیار جالب دیگر اینجاست که pod هایی که میسازیم حتی اگر هیچ toleration برای آنها تعریف نشده باشد به صورت خودکار چند toleration برای taint هایی از نوع بالا دارند:

```
[mohammadali@docker ~]$ kubectl describe pod/alpine -n test | grep -i tole -A 3
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason     Age   From                 Message
```

کل معنی toleration زده شده را میدانیم اما زیبایی کار همین for 300s است.اگر toleration این pod با taint نودی که روی آن نشسته برابری کند پس حتما بر روی نودی رفته است که دارای مشکل میباشد.for 300s بدین معناست که تا 300 ثانیه بر روی

همین نود بمان و اگر در این 300ثانیه taint عوض شد(یعنی مشکل حل شد) که به کارت ادامه بده وگرنه پاد ریستارت میشود و در جای دیگری میشیند.

برچسب زدن بر روی نود و استفاده از Node Selector

Label یا برچسب زدن در کلاستر کوبرنتیز روشی است برای اشاره کردن به آن موجودیت در صورت نیاز.شما میتوانید همه ی موجودیت های خود را بر چسب بزنید به عنوان مثال pod,deployment,nodes,service و غیره.در اینجا تمرکز بیشتر ما بر روی node label میباشد.

در صور اجرای دستور زیر میتوانیم بر چسب های موجود بر روی نود را پیدا کنیم

kubectl get nodes --show-labels

کوبرنتیز به صورت خودکار تعداد زیادی برچسب بر روی نود ها میزند که بسیاری از آنها میتواند جواب گوی نیاز ما باشد به این برچسب ها built-in node labels گفته میشود.در لینک زیر میتوان در مورد این برچسب ها اطلاعات بیشتر بدست آورد:

/https://kubernetes.io/docs/concepts/configuration/assign-pod-node

اما ممکن است زمانی بخواهیم بنا به دلایلی برچسب خاصی را تعریف کنیم .به عنوان مثال میخواهیم kuber-node2 را با برچسبی به عنوان special معرفی کنیم.در نظر داشته باشیم برچسب key value based میباشد.

برای برچسب زدن بر روی kuber-node2 از دستور زیر استفاده میکنیم:

```
[mohammadali@docker ~]$ kubectl label nodes kuber-node2 nodetype=special
node/kuber-node2 labeled
[mohammadali@docker ~]$
```

```
[mohammadali@docker ~]$ kubectl get nodes -l nodetype=special
NAME          STATUS   ROLES    AGE   VERSION
kuber-node2   Ready    worker   22d   v1.16.3
[mohammadali@docker ~]$
```

حال به راحتی میتوانیم در spec.containers با استفاده از nodeSelector به پاد عزیز بگوییم روی کدام نود بنشیند

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: test
spec:
  containers:
  - name: alpine
    image: alpine
    args:
    - /bin/sh
    - -c
    - sleep 30000
  nodeSelector:
    nodetype: special
```

