Kubenetes - Part 5

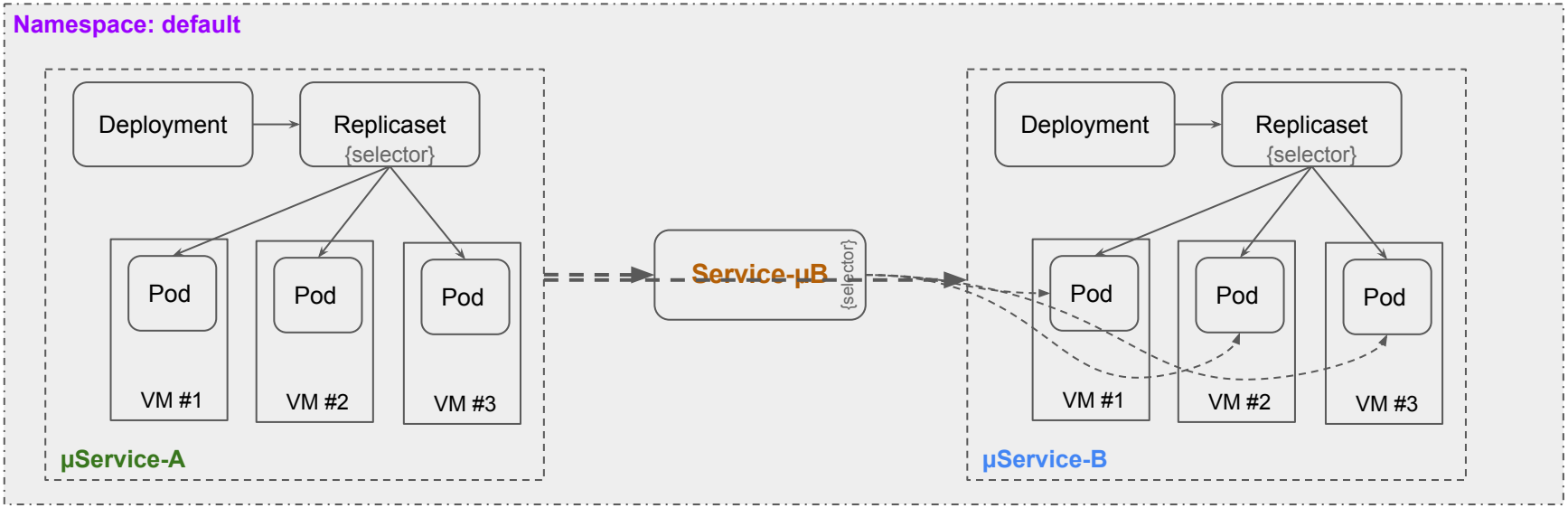
Solar Team

Version: 1.2

Agenda

- Kubernetes Services
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName
- Ingress

Service is an Abstraction!



- A Pod is not intended to be treated as a durable, long-lived entity.
- Pods have their own IP address.
- The Service abstraction enables the **decoupling** communication between microservices.

Services Overview

- Services are an abstract way to expose an application running on a set of Pods as a network service.
- Kubernetes uses a single DNS name for a set of pods and can load-balance across them.
- Service uses a label selector to determine pods.
- Services provide discoverable names and load balancing to Pod replicas.

Exposing Service Basic Example

- Using --expose option when creating a Pod.
  - kubectl run nginx --image=nginx --restart=Never **--port=80 --expose**
- Uses kubeclt expose command to expose service from existing Pod(s).
  - Possible resources include pod, service, deployment, replicaset.
  - Create deployment
    - kubectl run nginx --image=nginx
    - Or, kubectl create deploy nginx --image=nginx
  - Expose deployment as a service
    - kubectl expose deploy nginx --port=8080 **--target-port=80**
    - --port= port that the service should serve on.
    - --target-port= port on the container that the service should direct traffic to.
- Testing service with cURL

Exposing Service - Service Seletor

```
kubectl describe po nginx
Name:           nginx-65f88748fd-rt2h4
Namespace:      default
Priority:       0
Node:           docker-desktop/192.168.65.3
Start Time:     Mon, 17 May 2021 09:17:17 +0700
Labels:         app=nginx
                pod-template-hash=65f88748fd
Annotations:    <none>
Status:         Running
IP:             10.1.2.26
IPs:            <none>
Controlled By:  ReplicaSet/nginx-65f88748fd
Containers:
  nginx:
```

```
kubectl describe svc nginx
Name:              nginx
Namespace:         default
Labels:            app=nginx
Annotations:       <none>
Selector:          app=nginx
Type:              ClusterIP
IP:                10.98.167.206
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         10.1.2.26:80
Session Affinity:  None
Events:            <none>
```

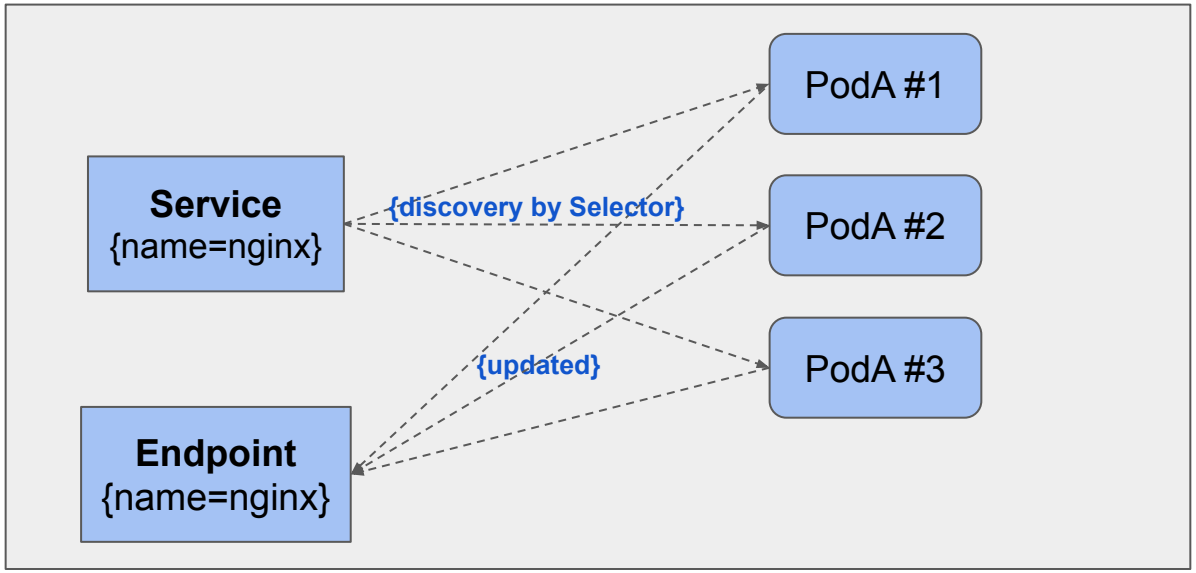
```
kubectl describe endpoints nginx
Name:         nginx
Namespace:    default
Labels:       app=nginx
Annotations:  endpoints.kubernetes.io/last-change-
Subsets:
  Addresses:          10.1.2.26
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
```

Service object named "**nginx**", which targets TCP port 80 on any Pod with the **app=nginx label**.

