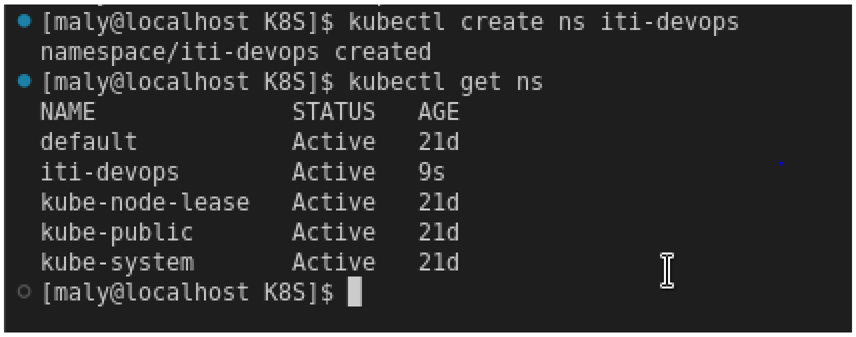
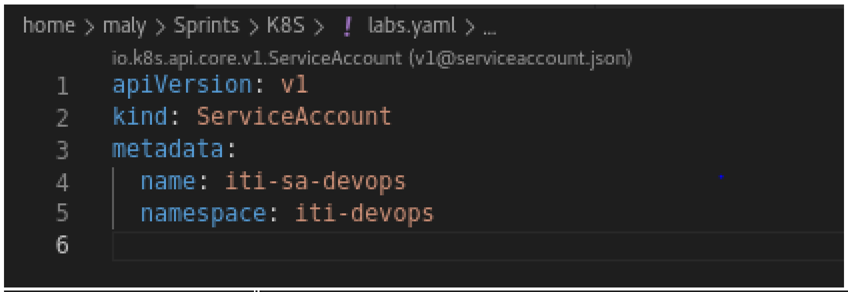
LAB5

1- create a namespace iti-devops



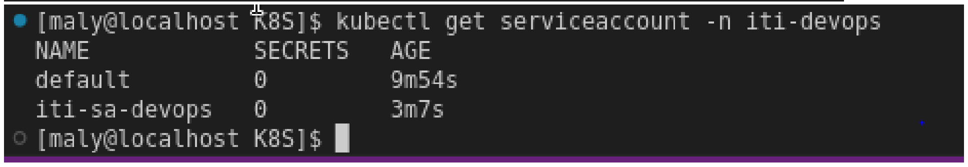
```
[maly@localhost K8S]$ kubectl create ns iti-devops
namespace/iti-devops created
[maly@localhost K8S]$ kubectl get ns
NAME              STATUS   AGE
default           Active   21d
iti-devops        Active   9s
kube-node-lease   Active   21d
kube-public       Active   21d
kube-system       Active   21d
[maly@localhost K8S]$
```

2- create a service account iti-sa-devops under the same namespace



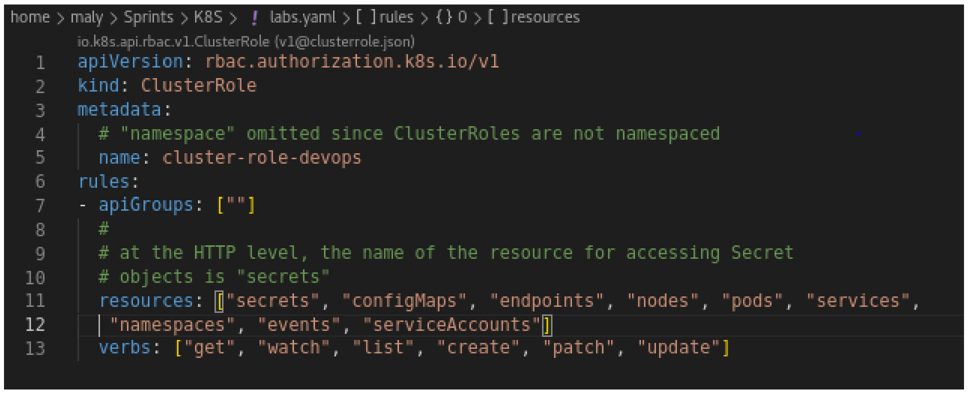
home > maly > Sprints > K8S > ! labs.yaml > ...
io.k8s.api.core.v1.ServiceAccount (v1@serviceaccount.json)