```
[mohammadali@docker ~]$ kubectl get pods -n test -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
alpine   1/1     Running   0          44s   10.42.106.78    kuber-node2   <none>           <none>
[mohammadali@docker ~]$
```

بسیاری از ما به کلمه node affinity برخورد کردیم. این کلمه به معنای وابستگی به نود میباشد و در حقیقت node Selector هم برای همین به وجود آمد. اما Node Selector در حالی که روشی ساده برای مدیریت و اجرا pod بر روی نود خاصی است، به خواسته های متفاوت نمیتواند جواب دهد و یا میتوان اینطور گفت که اصلا انعطاف پذیری خاصی ندارد و برای همین در بسیاری از منابع به عنوان یک درخواست خشن(hard requirment) برای pod ها در نظر گرفته شده است.

در حقیق با node Selector شرایطی را بوجود می آوریم که pod ما یا همان سرویس ما فقط و فقط باید روی نود مربوطه بنشیند و در صورتی که node از بین برود عملا pod دیگر نمیتواند بر روی نود دیگری سرویس دهی انجام دهد.برای حل عدم انعطاف پذیری node Selector باید به سراغ node affinity با قوانین جالب و وسوسه انگیزش برویم.

Affinity/anti-affinity

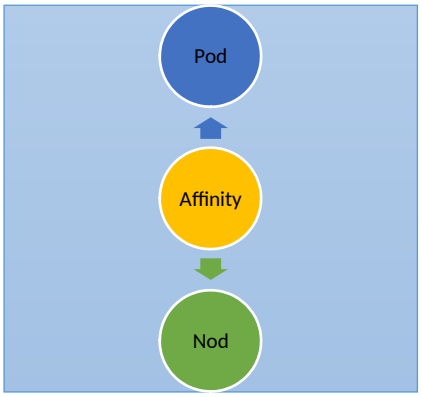
سایت رسمی کوبرنتیز 3 قدرت اصلی Affinity/anti-affinity را برای محدود سازی را اینطور بیان میکند:

1. زبان گرانی دارد،بدین معنا که شما میتوانید در زمان مقایسه کلید و مقدار برچسب ها غیر از AND از OR و operator های متفاوت استفاده کنید.
2. به غیر از درخواست خشن(hard requirement) میتوانیم از درخواست نرم و یا الویت دار(soft/preferences) تعریف کرد.

3. محدودیت ها نه تنها بر روی node label بلکه بر روی pod label هم قابل انجام است.

در ابتدا شاید این سوال در ذهن پیش بیاد که قرار بود در مورد node affinity صحبت کنیم پس affinity/anti-affinity چیست؟

خود affinity در دو سطح مختلف استفاده میشود



که در ابتدا به سراغ Node affinity میرویم و بعد از آن به Pod affinity میپردازیم.در مورد anti-affinity در حال حاضر همین را بدانیم که شاید سناریویی داشته باشیم که نخواهیم POD A در کنار POD B در یک نود کنار هم باشند برای همین باید سراغ anti-affinity برویم.

Node Affinity

Node affinity نوع پیشرفته ی node selector است. در حال حاضر در کوبرنتیز دو نوع مختلف Node Affinity وجود دارد

1-requiredDuringSchedulingIgnoredDuringExecution

2-preferredDuringSchedulingIgnoredDuringExecution

نوع اول با واژه ی required شروع میشود که به معنای ضروری است و میتوان آن را از نوع hard requirement و همتراز با همان چیزی که node selector ارایه میکند دانست.

نود دوم اما با واژه preferred شروع میشود که به معنای ارجح است و میتوان آن را از نوع soft requirement دانست چرا که این pod ها را مجبور به وابستگی کامل به نودی نمیکند در حالی که از آنها میخواهد اگر امکانش هست به نود مشخص شده وابسته باشند و اگر امکانش وجود نداشت میتوانند میزبانی Node های دیگر را بسنجند.

واژه ی DuringScheduling بدین معناست که در زمان برنامه ریزی(scheduling) برای پاد های جدید شروط را در نظر داشته باش حال IgnoredDuringExecution بدین معناست pod هایی که از قبل محل استقرار آنها مشخص شده است و در حال کار کردن هستند را مورد اعمال شروط قرار نده.

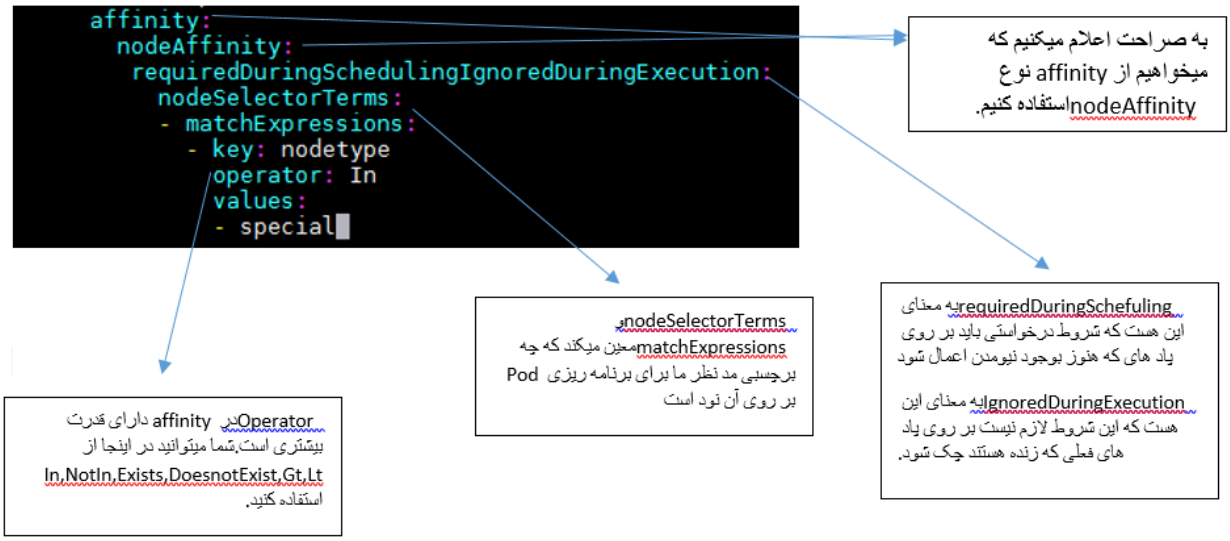
requiredDuringSchedulingIgnoredDuringExecution

به مثال زیر توجه کنیم.میخواهیم مثال alpine deployment با nodeSelector را با nodeaffinity پیاده سازی کنیم:

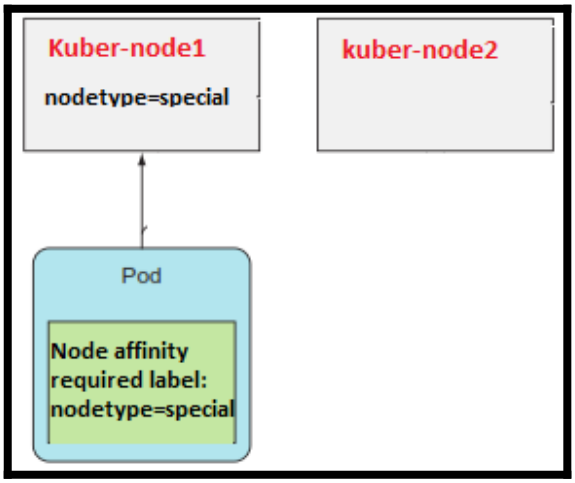
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 2
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      nodeSelector:
       nodetype: special
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 3
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: nodetype
               operator: In
               values:
               - special
```