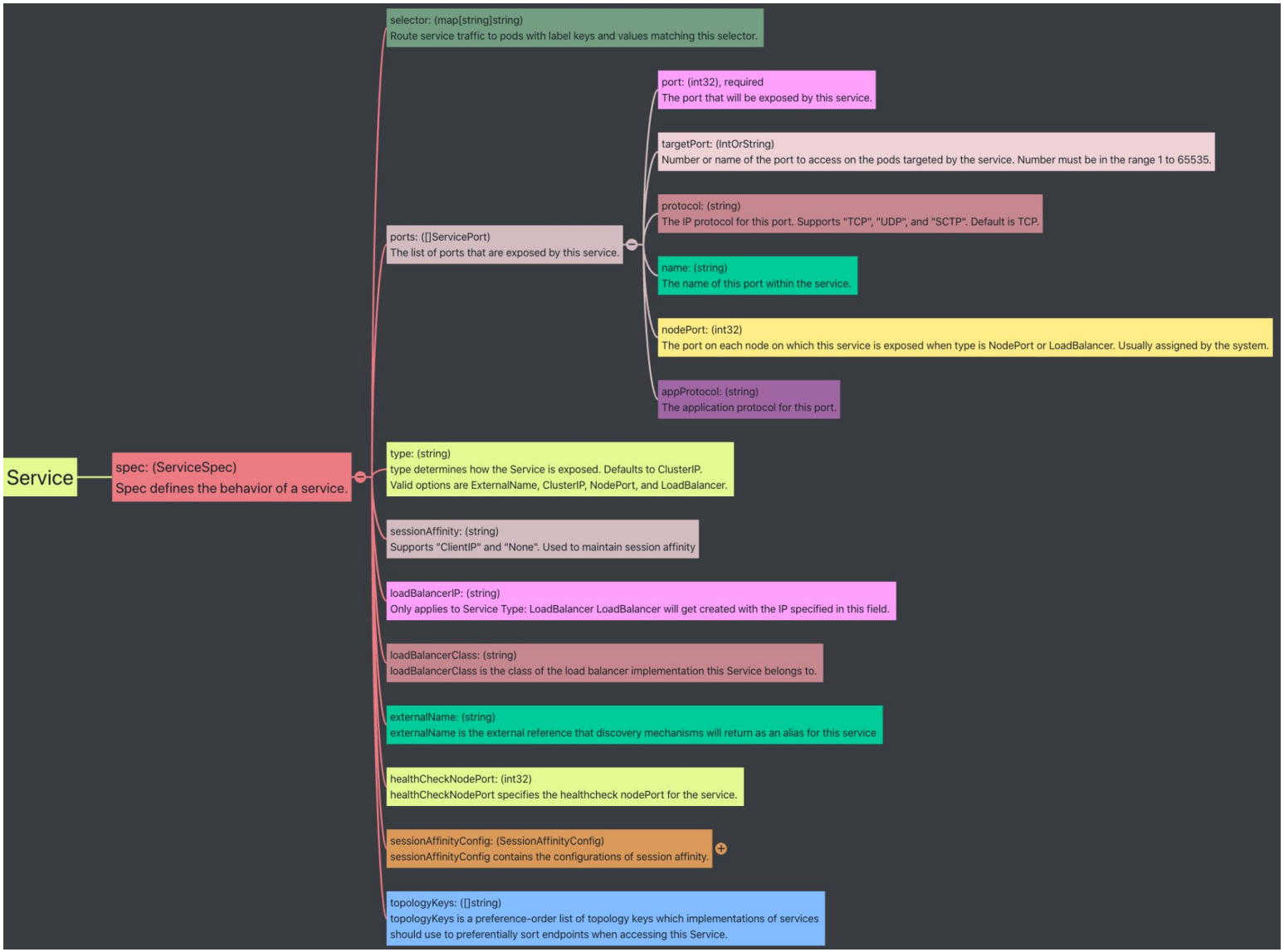
Notes: label app=nginx or run=nginx

Service and Endpoints

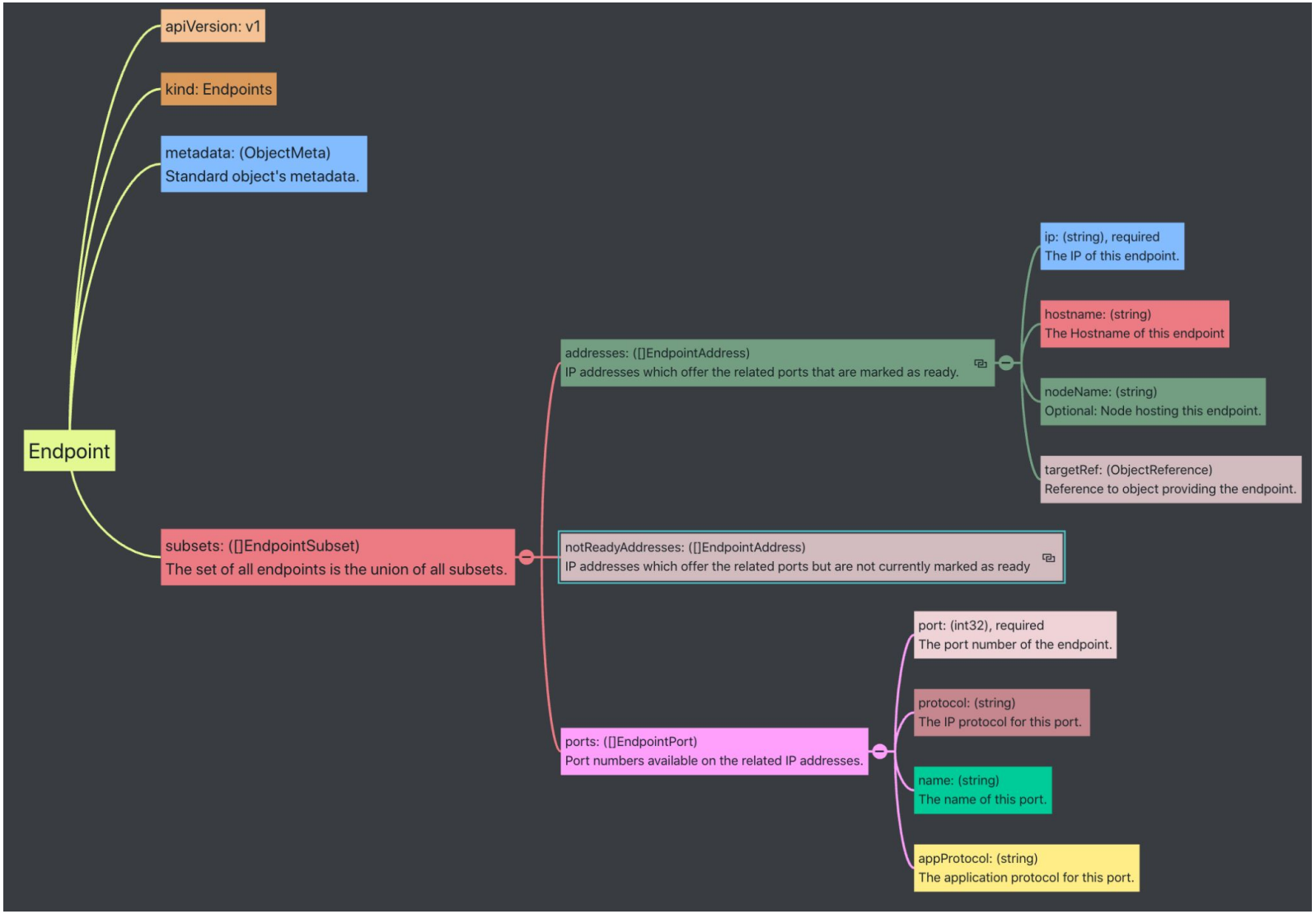


- The **controller** for the Service's selector will be evaluated continuously and the results will be POSTed to an Endpoints object.
- When a Pod dies, it is automatically removed from the endpoints.
- New Pods matching the Service's selector will automatically get added to the endpoints.
- Endpoints track the IP Addresses of the objects the service send traffic to.
- Services without selectors