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: iti-sa-devops
  namespace: iti-devops
```



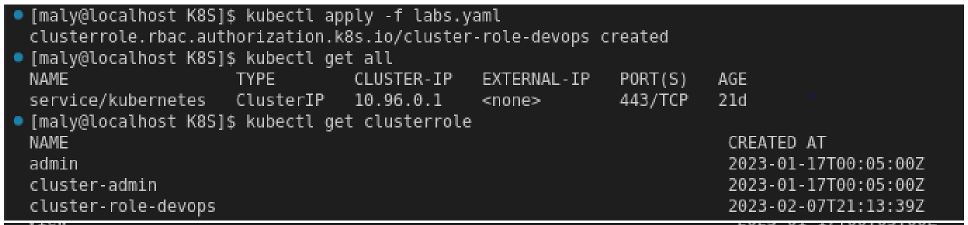
```
[maly@localhost K8S]$ kubectl get serviceaccount -n iti-devops
NAME            SECRETS   AGE
default         0         9m54s
iti-sa-devops   0         3m7s
[maly@localhost K8S]$
```

3- create a clusteRole which should be named as cluster-role-devops to grant permissions "get","list","watch","create","patch","update" to "configMaps"," secrets"," endpoints","nodes","pods"," services"," namespaces"," events","serviceAccounts".

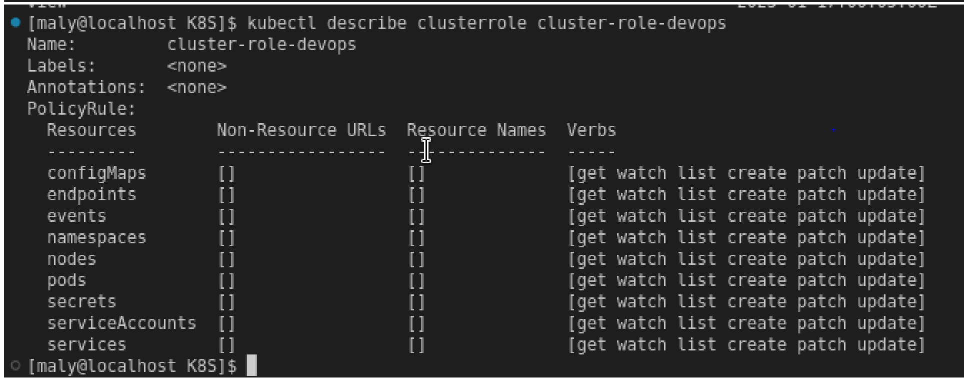


home > maly > Sprints > K8S > ! labs.yaml > [ ] rules > { } 0 > [ ] resources
io.k8s.api.rbac.v1.ClusterRole (v1@clusterrole.json)

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: cluster-role-devops
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets", "configMaps", "endpoints", "nodes", "pods", "services",
   "namespaces", "events", "serviceAccounts"]
  verbs: ["get", "watch", "list", "create", "patch", "update"]
```

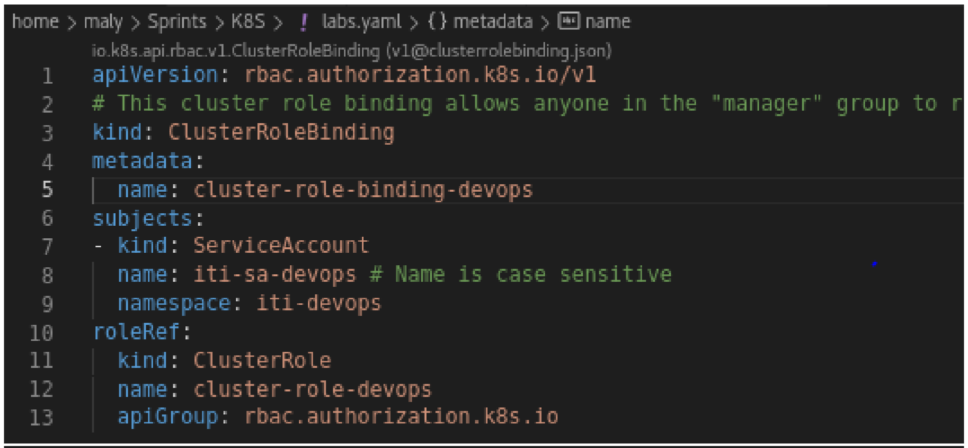


```
[maly@localhost K8S]$ kubectl apply -f labs.yaml
clusterrole.rbac.authorization.k8s.io/cluster-role-devops created
[maly@localhost K8S]$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   21d
[maly@localhost K8S]$ kubectl get clusterrole
NAME                                                                   CREATED AT
admin                                                                  2023-01-17T00:05:00Z
cluster-admin                                                          2023-01-17T00:05:00Z
cluster-role-devops                                                    2023-02-07T21:13:39Z
```



```
[maly@localhost K8S]$ kubectl describe clusterrole cluster-role-devops
Name:         cluster-role-devops
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources        Non-Resource URLs  Resource Names  Verbs
  ---------        -----------------  --------------  -----
  configMaps       []                 []              [get watch list create patch update]
  endpoints        []                 []              [get watch list create patch update]
  events           []                 []              [get watch list create patch update]
  namespaces       []                 []              [get watch list create patch update]
  nodes            []                 []              [get watch list create patch update]
  pods             []                 []              [get watch list create patch update]
  secrets          []                 []              [get watch list create patch update]
  serviceAccounts  []                 []              [get watch list create patch update]
  services         []                 []              [get watch list create patch update]
[maly@localhost K8S]$
```

