

Pod

Exemple - server http

- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: www
    image: nginx:1.12.2
```

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

Cycle de vie

```
# Lancement du Pod
```

```
$ kubectl create -f nginx-pod.yaml
```

```
# Liste des Pods présents
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
www	1/1	Running	0	2m

```
# Lancement d'une commande dans un Pod
```

```
$ kubectl exec www -- nginx -v
```

```
nginx version: nginx/1.12.2
```

```
# Shell interactif dans un Pod
```

```
$ kubectl exec -t -i www -- /bin/bash
```

```
root@nginx:/#
```

Cycle de vie

```
# Détails du Pod
$ kubectl describe po/www
Name:          www
Namespace:     default
Node:          minikube/192.168.99.100
...
Events:
  Type            Reason              Age   From                Message
  ----            -
  Normal          Scheduled            18s   default-scheduler   Successfully assigned nginx to minikube
  Normal          SuccessfulMountVolume 18s   kubelet, minikube   MountVolume.SetUp succeeded for volume "default-token-brp4l"
  Normal          Pulled               18s   kubelet, minikube   Container image "nginx:1.12.2" already present on machine
  Normal          Created              18s   kubelet, minikube   Created container
  Normal          Started              18s   kubelet, minikube   Started container

# Suppression du Pod
$ kubectl delete pod www
pod "www" deleted
```

Pod avec plusieurs containers

```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
  - image: wordpress:4.9-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_PASSWORD
      value: mysqlpwd
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
  - image: mysql:5.7
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: mysqlpwd
    volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
  volumes:
  - name: data
    emptyDir: {}
```


Service

Spécification : exemple de type ClusterIP

```
$ cat vote-service-clusterIP.yaml
```

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  name: vote-service
```

```
spec:
```

```
  selector:
```

```
    app: vote
```

```
  type: ClusterIP
```

```
  ports:
```

```
    - port: 80
```

```
      targetPort: 80
```


Spécification : exemple de type ClusterIP

Chaque requête reçue par le service est envoyée sur l'un des Pods ayant le label spécifié

```
$ cat vote-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: vote
  labels:
    app: vote
spec:
  containers:
  - name: vote
    image: instavote/vote
    ports:
    - containerPort: 80
```

```
$ cat vote-service-clusterIP.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Spécification : exemple de type ClusterIP


```
# Lancement du Pod vote
$ kubectl create -f vote-pod.yaml

# Lancement du Service de type ClusterIP
$ kubectl create -f vote-service-clusterIP.yaml

# Lancement d'un Pod utilisé pour le debug
$ kubectl create -f pod-debug.yaml

# Accès au Service vote depuis le Pod debug
$ kubectl exec -ti debug sh
/ # apk update && apk add curl
/ # curl http://vote-service
(code html de l'interface vote)
...

# Résolution DNS via SERVICE_NAME.NAMESPACE
/ # curl http://vote-service.default
...
```



```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
  - name: debug
    image: alpine
    command:
    - "sleep"
    - "10000"
```

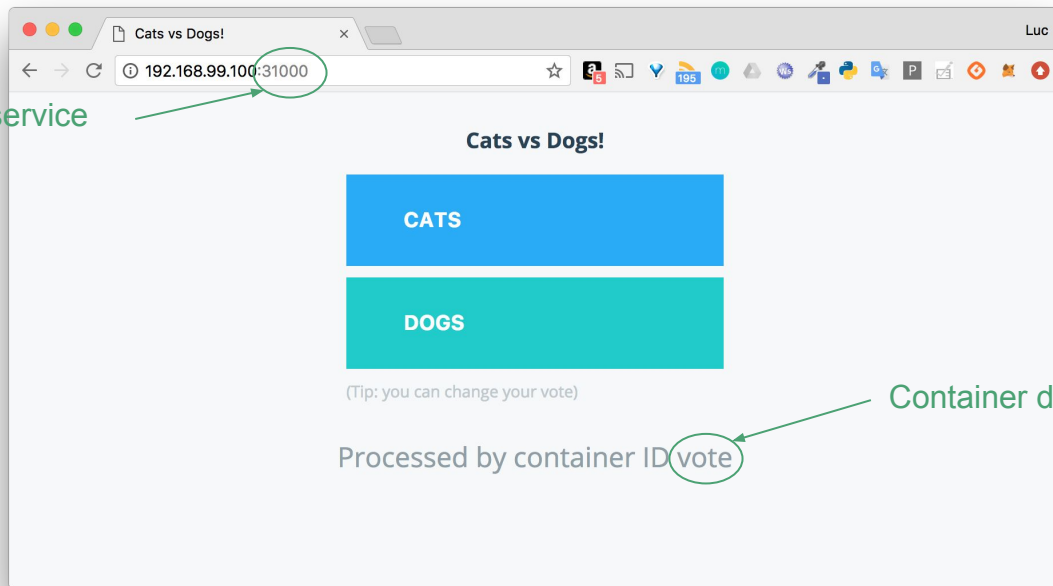
Spécification : exemple de type NodePort

```
$ cat vote-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-service
spec:
  selector:
    app: vote
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 31000
```