Service Object (partial)



Endpoints Object



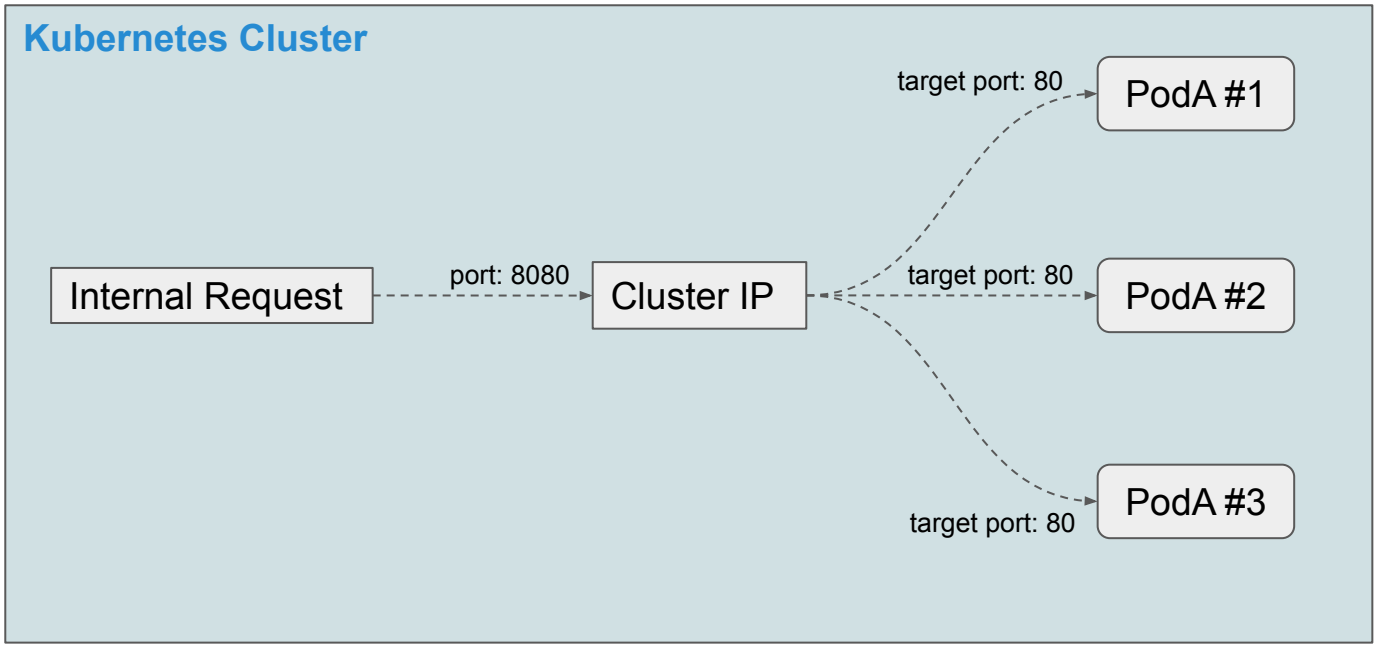
Publishing Services Using ServiceTypes

- Kubernetes ServiceTypes allow you to specify what kind of Service you want.

Types	Client	Summary
ClusterIP	Internal	This is the **default ServiceType**. Exposes the Service on a cluster-internal IP.
NodePort	External	Exposes the Service on each Node's IP at a **static port**.
LoadBalancer	External	Exposes the Service externally using a **cloud provider's load balancer.**
ExternalName	External	Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com)

ServiceType - ClusterIP

- Exposes the Service on a cluster-internal IP.
- Choosing this type makes the Service only reachable from within the cluster.
- This is the default ServiceType.



ServiceType - ClusterIP Examples

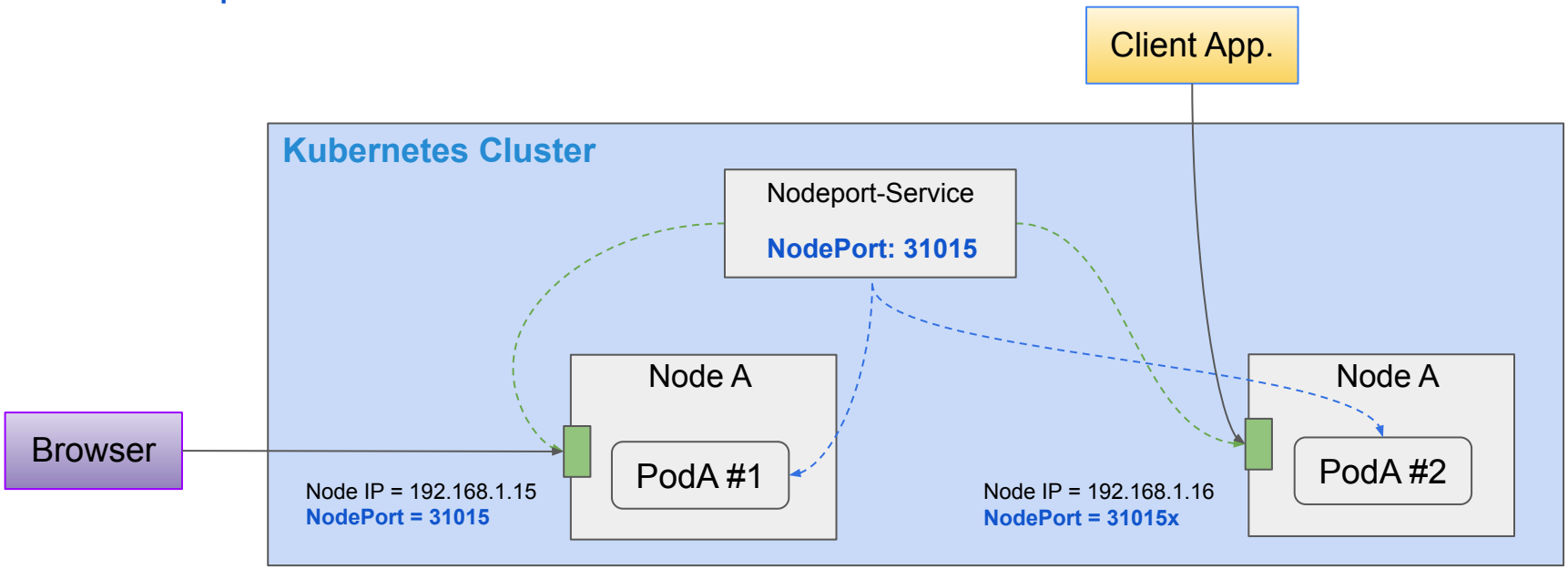
```yaml
apiVersion: v1
kind: Service
metadata:
 labels:
   app: nginx
 name: nginx
 namespace: default
spec:
 ports:
 - port: 8080
   protocol: TCP
   targetPort: 80
 selector:
   app: nginx
 type: ClusterIP
```

nginx-service.yml

- kubectl create deploy nginx --image=nginx
- kubectl expose deploy nginx --name nginx --port 8080 --target-port 80
- Within Busybox
  - nslookup nginx
  - curl nginx:8080
- kubectl scale --replicas=3 deploy nginx
- kubectl get po -lapp=nginx -owide