خوب همانطور که مشاهده میکنیم affinity (سمت راست) ساختار پیچیده تری نسبت به nodeSelector (سمت چپ) دارد آن هم بخاطره این هست که انعطاف پذیری و قدرت آن در پاسخگویی نیاز های کاربر شدیدا بالاتر است.برای درک هر چه بهترخط به خط template.spec.affinity را با هم تحلیل میکنیم:



با ساخت Pod ها و استفاده از node affinity مشاهده میکنیم که همه ی pod های ما بر روی kuber-node2 خواهند رفت و این دقیقا همان خروجی زمانی است که از node Selector استفادکردیم.



حال میخواهیم با استفاده از nodeaffinity سناریویی را جلو بریم که در آن دو شرط را بررسی کنیم و اگر یکی از آنها درست بود pod ها بر روی آن نود قرار بگیرند.برای این کار ابتدا برچسب nodetype=special مربوط به kuber-node2 را پاک میکنیم و برچسب جدیدی را بر روی kuber-node1 میزنیم:

-kubectl label nodes kuber-node2 nodetype

kubectl label nodes kuber-node1 production=live

template.spec.affinity را همانند زیر تغییر میدهیم:

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: nodetype
          operator: In
          values:
          - special
      - matchExpressions:
        - key: production
          operator: In
          values:
          - live
```

```
[root@docker mohammadali]# kubectl get pods -n advanced -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP             NODE          NOMINATED NODE   READINESS GATES
alpine-d66d57f6b-45t7g   1/1     Running   0          25s   10.42.181.63   kuber-node1   <none>           <none>
alpine-d66d57f6b-5tcpv   1/1     Running   0          25s   10.42.181.32   kuber-node1   <none>           <none>
alpine-d66d57f6b-kjjfk   1/1     Running   0          25s   10.42.181.10   kuber-node1   <none>           <none>
[root@docker mohammadali]#
```

همانطور که مشاهده میکنیم pod های ما بر روی kuber-node1 در جایی که برچسب production=live وجود داشت نشسته اند یعنی یکی از شرایط nodeSelectorTerms ما راضی کننده بوده است و همین کافی میباشد.

در صورتی که ما از چند تا matchexpressions در nodeselectorterms استفاده کنیم شرایط را OR در نظر گرفتیم در حالی که اگر از چند تا key در یک matchexpressions استفاده کنیم شرایط را AND در نظر گرفته ایم.به عنوان مثال:

```
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: nodetype
              operator: In
              values:
              - special
            - key: production
              operator: In
              values:
              - live
```

چون هیچ کدام از هر دو نود دارای برچسب nodetype=special و production=live نیستند بخاطره همین pod ها به مشکل میخورند:

