Kubernetes Label Selector And Field Selector

Ref: https://medium.com/mayadata/kubernetes-label-selector-and-field-selector-81d6967bb2f

The resources that we create in a kubernetes cluster can be organised by using labels. Before we talk about field selector in Kubernetes, let us walk through quickly about labels.

Labels are key value pairs that can be used to identify, or group the resources in Kubernetes. In other words, labels can be used to select resources from a list.
You can label Kubernetes native resources as well as Custom Resources. To understand it more clearly, let us do some hands on practice on labels.

The tutorial will assume that you have a working minikube setup or a Kubernetes cluster setup.

Following is link to the yaml . It's application will create a pod.
https://raw.githubusercontent.com/sonasingh46/artifacts/master/samples/sample-pod.yaml

The yaml looks following:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    env: development
spec:
  containers:
  - name: label-example
    image: sonasingh46/node-web-app:latest
    ports:
    - containerPort: 8000
```

Notice the bold text in above yaml. That is one way to add labels to a resource by specifying in yaml. Let us create a pod by executing following command:
kubectl apply -f https://raw.githubusercontent.com/sonasingh46/artifacts/master/samples/sample-pod.yaml

You can use above command directly or copy the content to save it on your local machine in a file , say `sample-pod.yaml`.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl apply -f sample-pod.yaml
pod/example-pod created
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get po
NAME READY STATUS RESTARTS AGE
example-pod 1/1 Running 0 3m
```

Now, let us run the following commands to check for labels in the pod.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod example-pod --show-labels
NAME                          READY     STATUS    RESTARTS   AGE     LABELS
example-pod                   1/1       Running   0          3m        env=development
```

As you can see in the above output `example-pod` is having a label of key value pair as `env=development` .
You can also do a `kubectl get pod example-pod -o yaml` to see all the fields along with labels.
Let us add another label to the above pod using `kubectl` command.

Adding a label:

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl label pod example-pod tier=backend
pod/example-pod labeled
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod example-pod --show-labels
NAME           READY     STATUS    RESTARTS   AGE       LABELS
example-pod    1/1       Running   0          13m       env=development,tier=backend
```

Removing a label:

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl label pod example-pod tier-
pod/example-pod labeled
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod example-pod --show-labels
NAME           READY     STATUS    RESTARTS   AGE       LABELS
example-pod    1/1       Running   0          23m       env=development
```

Updating label:

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl label --overwrite pods example-pod env=prod
pod/example-pod labeled
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod example-pod --show-labels
NAME          READY     STATUS    RESTARTS   AGE       LABELS
example-pod   1/1       Running   0          25m       env=prod
```

`kubectl label --overwrite pods example-pod env=prod` command will update the value of key `env` in the labels and if the label does not exist, it will create one.
Lets create one more pod by editing the above yaml and changing `metadata.name` to example-pod1.
Also we will remove the label from yaml.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod1
spec:
  containers:
  - name: label-example
    image: sonasingh46/node-web-app:latest
    ports:
    - containerPort: 8000
```

Create a yaml file with above content , lets say `sample-pod1.yaml` and apply it.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl apply -f sample-pod1.yaml
pod/example-pod1 created
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod
NAME                        READY     STATUS    RESTARTS   AGE
example-pod                 1/1       Running   0          17h
example-pod1                1/1       Running   0          6s
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods --show-labels
NAME                        READY     STATUS    RESTARTS   AGE       LABELS
example-pod                 1/1       Running   0          17h       env=prod
example-pod1                1/1       Running   0          1m        <none>
```

You can learn about few more kubectl label commands using `kubectl label --help`

Now we are good enough to tag our resources with labels either via providing it in yaml or using kubectl command. Let us now explore how the label can help in filtering or grouping the resources.

Selection Via Labels(Label Selector)

Selection via labels can have following two types of requirements:

1. Equality Based Requirement
2. Set Based Requirement

Equality Based Requirement

Equality based requirement will match for the specified label and filter the resources. The supported operators are `=`, `==`, `!=` .

Let us say I have following pods with the labels.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get po --show-labels
NAME                        READY     STATUS    RESTARTS   AGE       LABELS
example-pod                 1/1       Running   0          17h
env=prod,owner=Ashutosh,status=online,tier=backend
example-pod1                1/1       Running   0          21m
env=prod,owner=Shovan,status=offline,tier=frontend
example-pod2                1/1       Running   0          8m
env=dev,owner=Abhishek,status=online,tier=backend
example-pod3                1/1       Running   0          7m
env=dev,owner=Abhishek,status=online,tier=frontend
```

Now, I want to see all the pods with online status:

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l status=online
NAME             READY     STATUS    RESTARTS   AGE
example-pod      1/1       Running   0          17h
example-pod2     1/1       Running   0          9m
example-pod3     1/1       Running   0          9m
```

Similarly, go through following commands

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l status!=online
NAME                         READY     STATUS     RESTARTS   AGE
example-pod1                 1/1       Running    0          25m
example-pod4                 1/1       Running    0          11m

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l status==offline
NAME            READY     STATUS     RESTARTS   AGE
example-pod1    1/1       Running    0          26m
example-pod4    1/1       Running    0          11m

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l status==offline,status=online
No resources found.