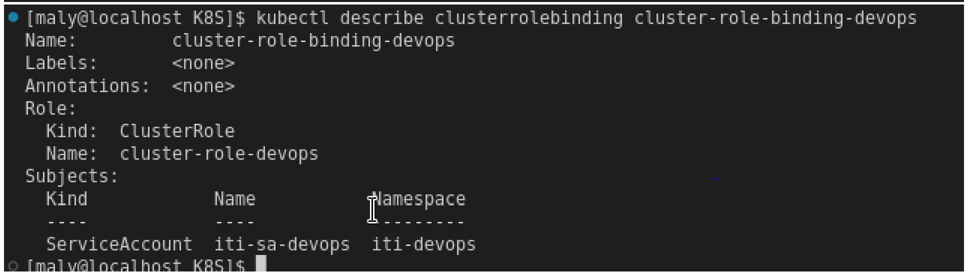
4- create a ClusterRoleBinding which should be named as cluster-role-binding-devops under the same namespace. Define roleRef apiGroup should be rbac.authorization.k8s.io. Kind should be ClusterRole, name should be cluster-role-devops and subjects' kind should be ServiceAccount: name should be iti-sadevops and namespace should be iti-devops



home > maly > Sprints > K8S > labs.yaml > {} metadata > name

io.k8s.api.rbac.v1.ClusterRoleBinding (v1@clusterrolebinding.json)

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to r
kind: ClusterRoleBinding
metadata:
  name: cluster-role-binding-devops
subjects:
- kind: ServiceAccount
  name: iti-sa-devops # Name is case sensitive
  namespace: iti-devops
roleRef:
  kind: ClusterRole
  name: cluster-role-devops
  apiGroup: rbac.authorization.k8s.io
```