```
[root@docker mohammadali]# kubectl get pods -n advanced -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP       NODE     NOMINATED NODE   READINESS GATES
alpine-65765b689-fv8v6   0/1     Pending   0          73s   <none>   <none>   <none>           <none>
alpine-65765b689-gcvtj   0/1     Pending   0          73s   <none>   <none>   <none>           <none>
alpine-65765b689-lz5b5   0/1     Pending   0          73s   <none>   <none>   <none>           <none>
[root@docker mohammadali]#
```

preferredDuringSchedulingIgnoredDuringExecution

همانطور که در مثال های قسمت قبل دیدیم requiredDuringSchedulingIgnoredDuringExecution با تمام انعطاف پذیری که nodeselectorterms ها به ما میدهد همچنان به عنوان یک hard requirement میباشد بخاطره اینکه در آخر یک سری شروط باید اعمال شوند و اگر موفقیت آمیز نبودند pod بدون میزبان میماند.

اما در preferredDuringSchedulingIgnoredDuringExecution قصه کاملا متفاوت است.بدین صورت که کوبرنتیز بر اساس درخواست کاربر ارجهیت بندی میکند و تمام تلاش خود را برای یافتن میزبان برای pod ها انجام میدهد.به سراغ مثال برویم:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 2
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        nodeAffinity:
         preferredDuringSchedulingIgnoredDuringExecution:
         - weight: 90
           preference:
             matchExpressions:
             - key: nodetype
               operator: In
               values:
               - special
```

همانطور که مشاهده میکنیم در مثال بالا از preferredDuringSchedulingIgnoredDuringExecution استفاده میکنیم.یکی از بازیگران اصلی در اینجا weight میباشد.weight میزان تمایل و یا همان ارجهیت ما برای نشستن بر روی node درخواستی با برچسب مشخص شده است که مقداری بین 1 تا 100 را میگیرد.

```
[root@docker mohammadali]# kubectl apply -f alpine_deployment_nodeaffinity_singlepreferred.yaml
deployment.apps/alpine   changed
[root@docker mohammadali]# kubectl get pods -n advanced -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
alpine-7b7876d75b-fb8np   1/1     Running   0          97s   10.42.181.40    kuber-node1   <none>           <none>
alpine-7b7876d75b-lglnl   1/1     Running   0          97s   10.42.106.113   kuber-node2   <none>           <none>
[root@docker mohammadali]#
```

اما جالب اینجاست بعد از ساخت این deployment با weight 90 ،دو تا pod درخواستی بر روی دو تا نود پخش شدند در حالی که برچسب ذکر شده فقط در kuber-node1 هست.دلیل هم بسیار روشن میباشد کوبرنتیز نگران سرویس ما در قلب این pod هاست و با میخواهد حداقل اگر یک نود به هر دلیلی از بین رفت همچنان سرویس دهی داشته باشیم.حال میاییم و replicas خود را بالا میبریم تا رفتار کلاستر را ببینیم:

```
[root@docker mohammadali]# kubectl scale --replicas=5 deployment/alpine -n advanced
deployment.apps/alpine scaled
[root@docker mohammadali]#
```

```
[root@docker mohammadali]# kubectl get pods -n advanced -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP             NODE          NOMINATED NODE   READINESS GATES
alpine-7b7876d75b-fb8np   1/1     Running   0          7m57s   10.42.181.40   kuber-node1   <none>           <none>
alpine-7b7876d75b-lglnl   1/1     Running   0          7m57s   10.42.106.113  kuber-node2   <none>           <none>
alpine-7b7876d75b-mq965   1/1     Running   0          30s     10.42.181.39   kuber-node1   <none>           <none>
alpine-7b7876d75b-ps8s5   1/1     Running   0          30s     10.42.181.30   kuber-node1   <none>           <none>
alpine-7b7876d75b-rtgb9   1/1     Running   0          30s     10.42.181.36   kuber-node1   <none>           <none>
[root@docker mohammadali]#
```

خوب بسیار هم عالی..خروجی دقیقا همانطوری میباشد که فکرش را میکردیم..بخاطره weight بالایی که در نظر گرفتیم تمامی پاد ها به غیر از یکی بر روی kuber-node1 قرار گرفته است.در حقیقت ارجهیت را kuber-node1 با weight بالا در نظر گرفته است.