Spécification : exemple de type NodePort

```
# Lancement du Service  
$ kubectl create -f vote-service.yaml
```

Port exposé par le service



Container défini dans le Pod

Deployment

Spécification : exemple

```
$ cat vote-deployment.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: vote-deploy
```

```
spec:
```

```
  replicas: 3
```

```
  selector:
```

```
    matchLabels:
```

```
      app: vote
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: vote
```

```
    spec:
```

```
      containers:
```

```
        - name: vote
```

```
          image: instavote/vote
```

```
          ports:
```

```
            - containerPort: 80
```

Deployment

ReplicaSet

Pod

Spécification : exemple

```
# Lancement du Deployment
$ kubectl create -f vote-deployment.yaml
deployment "vote-deploy" created
```

```
# Liste des Deployments
```

```
$ kubectl get deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
vote-deploy	3	3	3	3	31s

```
# Liste des ReplicaSet
```

```
$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
vote-deploy-584c4c76db	3	3	3	34s


```
# Liste des Pods
```

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
vote-deploy-584c4c76db-65r7r	1/1	Running	0	36s
vote-deploy-584c4c76db-gl9ps	1/1	Running	0	36s
vote-deploy-584c4c76db-s7gdn	1/1	Running	0	36s



Un ReplicaSet est créé et associé au Deployment



Le ReplicaSet gère les 3 Pods (replicas) définis dans la spécification du Deployment

Secret

Secret de type generic : création

```
# Création de fichiers contenant les credentials de connexion à un service tiers
$ echo -n "admin" > ./username.txt
$ echo -n "45fe3efa" > ./password.txt

# Création de l'objet Secret avec kubectl
$ kubectl create secret generic service-creds --from-file=./username.txt --from-file=./password.txt
secret "service-creds" created

# Création depuis des valeurs littérales
$ kubectl create secret generic service-creds2 \
  --from-literal=username=admin --from-literal=password=45fe3efa
secret "service-creds2" created

# Liste des Secrets présents
$ kubectl get secrets
```

NAME	TYPE	DATA	AGE
default-token-ps46p	kubernetes.io/service-account-token	3	82d
service-creds	Opaque	2	3s
service-creds2	Opaque	2	3m

Secret généré par kubernetes pour permettre aux services internes d'accéder à l'API server

Secret de type generic : création

```
# Inspection du contenu de l'objet
$ kubectl describe secrets/service-creds
Name:          service-creds
Namespace:     default
Labels:        <none>
Annotations:   <none>

Type: Opaque

Data
====
password.txt:  8 bytes
username.txt:  5 bytes
```

```
# Détails de la spécification de l'objet
$ kubectl get secrets service-creds -o yaml
apiVersion: v1
data:
  password.txt: NDVmZTNlZmE=
  username.txt: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2018-02-28T22:37:47Z
  name: service-creds
  namespace: default
  resourceVersion: "589455"
  selfLink: /api/v1/namespaces/default/secrets/service-creds
  uid: ffd1b396-1cd7-11e8-ba7f-080027f0e385
type: Opaque
```

Encodé en base64

```
$ echo "NDVmZTNlZmE=" | base64 -D
45fe3efa
$ echo "YWRtaW4=" | base64 -D
admin
```