ServiceType - NodePort

- Exposing Kubernetes services to external clients.
- Kubernetes exposes the Service on each Node's IP at a static port (the NodePort in the default range of 30000-32767).
- You can specify node port or let Kubernetes automatic created.
- External clients can access the Service from outside the cluster by requesting <NodeIp>:<NodePort>



ServiceType - NodePort (Random Port)

```yaml
apiVersion: v1
kind: Service
metadata:
 name: nodeport-service
 labels:
   app: nginx
spec:
 selector:
   app: nginx
 ports:
 - port: 8080
   targetPort: 80
 type: NodePort
```

nodeport-service.yml

```
kubectl describe svc  nodeport-service
Name:                     nodeport-service
Namespace:                default
Labels:                   app=nginx
Annotations:              kubectl.kubernetes.io/last-applied-conf
                            {"apiVersion":"v1","kind":"Service","
service","namespace":"default"}...
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.96.31.93
LoadBalancer Ingress:     localhost
Port:                     <unset>  8080/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30997/TCP
Endpoints:                10.1.2.38:80,10.1.2.39:80,10.1.2.40:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

```
kubectl get po -l app=nginx -owide
NAME                     READY   STATUS    RESTARTS   AGE     IP
nginx-65f88748fd-djflr   1/1     Running   0          2d16h   10.1.2.40
nginx-65f88748fd-j9z4f   1/1     Running   0          2d16h   10.1.2.39
nginx-65f88748fd-rt2h4   1/1     Running   3          10d     10.1.2.38
```

```
kubectl describe ep nodeport-service
Name:         nodeport-service
Namespace:    default
Labels:       app=nginx
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2021-05-25T10:14:07Z
Subsets:
  Addresses:          10.1.2.38,10.1.2.39,10.1.2.40
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
```

Pod ip(s)

- curl localhost:30997

ServiceType - NodePort (Specify port number)

```yaml
apiVersion: v1
kind: Service
metadata:
 name: nodeport-service
 labels:
   app: nginx
spec:
 selector:
   app: nginx
 ports:
 - port: 8080
   targetPort: 80
   nodePort: 310001
 type: NodePort
```

nodeport31001-service.yml

```
kubectl describe svc  nodeport31001-service
Name:                     nodeport31001-service
Namespace:                default
Labels:                   app=nginx
Annotations:              kubectl.kubernetes.io/last-applied-conf
                            {"apiVersion":"v1","kind":"Service","
1001-service","namespace":"defa...
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.109.187.59
LoadBalancer Ingress:     localhost
Port:                     <unset>  8080/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  31001/TCP
Endpoints:                10.1.2.38:80,10.1.2.39:80,10.1.2.40:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

```
kubectl get po -l app=nginx -owide
NAME                     READY   STATUS    RESTARTS   AGE     IP
nginx-65f88748fd-djflr   1/1     Running   0          2d16h   10.1.2.40
nginx-65f88748fd-j9z4f   1/1     Running   0          2d16h   10.1.2.39
nginx-65f88748fd-rt2h4   1/1     Running   3          10d     10.1.2.38
```

```
kubectl describe ep  nodeport31001-service
Name:         nodeport31001-service
Namespace:    default
Labels:       app=nginx
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2021-05-25T10:49:03Z
Subsets:
  Addresses:          10.1.2.38,10.1.2.39,10.1.2.40
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
```

Pod ip(s)

Warning - Need to take care of possible port collisions yourself.

Create NodePort Service Using Imperative Way

- kubectl create svc nodeport --node-port=31013 --tcp=8080:80 **nginx**

```
kubectl describe svc nginx
Name:                     nginx
Namespace:                default
Labels:                   app=nginx
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.101.202.91
LoadBalancer Ingress:     localhost
Port:                     8080-80  8080/TCP
TargetPort:               80/TCP
NodePort:                 8080-80  31013/TCP
Endpoints:                10.1.2.38:80,10.1.2.39:80,10.1.2.40:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Note: “nginx” was used as a Selector.

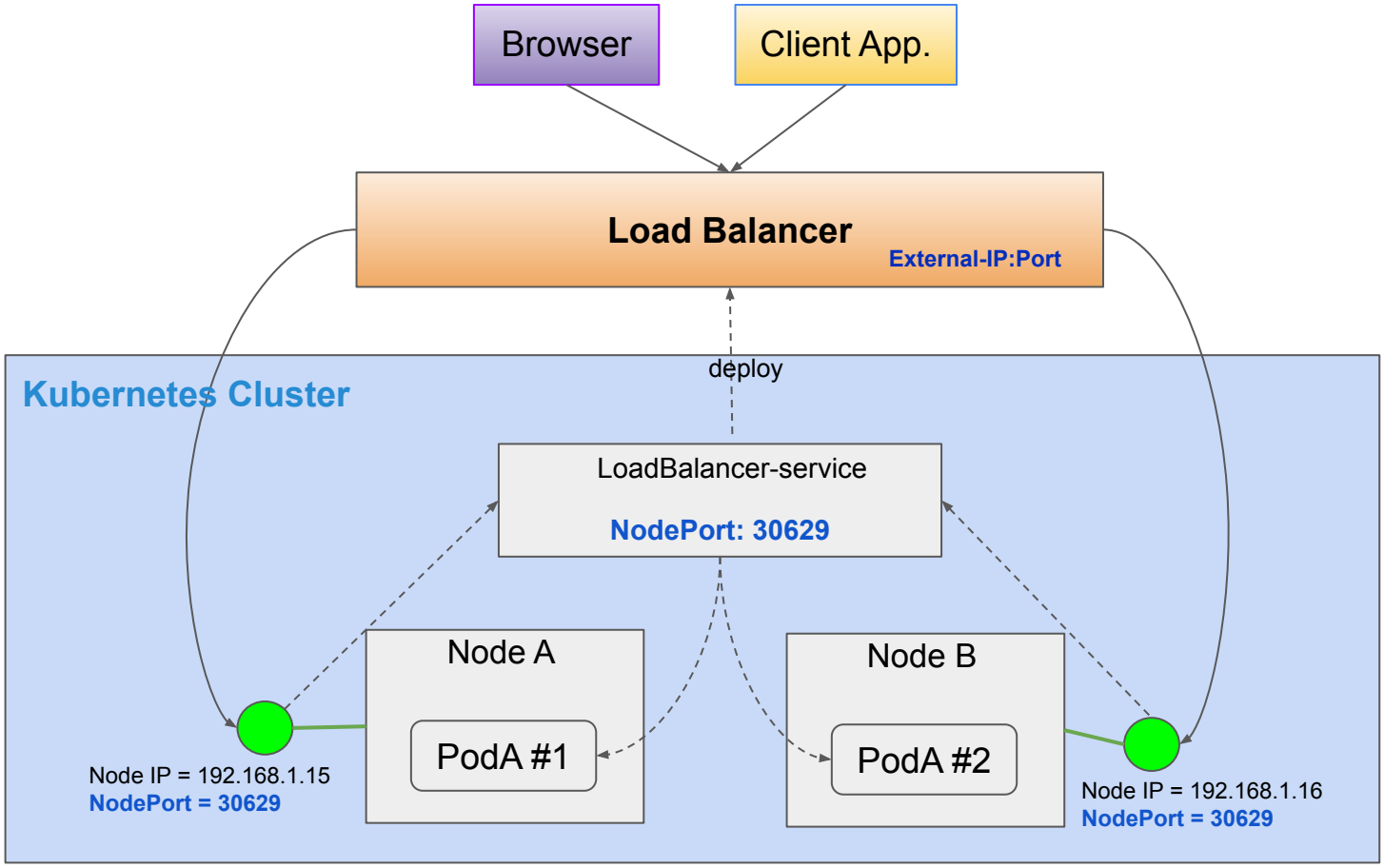
- kubectl expose deploy nginx --port=8080 --target-port=80 --type NodePort --name nginx-svc2