حال میخواهیم kuber-master که همانطور از اسمش پیداست را هم وارد بازی کنیم و از او به عنوان worker هم کار بکشیم تا یک مثال preferredDuringSchedulingIgnoredDuringExecution با 3 نود را با هم مرور کنیم.

برای این کار ابتدا باید taint های مربوط به kuber-master را برداریم:

```
[mohammadali@docker ~]$ kubectl describe node kuber-master | grep -i taint -A 3
Taints:             node-role.kubernetes.io/etcd=true:NoExecute
                    node-role.kubernetes.io/controlplane=true:NoSchedule
Unschedulable:      false
Conditions:
[mohammadali@docker ~]$
```

برای این کار از دستورات زیر استفاده میکنیم:

```
[mohammadali@docker ~]$ kubectl taint node kuber-master node-role.kubernetes.io/etcd:NoExecute-
node/kuber-master untainted
[mohammadali@docker ~]$ kubectl taint node kuber-master node-role.kubernetes.io/controlplane:NoSchedule-
node/kuber-master untainted
[mohammadali@docker ~]$
```

Deployment مربوط به alpine را با 12 replicas و با 3 تا weight مختلف برای هر نود به صورت زیر آماده میکنیم:

```yaml
    affinity:
      nodeAffinity:
       preferredDuringSchedulingIgnoredDuringExecution:
       - weight: 20
         preference:
           matchExpressions:
           - key: kubernetes.io/hostname
             operator: In
             values:
             - kuber-master
       - weight: 40
         preference:
           matchExpressions:
           - key: kubernetes.io/hostname
             operator: In
             values:
             - kuber-node1
       - weight: 40
         preference:
           matchExpressions:
           - key: kubernetes.io/hostname
             operator: In
             values:
             - kuber-node2
```

خروجی که به دست می آید بسیار جالب است:

```
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                     READY   STATUS    RESTARTS   AGE     IP              NODE           NOMINATED NODE   READINESS GATES
alpine-7b76ff595f-8gf4v  1/1     Running   0          13m     10.42.181.50    kuber-node1    <none>           <none>
alpine-7b76ff595f-cwk7n  1/1     Running   0          13m     10.42.106.95    kuber-node2    <none>           <none>
alpine-7b76ff595f-d6gwt  1/1     Running   0          13m     10.42.56.201    kuber-master   <none>           <none>
alpine-7b76ff595f-gn6dr  1/1     Running   0          12m     10.42.56.202    kuber-master   <none>           <none>
alpine-7b76ff595f-h6t7h  1/1     Running   0          13m     10.42.106.124   kuber-node2    <none>           <none>
alpine-7b76ff595f-hgj9m  1/1     Running   0          13m     10.42.56.200    kuber-master   <none>           <none>
alpine-7b76ff595f-j9mm5  1/1     Running   0          13m     10.42.181.51    kuber-node1    <none>           <none>
alpine-7b76ff595f-kkwnm  1/1     Running   0          7m29s   10.42.106.75    kuber-node2    <none>           <none>
alpine-7b76ff595f-r9lnd  1/1     Running   0          12m     10.42.106.73    kuber-node2    <none>           <none>
alpine-7b76ff595f-tcg6r  1/1     Running   0          16s     10.42.181.43    kuber-node1    <none>           <none>
alpine-7b76ff595f-xhx5m  1/1     Running   0          7m29s   10.42.106.123   kuber-node2    <none>           <none>
alpine-7b76ff595f-zjjb6  1/1     Running   0          12m     10.42.181.46    kuber-node1    <none>           <none>
```

Node Name	Wight	Number of Pod
Kuber-master	20	3
Kuber-node1	40	4
Kuber-node2	40	5

همانطور که مشاهده میکنید تعداد pod هایی که بر روی kuber-node1 و kuber-node2 نشسته اند بیشتر از Kuber-master میباشد و دلیلش weight بالاتری هست که دارند.

Pod Affinity

وابستگی بین pod و نود را با استفاده از node Affinity مورد بررسی قرار دادیم حال با استفاده از pod Affinity به سراغ چگونگی مطرح کردن وابستگی بین pod ها میرویم.فرض کنید با سناریویی مواجه هستید که برایتان بسیار مهم است که pod هایی که میسازید حتما در کنار هم بر روی یک نود قرار گیرند.