```
[maly@localhost K8S]$ kubectl describe clusterrolebinding cluster-role-binding-devops
Name:         cluster-role-binding-devops
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cluster-role-devops
Subjects:
  Kind            Name           Namespace
  ----            ----           ---------
  ServiceAccount  iti-sa-devops  iti-devops
[maly@localhost K8S]$
```

5- What is the difference between statefulSets and deployments?

Deployments:

A Deployment is a Kubernetes resource object that provides declarative updates for pods that encapsulate application containers. A Deployment represents a number of identical pods without unique IDs, while specifying the pods' desired state and attributes. Deployments are typically used to auto scale the number of pod replicas, perform controlled rollouts for application code, and perform rollbacks when necessary.

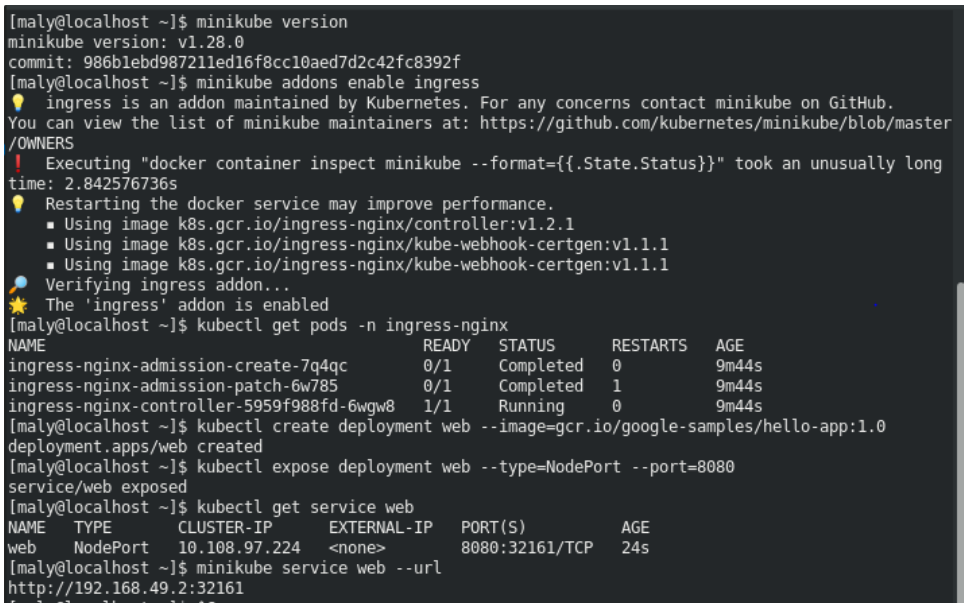
Kubernetes administrators rely on Deployments to manage a containerized application's lifecycle by defining the number of pods to be deployed, the image to be used for the application, and how to perform code updates. Kubernetes deployments help automate repeatable application updates, subsequently reducing the effort, time, and number of errors associated with manual updates.

StatefulSets:

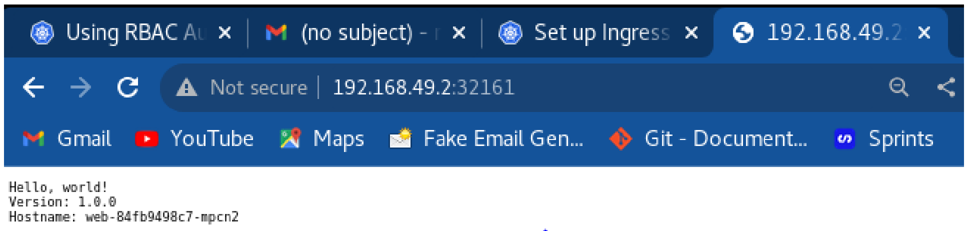
A StatefulSet is a Kubernetes resource object that manages a set of pods with unique identities. By assigning a persistent ID that is maintained even if the pod is rescheduled, a StatefulSet helps maintain the uniqueness and ordering of pods. With unique pod identifiers, administrators can efficiently attach cluster volumes to new pods across failures.

Although the StatefulSet controller deploys pods using similar specifications, pods are not interchangeable. As a StatefulSet does not create a ReplicaSet, the pod replicas cannot be rolled back to previous versions. StatefulSets are typically used for applications that require persistent storage for stateful workloads, and ordered, automated rolling updates.

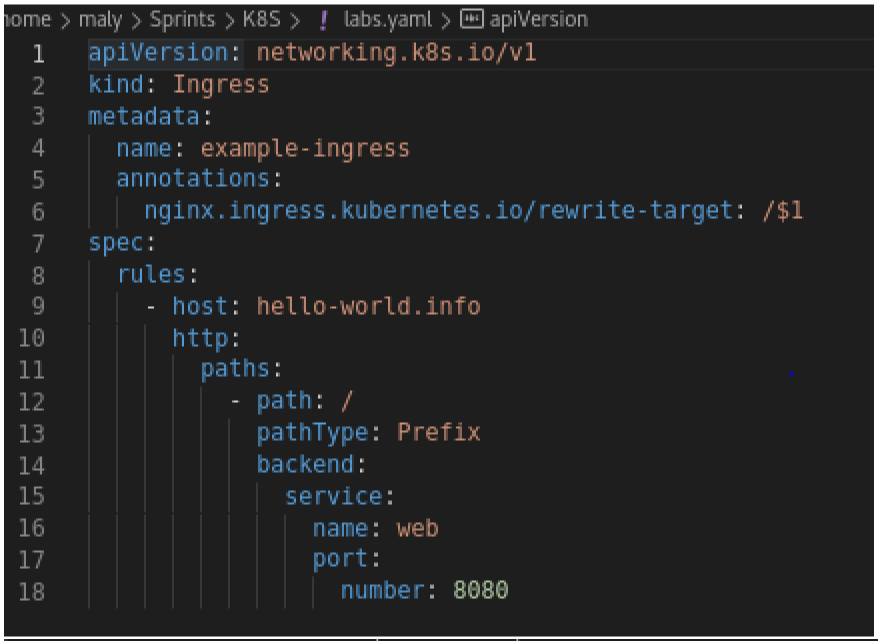
6- Set up Ingress on Minikube with the NGINX Ingress Controller play around with paths, you can create more than 2 deployments if you like
https://kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/