```
kubectl describe svc nginx-svc2
Name:                     nginx-svc2
Namespace:                default
Labels:                   app=nginx
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.104.233.73
LoadBalancer Ingress:     localhost
Port:                     <unset>  8080/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32652/TCP
Endpoints:                10.1.2.38:80,10.1.2.39:80,10.1.2.40:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

ServiceType - LoadBalancer

- A LoadBalancer service is the standard way to expose a service to the internet.
- On cloud providers, setting the type field to “LoadBalancer” automatically deploys an external load balancer and provided a load balancer for your Service.
- The exact implementation of a LoadBalancer is dependent on your cloud provider.
- The load balancer will have its own unique, publicly accessible IP address.
- Client access your services through the load balancer’s IP address.
- a LoadBalancer service is an extension of a NodePort service.
- metadata.annotations are used to define the properties and features of each cloud provider.

ServiceType - LoadBalancer



https://kubernetes.io/docs/concepts/services-networking/service/#internal-load-balancer
https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/#preserving-the-client-source-ip

ServiceType - LoadBalancer Local Example

```yaml
apiVersion: v1
kind: Service
metadata:
 name: lb-service
spec:
 selector:
   app: nginx
 ports:
   - protocol: TCP
     port: 8080
     targetPort: 80
 type: LoadBalancer
```

lb-service.yml

```
kubectl describe  svc lb-service
Name:                     lb-service
Namespace:                default
Labels:                   <none>
Annotations:              kubectl.kubernetes.io/last-applied-confi
                            {"apiVersion":"v1","kind":"Service","m
rt":8080,...
Selector:                 app=nginx
Type:                     LoadBalancer
IP:                       10.109.242.6
LoadBalancer Ingress:     localhost
Port:                     <unset>  8080/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30629/TCP
Endpoints:                10.1.2.38:80,10.1.2.39:80,10.1.2.40:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

```
kubectl get ep lb-service
NAME         ENDPOINTS                                      AGE
lb-service   10.1.2.38:80,10.1.2.39:80,10.1.2.40:80   14m
```

- curl localhost:8080

Accessing LoadBalancer Service

```
kubectl get svc nodeport-service lb-service
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
nodeport-service   NodePort       10.106.1.65    <none>        8080:30567/TCP   6h
lb-service         LoadBalancer   10.109.242.6   localhost     8080:30629/TCP   31m
```

```
kubectl get ep nodeport-service lb-service
NAME               ENDPOINTS                                      AGE
nodeport-service   10.1.2.38:80,10.1.2.39:80,10.1.2.40:80         6h2m
lb-service         10.1.2.38:80,10.1.2.39:80,10.1.2.40:80         33m
```

```
curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

LoadBalancer

```
curl localhost:30629
curl: (7) Failed to connect to localhost port 30629: Connection refused
```

LoadBalancer

```
curl localhost:30567
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
```

NodePort

ServiceType - LoadBalancer Service YAML

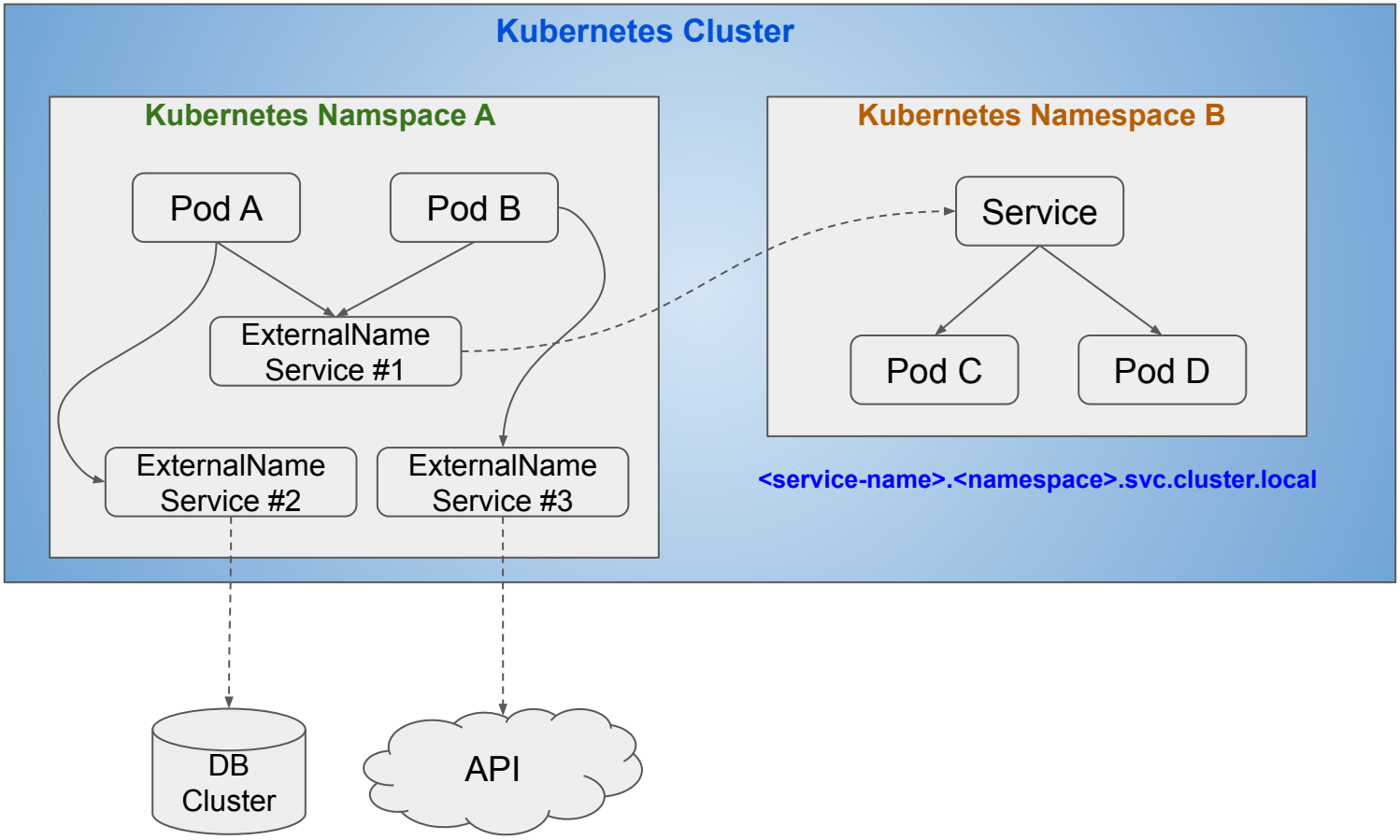
```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name":"lb-service","namespace":"default"},"spec":{"ports":[{"
port":8080,"protocol":"TCP","targetPort":80}],"selector":{"app":"nginx"},"type":"LoadBalancer"}}
  creationTimestamp: "2021-05-27T03:07:28Z"
  name: lb-service
  namespace: default
  resourceVersion: "3046010"
  selfLink: /api/v1/namespaces/default/services/lb-service
  uid: ab52795a-be98-11eb-90fc-025000000001
spec:
  clusterIP: 10.107.160.172
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 31033
    port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - hostname: localhost
```

ServiceType - ExternalName

- Required to refer to external services from applications inside your Kubernetes cluster.
- External services may be,
  - Services in another Kubernetes Namespace.
  - Services outside a Kubernetes Cluster.
- An ExternalName Service is a special case of Service that does not have selectors and uses DNS names instead.
- Services of type ExternalName map a Service to a DNS name (not an IP/port).