برای درک هر چه بهترچگونگی استفاده از pod Affinity مثال زیر را با هم مرور میکنیم:

1-ابتدا یک busybox میسازیم و با node Selector مجبورش میکنیم که حتما بر روی kuber-node2 بنشیند:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  labels:
   app: busybox
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: busybox
  replicas: 2
  template:
    metadata:
     labels:
      app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox
        args:
        - /bin/sh
        - -c
        - sleep 30000
      nodeSelector:
       kubernetes.io/hostname: kuber-node2
```

```
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                       READY   STATUS    RESTARTS   AGE     IP              NODE          NOMINATED NODE   READINESS GATES
busybox-5fb44875d9-g8xmq   1/1     Running   0          3m32s   10.42.106.84    kuber-node2   <none>           <none>
busybox-5fb44875d9-gqzrj   1/1     Running   0          3m32s   10.42.106.100   kuber-node2   <none>           <none>
[mohammadali@docker ~]$
```

2-حال میخواهیم alpine خود را طوری مدیریت کنیم که حتما در کنار busybox بر روی kuber-node2 بنشیند:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 3
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        podAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
         - topologyKey: kubernetes.io/hostname
           labelSelector:
            matchLabels:
             app: busybox
```

```
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                       READY   STATUS    RESTARTS   AGE    IP              NODE          NOMINATED NODE   READINESS GATES
alpine-6d9fc69b7d-67f2l    1/1     Running   0          100s   10.42.106.85    kuber-node2   <none>           <none>
alpine-6d9fc69b7d-f4mw7    1/1     Running   0          100s   10.42.106.83    kuber-node2   <none>           <none>
alpine-6d9fc69b7d-fgs4k    1/1     Running   0          100s   10.42.106.101   kuber-node2   <none>           <none>
busybox-5fb44875d9-g8xmq   1/1     Running   0          8m7s   10.42.106.84    kuber-node2   <none>           <none>
busybox-5fb44875d9-gqzrj   1/1     Running   0          8m7s   10.42.106.100   kuber-node2   <none>           <none>
[mohammadali@docker ~]$
```

همانطور که مشاهده میکنید همه ی pod های مربوط به busybox و alpine در کنار هم بر روی kuber-node2 قرار گرفته اند.

قبل از اینکه به تحلیل ساختار manifest مربوط به alpine خود در قسمت podAffinity بپردازیم چند نکته را با هم مرور میکنیم:

1. نکته ی بسیار مهمی که تفاوت را بین podAffinity و nodeAffinity رقم می زند label هست.ما در nodeAffinity با استفاده از nodeLabel وابستگی بین pod و نود را مطرح میکردیم حال در PodAffinity با استفاده از podlabel وابستگی بین pod ها را رقم میزنیم.
2. topologyKey یکی از کلید واژه های مهم در pod Affinity میباشد.topology در زبان انگلیسی به معنای مکان شناسی و وضعیت جغرافیایی میباشد

و اما تحلیل ماجرا:ما در ابتدا یک سری busybox ساختیم و آنها را مجبور کردیم که بر روی kuber-node2 بنشینند.حال برای اینکه alpine ما کنار busybox ها بالا بیایید باید نکاتی را برای podAffinity مشخص کنیم.ابتدا باید بگوییم که مقیاس و مکان یابی ما در چه سطحی میباشد.بعد از معین کردن سطح مکانیابی برای pod ها باید مشخص کنیم مکان مورد نظر ما باید دارای چه شرایطی باشد.

در مثالی که با هم انجام دادیم سطح مکان یابی مورد نظر ما نود میباشد (topologyKey: kubernetes.io/hostname)و شروطی که باید نود ما داشته باشد این است که یک سری pod با برچسب app: busybox بر روی آن نشسته باشد.از آنجایی که نود kuber-node2 شروط مورد نظر ما را دارد alpine هم بر روی همین نود مینشیند.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  labels:
    app: busybox
  namespace: advanced
spec:
  selector:
    matchLabels:
      app: busybox
  replicas: 2
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox
        args:
        - /bin/sh
        - -c
        - sleep 30000
      nodeSelector:
        kubernetes.io/hostname: kuber-node2
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
    app: alpine
  namespace: advanced
spec:
  selector:
    matchLabels:
      app: alpine
  replicas: 3
  template:
    metadata:
      labels:
        app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: busybox
```

از آنجایی که pod ها را میتوان در namespace های مختلف قرار داد،میتوان در کنار label از namespace هم در podaffinity استفاده کرد:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 3
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        podAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
         - topologyKey: kubernetes.io/hostname
           labelSelector:
            matchLabels:
             app: busybox
           namespaces: ["default"]
```

در اینجا ما برای Pod های خود به دنبال نودی هستیم که pod هایی با برچسب app: busybox در namespaces default بر روی آن نشسته باشد.میتوانید این را تست کنید و نتیجه ی آن را ببینید.