```
[maly@localhost ~]$ minikube version
minikube version: v1.28.0
commit: 986b1ebd987211ed16f8cc10aed7d2c42fc8392f
[maly@localhost ~]$ minikube addons enable ingress
💡  ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master
/OWNERS
❗  Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long
time: 2.842576736s
💡  Restarting the docker service may improve performance.
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v1.2.1
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
[maly@localhost ~]$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-7q4qc        0/1     Completed   0          9m44s
ingress-nginx-admission-patch-6w785         0/1     Completed   1          9m44s
ingress-nginx-controller-5959f988fd-6wgw8   1/1     Running     0          9m44s
[maly@localhost ~]$ kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/web created
[maly@localhost ~]$ kubectl expose deployment web --type=NodePort --port=8080
service/web exposed
[maly@localhost ~]$ kubectl get service web
NAME   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
web    NodePort   10.108.97.224   <none>        8080:32161/TCP   24s
[maly@localhost ~]$ minikube service web --url
http://192.168.49.2:32161
```

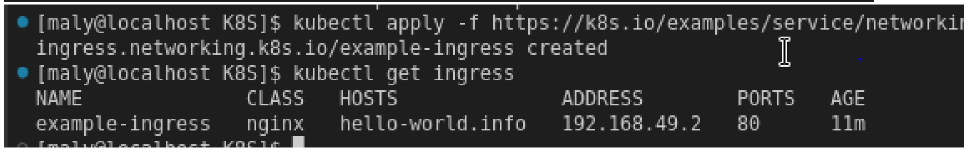


```
Hello, world!
Version: 1.0.0
Hostname: web-84fb9498c7-mpcn2
```

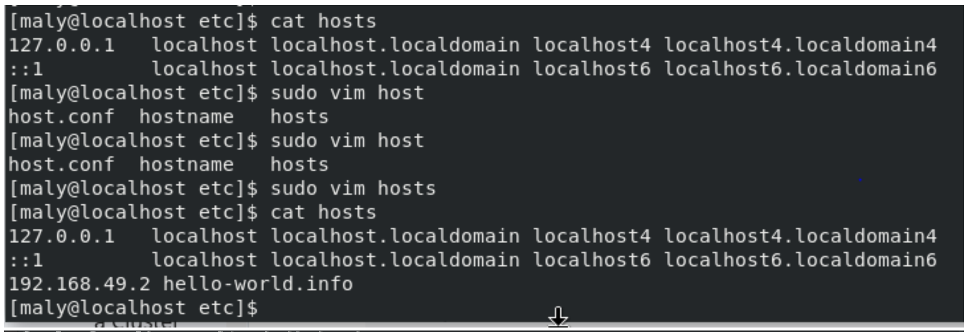


ome > maly > Sprints > K8S > labs.yaml > apiVersion

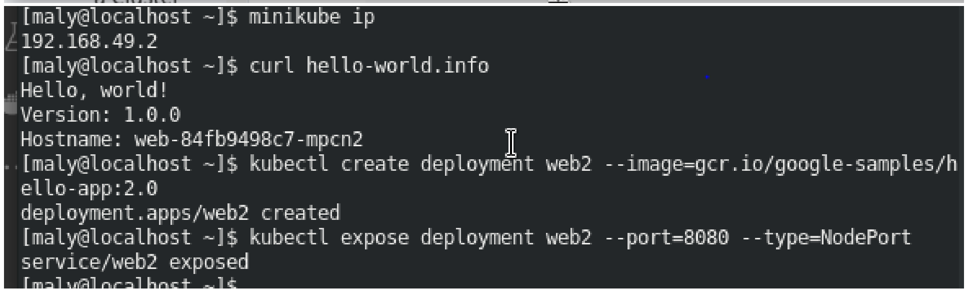
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
```