ServiceType - ExternalName



ServiceType - ExternalName Example #1

- Call other namespace service.

```yaml
apiVersion: v1
kind: Service
metadata:
 name: other-ns-service
spec:
 type: ExternalName
 externalName: hello.test.svc.cluster.local
```

other-ns-service.yml

- kubectl create ns test
- kubectl run hello --image=gcr.io/google-samples/hello-app:1.0 -n test
- kubectl scale --replicas=3 deploy hello -n test
- kubectl expose deploy hello --port=8090 --target-port=8080 --name hello -n test
- kubectl apply -f other-ns-service.yml
- curl other-ns-service:8090 # within busybox

ServiceType - ExternalName Example #2

- Call external API

```yaml
apiVersion: v1
kind: Service
metadata:
 name: testapi
spec:
 type: ExternalName
 externalName: postman-echo.com
```

postman-echo-service.yml

```python
import requests
response =
requests.get('http://testapi/get?arg1=value123')
print(response.text)
```

client.py

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "./client.py"]
```

Dockerfile

- kubectl apply -f postman-echo-service.yml
- curl testapi/get?a=b #within busybox
- kubectl run apiclient --restart=Never --image=apiclient:v1.0 # check pod's log

Services without selectors - Mapping a hostname to an IP

```yaml
kind: "Service"
apiVersion: "v1"
metadata:
 name: "mariadb"
spec:
 ports:
   - name: "ext-mariadb"
protocol: "TCP"
     port: 3333
     targetPort: 3306
```

external-mariadb.yml

```yaml
kind: "Endpoints"
apiVersion: "v1"
metadata:
 name: "mariadb"
subsets:
 - addresses:
     - ip: "192.168.1.16"
   ports:
     - port: 3306
       name: "ext-mariadb"
```

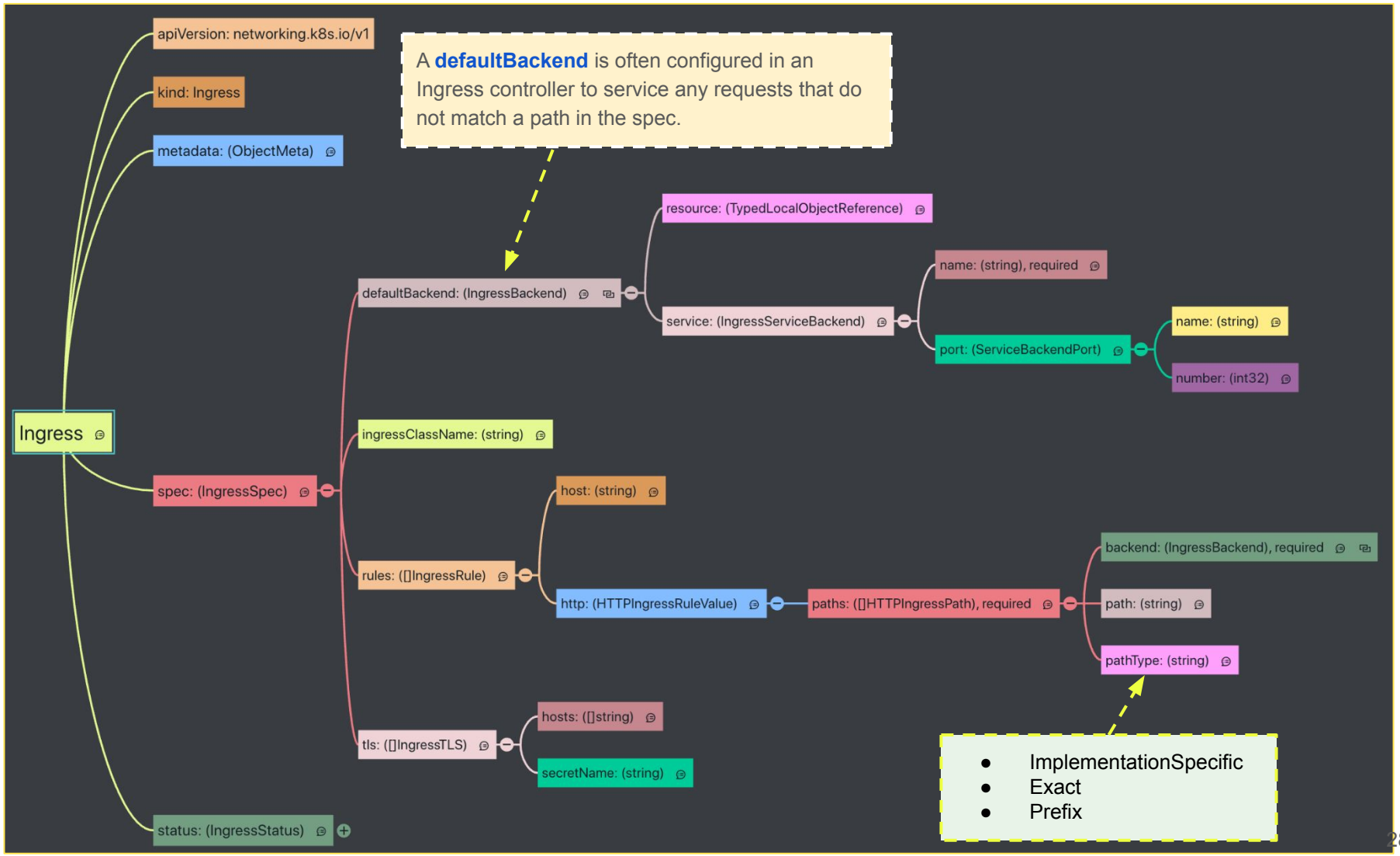
```
/ # telnet -l root mariadb 3333
X
5.5.5-10.5.8-MariaDBW;`R[Ri8??-y|"0kKu,w6G@mysql_native_password
Connection closed by foreign host
```

- Because this Service has no selector, the corresponding Endpoints object is not created automatically.

Ingress Overview

- Ingress is an Kubernetes API object that exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- Ingress is not a Service type, but it a reverse proxy and single entry-point to your cluster that routes the request to different services.
- it can expose multiple services under the same IP address.
- Traffic routing is controlled by rules defined on the Ingress resource.
- An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name-based virtual hosting.
- You must have an Ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect.
- Ingress frequently uses annotations to configure some options depending on the Ingress controller.

Ingress Object



Hostname and PathType

Hosts can be precise matches (for example "`foo.bar.com`") or a wildcard (for example "`*.foo.com`").

Host	Host header	Match?
`*.foo.com`	`bar.foo.com`	Matches based on shared suffix
`*.foo.com`	`baz.bar.foo.com`	No match, wildcard only covers a single DNS label
`*.foo.com`	`foo.com`	No match, wildcard only covers a single DNS label

Kind	Path(s)	Request path(s)	Matches?
Prefix	`/`	(all paths)	Yes
Exact	`/foo`	`/foo`	Yes
Exact	`/foo`	`/bar`	No
Exact	`/foo`	`/foo/`	No
Exact	`/foo/`	`/foo`	No
Prefix	`/foo`	`/foo` , `/foo/`	Yes
Prefix	`/foo/`	`/foo` , `/foo/`	Yes
Prefix	`/aaa/bb`	`/aaa/bbb`	No
Prefix	`/aaa/bbb`	`/aaa/bbb`	Yes
Prefix	`/aaa/bbb/`	`/aaa/bbb`	Yes, ignores trailing slash
Prefix	`/aaa/bbb`	`/aaa/bbb/`	Yes, matches trailing slash
Prefix	`/aaa/bbb`	`/aaa/bbb/ccc`	Yes, matches subpath
Prefix	`/aaa/bbb`	`/aaa/bbbxyz`	No, does not match string prefix
Prefix	`/` , `/aaa`	`/aaa/ccc`	Yes, matches `/aaa` prefix
Prefix	`/` , `/aaa` , `/aaa/bbb`	`/aaa/bbb`	Yes, matches `/aaa/bbb` prefix
Prefix	`/` , `/aaa` , `/aaa/bbb`	`/ccc`	Yes, matches `/` prefix
Prefix	`/aaa`	`/ccc`	No, uses default backend
Mixed	`/foo` (Prefix), `/foo` (Exact)	`/foo`	Yes, prefers Exact

Ingress Controller

- An ingress controller is responsible for reading the Ingress Resource information and processing that data accordingly.
- A collection of routing rules that govern how external users access services running in a Kubernetes cluster.
- Different ingress controllers have extended the specification in different ways to support additional use cases.
- Note that an ingress controller typically doesn't eliminate the need for an external load balancer — the ingress controller simply adds an additional layer of routing and control behind the load balancer.
- An Ingress controller is bootstrapped with some load balancing policy settings that it applies to all Ingress.
- The most basic Ingress is the NGINX Ingress Controller.

Nginx Ingress Controller

- Install for Kubernetes 1.14
  - kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.35.0/deploy/static/provider/cloud/deploy.yaml
- kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.47.0/deploy/static/provider/cloud/deploy.yaml
- Ingress frequently uses annotations to configure some options depending on the Ingress controller.
- Different Ingress controller support different annotations.