Pod anti-affinity

مثال هایی که برای pod affinity را با هم مرور کردیم همگی یک هدف مشترک را دنبال میکردند:میخواهیم pod های مورد نظر در کنار هم باشند.اما در pod anti affinity قضیه کاملا برعکس است و میخواهیم pod های مورد نظر ما در کنار هم نباشند.

از pod anti affinity میتوان در سناریو هایی استفاده کرد که اجرا شدن pod های موردنظر بر روی یک نود ممکن است باعث به وجود آمدن مشکلات performance بشود یا زمانی که می میخواهیم مطمین شویم نسخه های pod های مورد نظر ما حتما بر روی نود های مختلف پخش شده اند تا در صورت به مشکل خوردن یک نود همچنان سرویس ما بالا باشد یا در مقیاس بزرگ تر pod های ما بر روی rack های مختلف و یا zone های مختلف پخش شده باشند و فقط در یک محل نباشند.

در مثال زیر میخواهیم دو نسخه از pod alpine بالا بیاوریم به طوری که هیچکدام از pod ها که دارای برچسب app: alpine هستند بر روی یک نود در کنار هم نباشند:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alpine
  labels:
   app: alpine
  namespace: advanced
spec:
  selector:
   matchLabels:
    app: alpine
  replicas: 2
  template:
    metadata:
     labels:
      app: alpine
    spec:
      containers:
      - name: alpine
        image: alpine
        args:
        - /bin/sh
        - -c
        - sleep 30000
      affinity:
        podAntiAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
         - topologyKey: kubernetes.io/hostname
           labelSelector:
            matchLabels:
             app: alpine
```

بعد از ساخت deployment بالا وضعیت pod ها را چک میکنیم:

```
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
alpine-794d9bfdb9-786xx   1/1     Running   0          12s   10.42.181.50    kuber-node1   <none>           <none>
alpine-794d9bfdb9-mf5vh   1/1     Running   0          12s   10.42.56.194    kuber-master  <none>           <none>
[mohammadali@docker ~]$
```

همانطور که مشاهده میکنیم دو نسخه ی مربوط به alpine در کنار هم نیستند.حال برای اطمینان از اینکه podAntiAffinity واقعا کار میکند تعداد نسخه های مربوط به alpine را به 4 افزایش میدهیم و بعد وضعیت pod ها را چک میکنیم:

```
[mohammadali@docker ~]$ kubectl scale --replicas=4 deployment/alpine -n advanced
deployment.apps/alpine scaled
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP              NODE          NOMINATED NODE   READINESS GATES
alpine-794d9bfdb9-786xx   1/1     Running   0          3m23s   10.42.181.50    kuber-node1   <none>           <none>
alpine-794d9bfdb9-7qcnz   1/1     Running   0          16s     10.42.106.116   kuber-node2   <none>           <none>
alpine-794d9bfdb9-djvm2   0/1     Pending   0          16s     <none>          <none>        <none>           <none>
alpine-794d9bfdb9-mf5vh   1/1     Running   0          3m23s   10.42.56.194    kuber-master  <none>           <none>
[mohammadali@docker ~]$
```

```
[mohammadali@docker ~]$ kubectl describe pod/alpine-794d9bfdb9-djvm2 -n advanced | grep -i Events -A 10
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  74s (x4 over 3m51s)  default-scheduler  0/3 nodes are available: 3 node(s) didn't match pod affinity/anti-affinity, 3 node(s) didn't satisfy existing pods anti-affinity rules.
[mohammadali@docker ~]$
```

بعد از اضافه شدن دو نسخه دیگر عملا هر 3 نود ما یک نسخه از pod alpine را بر روی خود دارد و از آنجا که ما فقط میخواهیم بر روی هر نود فقط یک نسخه باشد، یکی از pod ها در حالت pending می ماند چون هیچ نودی نمیتواند او را قبول کند.

به سراغ یک مثال دیگر میرویم که در آن deployment هایی را با خاصیت های زیر درست میکنیم:

alpine deployment-1 با 2 replicas و pod anti affinity به طوری که pod بر روی نودی بنشیند که pod با برچسب app: alpine وجود نداشته باشد.

busybox deployment-2 با 1 replicas و pod anti affinity به طوری که pod بر روی نودی بنشیند که pod با برچسب app: alpine وجود نداشته باشد.

هر دو deployment ما دارای ساختار pod anti affinity زیر خواهند بود.از مثال های قبل هم میتوان برای ساخت manifest این مثال کمک گرفت.