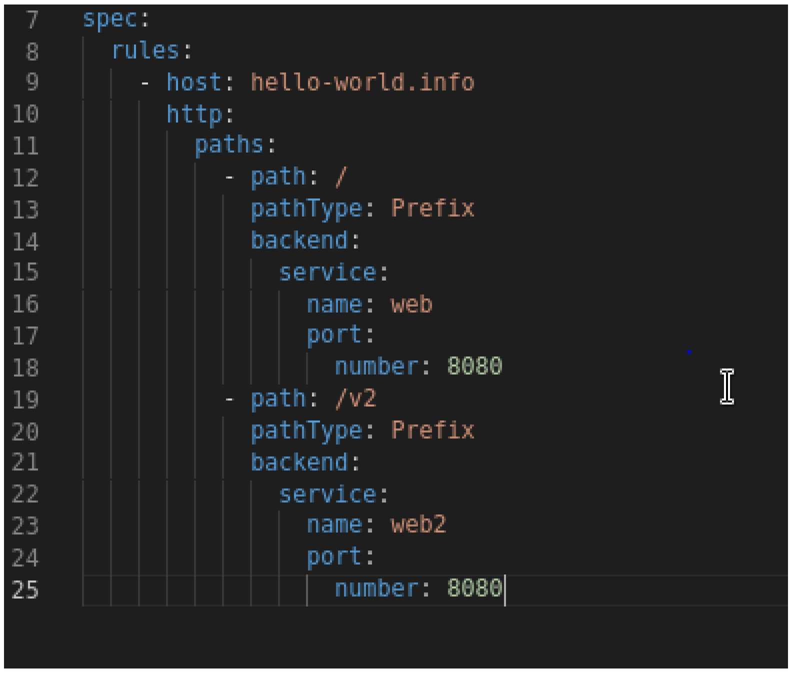
```
[maly@localhost K8S]$ kubectl apply -f https://k8s.io/examples/service/networki
ingress.networking.k8s.io/example-ingress created
[maly@localhost K8S]$ kubectl get ingress
NAME              CLASS   HOSTS              ADDRESS        PORTS   AGE
example-ingress   nginx   hello-world.info   192.168.49.2   80      11m
[maly@localhost K8S]$
```



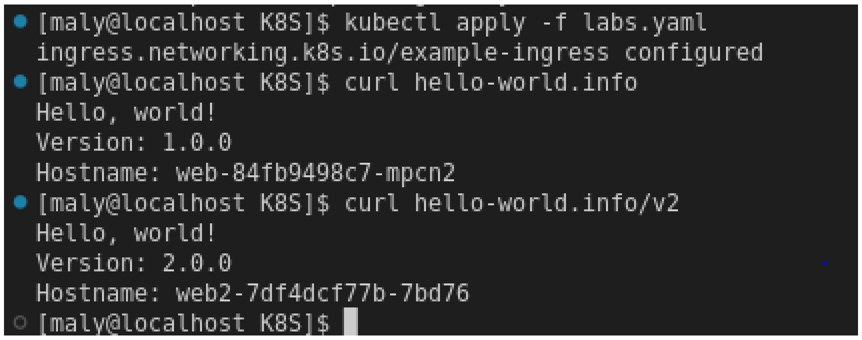
```
[maly@localhost etc]$ cat hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
[maly@localhost etc]$ sudo vim host
host.conf  hostname   hosts
[maly@localhost etc]$ sudo vim host
host.conf  hostname   hosts
[maly@localhost etc]$ sudo vim hosts
[maly@localhost etc]$ cat hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.49.2 hello-world.info
[maly@localhost etc]$
```



```
[maly@localhost ~]$ minikube ip
192.168.49.2
[maly@localhost ~]$ curl hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-84fb9498c7-mpcn2
[maly@localhost ~]$ kubectl create deployment web2 --image=gcr.io/google-samples/h
ello-app:2.0
deployment.apps/web2 created
[maly@localhost ~]$ kubectl expose deployment web2 --port=8080 --type=NodePort
service/web2 exposed
[maly@localhost ~]$
```



```yaml
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
          - path: /v2
            pathType: Prefix
            backend:
              service:
                name: web2
                port:
                  number: 8080
```



```
[maly@localhost K8S]$ kubectl apply -f labs.yaml
ingress.networking.k8s.io/example-ingress configured
[maly@localhost K8S]$ curl hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-84fb9498c7-mpcn2
[maly@localhost K8S]$ curl hello-world.info/v2
Hello, world!
Version: 2.0.0
Hostname: web2-7df4dcf77b-7bd76
[maly@localhost K8S]$
```