```
✔ kubectl get all  -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-vrxdw    0/1     Completed   0          3d23h
pod/ingress-nginx-admission-patch-5z2jb     0/1     Completed   0          3d23h
pod/ingress-nginx-controller-68556b9795-gj4n5   1/1     Running     2          3d23h

NAME                                         TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer   10.96.31.105   localhost     80:31617/TCP,443:31755/TCP   3d23h
service/ingress-nginx-controller-admission   ClusterIP      10.98.31.172   <none>        443/TCP                      3d23h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           3d23h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-68556b9795   1         1         1       3d23h

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           11s        3d23h
job.batch/ingress-nginx-admission-patch    1/1           16s        3d23h
```

Additional controllers

- AKS Application Gateway Ingress Controller is an ingress controller that configures the Azure Application Gateway.
- Ambassador API Gateway is an Envoy-based ingress controller.
- Apache APISIX ingress controller is an Apache APISIX-based ingress controller.
- Avi Kubernetes Operator provides L4-L7 load-balancing using VMware NSX Advanced Load Balancer.
- The Citrix ingress controller works with Citrix Application Delivery Controller.
- Contour is an Envoy based ingress controller.
- EnRoute is an Envoy based API gateway that can run as an ingress controller.
- F5 BIG-IP Container Ingress Services for Kubernetes lets you use an Ingress to configure F5 BIG-IP virtual servers.
- Gloo is an open-source ingress controller based on Envoy, which offers API gateway functionality.
- HAProxy Ingress is an ingress controller for HAProxy.
- The HAProxy Ingress Controller for Kubernetes is also an ingress controller for HAProxy.
- Istio Ingress is an Istio based ingress controller.
- The Kong Ingress Controller for Kubernetes is an ingress controller driving Kong Gateway.
- The NGINX Ingress Controller for Kubernetes works with the NGINX webserver (as a proxy).
- Skipper HTTP router and reverse proxy for service composition, including use cases like Kubernetes Ingress, designed as a library to build your custom proxy.
- The Traefik Kubernetes Ingress provider is an ingress controller for the Traefik proxy.
- Tyk Operator extends Ingress with Custom Resources to bring API Management capabilities to Ingress. Tyk Operator works with the Open Source Tyk Gateway & Tyk Cloud control plane.
- Voyager is an ingress controller for HAProxy.

Ingress - Example#1 No Host

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-nohost
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 rules:
   - http:
       paths:
         - path: /hello
           backend:
             serviceName: hello
             servicePort: 80
         - path: /web
           backend:
             serviceName: nodeport-service
             servicePort: 8080
```