```
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: alpine
```

بعد از ساخت deployment های مدنظر موقعیت pod ها را چک میکنیم:

```
[mohammadali@docker ~]$ kubectl apply -f 1.yaml
deployment.apps/alpine created
deployment.apps/busybox created
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
alpine-794d9bfdb9-6xtrr    1/1     Running   0          17m   10.42.56.203    kuber-master  <none>           <none>
alpine-794d9bfdb9-lzm8p    1/1     Running   0          17m   10.42.181.21    kuber-node1   <none>           <none>
busybox-7645765484-gwkgz   1/1     Running   0          17m   10.42.106.106   kuber-node2   <none>           <none>
```

حال میخواهیم برای آزمایش کردن رفتار kubernetes تعداد نسخه های alpine deployment را افزایش دهیم و بعد وضعیت pod ها را چک کنیم:

```
[mohammadali@docker ~]$ kubectl scale --replicas=3 deployment/alpine -n advanced
deployment.apps/alpine scaled
[mohammadali@docker ~]$ kubectl get pods -n advanced -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
alpine-794d9bfdb9-66kl2    0/1     Pending   0          3s    <none>          <none>        <none>           <none>
alpine-794d9bfdb9-6xtrr    1/1     Running   0          27m   10.42.56.203    kuber-master  <none>           <none>
alpine-794d9bfdb9-lzm8p    1/1     Running   0          27m   10.42.181.21    kuber-node1   <none>           <none>
busybox-7645765484-gwkgz   1/1     Running   0          27m   10.42.106.106   kuber-node2   <none>           <none>
[mohammadali@docker ~]$
```

Pod جدید در حالت Pending باقی مانده است دلیل این است که هیچ نودی وجود ندارد که شروط مربوط به pod anti affinity را قبول کند.

در pod anti affinity هم میتوانید از preferredDuringSchedulingIgnoredDuringExecution برای بیان درخواست استفاده کرد تا به نوعی hard requirement خود را به soft requirement تبدیل کنیم.به ساختار زیر توجه کنید و بعد برای خود مثال هایی را انجام دهید:

```
affinity:
  podAntiAffinity:
   preferredDuringSchedulingIgnoredDuringExecution:
   - weight: 50
     podAffinityTerm:
      labelSelector:
       matchExpressions:
       - key: app
         operator: In
         values:
         - alpine
       topologyKey: kubernetes.io/hostname
```

نتیجه گیری و پایان

بسیاری از سازمان ها ممکن است در ابتدا نیازی برای ورود به مباحث Advanced scheduling نداشته باشند و دلیلش هم این است که تعداد سرویس ها کم و تعداد کلاستر همچنان در مقیاس کوچکی قرار دارد.اما با گذشت زمان و بزرگ تر شدن تعداد سرویس ها و بالا رفتن وابستگی ها بین سرویس ها و همچنین زیاد شدن تعداد نود ها ورود به این نوع مباحث امری اجتناب ناپذیر است.در این داکیومنت سعی بر آن شد که نگاهی سطحی به مباحث Advanced scheduling داشته باشم و با یک سری مثال ها مفاهیم را بهتر مرور کنیم.در پایان امیدوارم این داکیومنت مورد قبول شما دوستان قرار گرفته باشد و پیشاپیش بابت هرگونه اشتباه چه در سطح نوشتاری،تعاریف مفاهیم و پیش بردن مثال ها عذر خواهی بنده را بپذیرید.

Contact me on Linkedin: https://www.linkedin.com/in/mohammad-ali-kamalian-7a3a72124/

Resource

/https://kubernetes.io/docs/concepts/configuration/assign-pod-node

/https://kubernetes.io/docs/concepts/overview/working-with-objects/labels

https://docs.okd.io/latest/admin_guide/scheduling/node_selector.html

https://medium.com/kubernetes-tutorials/learn-how-to-assign-pods-to-nodes-in-kubernetes-using-nodeselector-and-affinity-features-e62c437f3cf8

/https://banzaicloud.com/blog/k8s-affinities

Marko Luksa “Kubernetes in action” pdf