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l status==offline,env=prod
NAME            READY     STATUS     RESTARTS   AGE
example-pod1    1/1       Running    0          28m

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods  -l owner=Abhishek
NAME            READY     STATUS     RESTARTS   AGE
example-pod2    1/1       Running    0          15m
example-pod3    1/1       Running    0          14m
```

In above commands, labels separated by comma is a kind of **AND** satisfy operation.
Similarly, you can try other combination using the operators (`=` , `!=`, `==`)and play!

Set Based Requirement

Label selectors also support set based requirements. In other words, label selectors can be use to specify a set of resources.
The supported operators here are `in` , `notin` and `exists` .

Let us walk through kubectl commands for filtering resources using set based requirements.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod -l 'env in (prod)'
NAME            READY     STATUS    RESTARTS   AGE
example-pod     1/1       Running   0          18h
example-pod1    1/1       Running   0          41m

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod -l 'env in (prod,dev)'
NAME            READY     STATUS    RESTARTS   AGE
example-pod     1/1       Running   0          18h
example-pod1    1/1       Running   0          41m
example-pod2    1/1       Running   0          27m
example-pod3    1/1       Running   0          27m
```

Here `env in (prod,dev)` the comma operator acts as a **OR** operator. That is it will list pods which are in `prod` or `dev`.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod -l 'env in (prod),tier in
(backend)'
NAME           READY     STATUS    RESTARTS   AGE
example-pod    1/1       Running   0          18h

ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod -l 'env in (qa),tier in
(frontend)'
No resources found.
```

Here the comma operator separating `env in (qa)` and `tier in (frontend)` will act as an AND operator. To understand the `exists` operator let us add label `region=central` to `example-pod` and `example-pod1` and `region=northern` to `example-pod2` .

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get  pod --show-labels
NAME                         READY     STATUS    RESTARTS   AGE       LABELS
example-pod                  1/1       Running   0          18h
env=prod,owner=Ashutosh,region=central,status=online,tier=backend
example-pod1                 1/1       Running   0          54m
env=prod,owner=Shovan,region=central,status=offline,tier=frontend
example-pod2                 1/1       Running   0          40m
env=dev,owner=Abhishek,region=northern,status=online,tier=backend
example-pod3                 1/1       Running   0          40m
env=dev,owner=Abhishek,status=online,tier=frontend
example-pod4                 1/1       Running   0          40m
env=qa,owner=Atul,status=offline,tier=backend
```

Now, I want to view pods that is not in central region:

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods -l 'region notin (central)'
NAME                      READY     STATUS     RESTARTS    AGE
example-pod2              1/1       Running    0           42m
example-pod3              1/1       Running    0           42m
example-pod4              1/1       Running    0           41m
```

You can realise here that `example-pod2` is having a `region` key with value `northern` and hence appears in result. But one point to note is that other two pods in result is not having any region field and will satisfy the condition to appear in result.

If we want that pods having `region` key should only be the set of resources over which filtering should be done we can restrict via the `exists`operator.We do not specifically write `exists` like we do write `in` and `notin`in command.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pods -l 'region,region notin
(central)'
NAME           READY     STATUS     RESTARTS    AGE
example-pod2   1/1       Running    0           46m
```

milarly, you can play by using various combinations in set based requirements too for selecting a set of pods.

For more information about Labels and Selectors you can visit
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selection Via Fields(Field Selector)

We can also select kubernetes resources via field selector but it has very limited support as of now.

Field selector do not support set based requirement. Even the support for equality based requirement is not that extensible.

There are only limited fields that can be used for selection.

```
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod --field-selector
metadata.name=example-pod
NAME           READY     STATUS    RESTARTS   AGE
example-pod    1/1       Running   0          18h
ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod --field-selector
metadata.namespace=default
NAME                      READY     STATUS    RESTARTS   AGE
example-pod               1/1       Running   0          18h
example-pod1              1/1       Running   0          1h
example-pod2              1/1       Running   0          1h
example-pod3              1/1       Running   0          1h
example-pod4              1/1       Running   0          1h


ashutosh@miracle:~/Desktop/artifacts/samples$ kubectl get pod --field-selector spec.name=label-
example
No resources found.
```

```
Error from server (BadRequest): Unable to find {"" "v1" "pods"} that match label selector "",
field selector "spec.name=label-example": field label not supported: spec.name
```

So one can conclude that, field-selector only works for `metadata.name`
and for additional fields for some types, but it is a very select set. For example, visit the link to see fields supported on pods:
https://github.com/kubernetes/kubernetes/blob/master/pkg/apis/core/v1/conversion.go#L160-L167