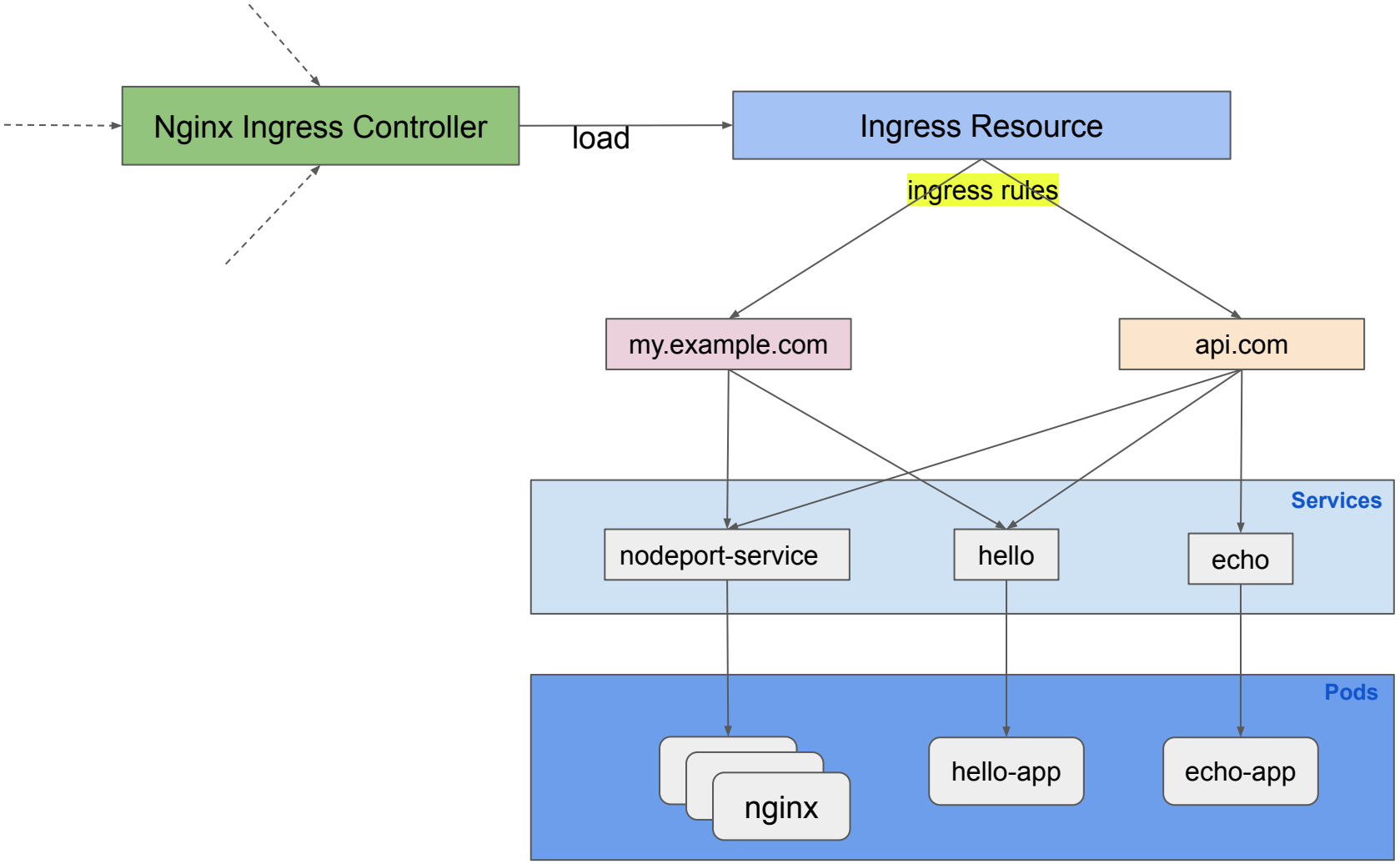
ingress-nohost.yml

- this rule applies to all inbound HTTP traffic through the IP address specified.

Ingress - Example#1 No Host Steps

- kubectl run hello --image=gcr.io/google-samples/hello-app:1.0
- kubectl scale --replicas=3 deploy hello
- kubectl expose deploy hello --port=80 --target-port=8080 --name hello
- kubectl apply -f ingress-nohost.yml
- curl localhost/hello
- kubectl delete deploy hello
- kubectl delete svc hello
- kubectl delete ingress ingress-nohost

Ingress - Example#2 Name Based Virtual Hosting



Ingress - Example #2 Ingress Resource

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-multiple
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 rules:
 - host: my.example.com
   http:
     paths:
     - path: /nginx
       backend:
         serviceName: nodeport-service
         servicePort: 8080
     - path: /hello
       backend:
         serviceName: hello
         servicePort: 80
```

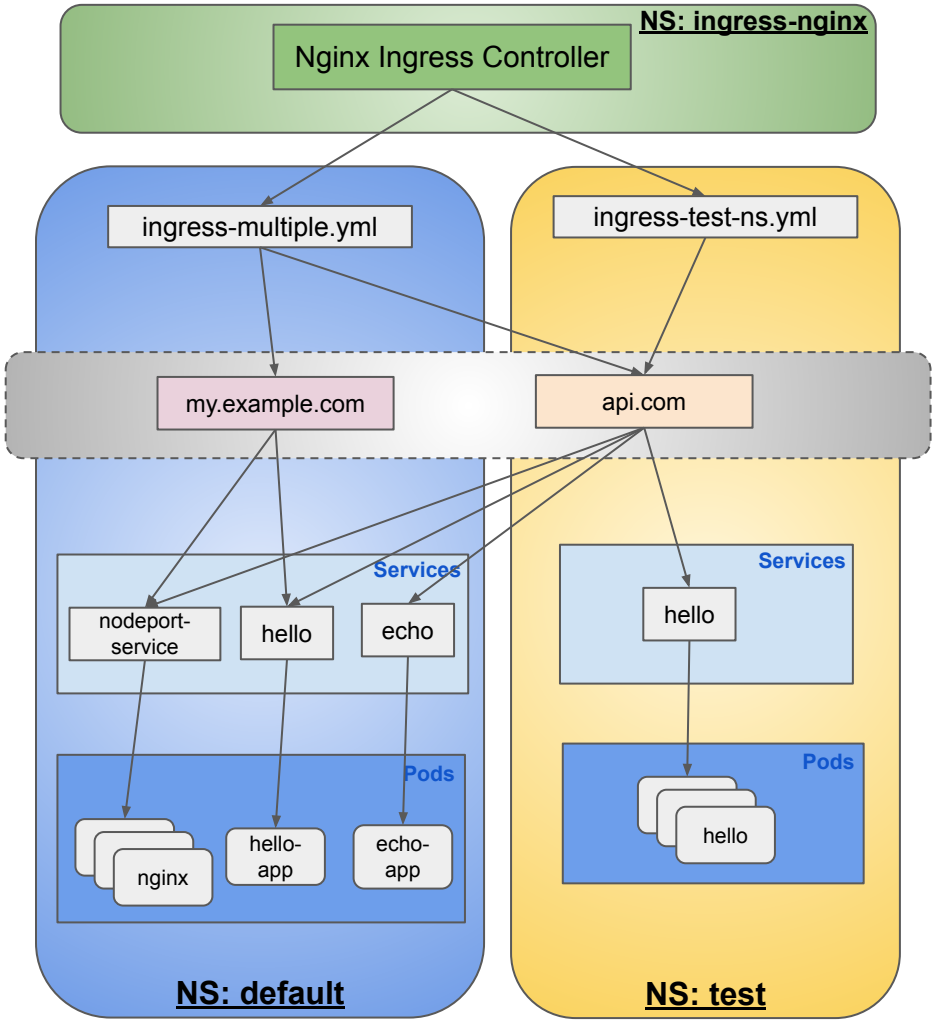
```yaml
- host: api.com
   http:
     paths:
     - path: /doc
       backend:
         serviceName: nodeport-service
         servicePort: 8080
     - path: /ping
       backend:
         serviceName: hello
         servicePort: 80
     - path: /echo
       backend:
         serviceName: echo
         servicePort: 80
```

ingress-multiple.yml

Ingress - Example #2 Steps

- kubectl create deployment hello-app --image=gcr.io/google-samples/hello-app:1.0
- kubectl create deployment echo-app --image=k8s.gcr.io/echoserver:1.4
- kubectl expose deployment hello-app --port=80 --target-port=8080 --name=hello
- kubectl expose deployment echo-app --port=80 --target-port=8080 --name=echo
- kubectl apply -f nodeport-service.yml
- kubectl logs -f -n ingress-nginx ingress-nginx-controller-68556b9795-gj4n5
- kubectl apply -f ingress-multiple.yml
- kubectl get ing
- Add “127.0.0.1 my.example.com api.com” to /etc/hosts file
- Tesing
  - curl --header "Host: my.example.com" http://localhost/hello
  - curl http://my.example.com/nginx
  - curl --header "Host: api.com" http://localhost/doc
  - curl --header "Host: api.com" http://localhost/echo\?a\=b
  - curl http://api.com/ping

Ingress - Multiple Namespaces Ingress Resource




```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: ingress-test-ns
 namespace: test
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 rules:
 - host: my.example.com
   http:
     paths:
     - path: /hello-test
       backend:
         serviceName: hello
         servicePort: 8090
```

ingress-test-ns.yml

- kubectl create deployment hello --image=gcr.io/google-samples/hello-app:1.0 -n test
- kubectl expose deploy hello --port=8090 --target-port=8080 --name hello -n test
- kubectl scale --replicas=3 deploy hello -n test
- curl my.example.com/hello # service in default namespace
- curl my.example.com/hello-test # service in test namespace

End.