Secret de type generic : création

```
# Conversion de la donnée sensible en base64
$ echo -n "mongodb://admin:45fe3efa@mgserv1.org/mgmt" | base64
bW9uZ29kYjovL2FkbWlu0jQ1ZmUzZWZhQG1nc2VydjEub3JnL21nbXQ=

# Spécification de l'objet Secret
$ cat mongo-creds.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongo-creds
data:
  mongoURL: bW9uZ29kYjovL2FkbWlu0jQ1ZmUzZWZhQG1nc2VydjEub3JnL21nbXQ=

# Création de l'objet Secret
$ kubectl create -f ./mongo-creds.yaml
secret "mongo-creds" created
```

Secret de type generic : utilisation (volume 1)

```
$ cat pod-secret-volume-1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine
    command:
    - "sleep"
    - "10000"
    volumeMounts:
    - name: creds
      mountPath: "/etc/creds"
      readOnly: true
  volumes:
  - name: creds
    secret:
      secretName: mongo-creds
```

Montage du volume dans le container sur le point de montage spécifié



Définition d'un volume basé sur le Secret mongo-creds

Secret de type generic : utilisation (volume 1)

```
# Création du Pod alpine
$ kubectl create -f pod-secret-volume-1.yaml
pod "alpine" created

# Lancement d'un shell interactif dans le container alpine
$ kubectl exec -ti alpine -- sh
/ # cat /etc/creds/mongoURL
mongodb://admin:45fe3efa@mgserv1.org/mgmt
```

Secret de type generic : utilisation (volume 2)

```
$ cat pod-secret-volume-2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine
    command:
    - "sleep"
    - "10000"
    volumeMounts:
    - name: creds
      mountPath: "/etc/creds"
      readOnly: true
  volumes:
  - name: creds
    secret:
      secretName: service-creds
      items:
      - key: username.txt
        path: service/user
      - key: password.txt
        path: service/pass
```

Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur le Secret service-creds
On précise les path relatifs ou les clés seront montées



Secret de type generic : utilisation (volume 2)

```
# Création du Pod alpine
$ kubectl create -f pod-secret-volume-2.yaml
pod "alpine" created

# Lancement d'un shell interactif dans le container api
$ kubectl exec -ti alpine -- sh
/ # cat /etc/creds/service/user
admin
/ # cat /etc/creds/service/pass
45fe3efa
```

Secret de type generic : utilisation (env)

```
$ cat pod-secret-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine
    command:
    - "sleep"
    - "10000"
    env:
    - name: MONGO_URL
      valueFrom:
        secretKeyRef:
          name: mongo-creds
          key: mongoURL
```


Secret de type generic : utilisation (env)

```
# Création du Pod api
$ kubectl create -f pod-secret-env.yaml
pod "api" created

# Lancement d'un shell interactif dans le container api
$ kubectl exec -ti alpine -- sh
/ # env | grep MONGO
MONGO_URL=mongodb://admin:45fe3efa@mgserv1.org/mgmt
```

Secret de type docker-registry : création

```
$ kubectl create secret docker-registry registry-creds \  
  --docker-server=REGISTRY_FQDN --docker-username=USERNAME --docker-password=PASSWORD --docker-email=EMAIL  
  
$ kubectl get secret registry-creds -o yaml  
apiVersion: v1  
data:  
  .dockercfg: eyJhdXRocyI6eyJodHR...QVwtRPSJ9fX0=  
kind: Secret  
metadata:  
  creationTimestamp: 2018-03-02T22:26:14Z  
  name: registry-creds  
  namespace: default  
  resourceVersion: "626790"  
  selfLink: /api/v1/namespaces/default/secrets/registry-creds  
  uid: b7b70613-1e68-11e8-ba7f-080027f0e385  
type: kubernetes.io/dockercfg
```

Secret de type docker-registry : utilisation

```
# Pod utilisant une image privée
$ cat pod-private-image.yaml
apiVersion: v1
kind: Pod
metadata:
  name: private-image
spec:
  containers:
  - name: api
    image: my_private_image
    imagePullSecrets:
    - name: registry-creds

$ kubectl create -f pod-private-reg.yaml
pod "private-reg" created
```

Secret de type TLS : création

- Gestion des PKI
- Créé à partir d'un couple clé publique / clé privée

```
# Création du couple de clés
$ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out cert.pem

# Création du Secret à partir des clés
$ kubectl create secret tls domain-pki --cert cert.pem --key key.pem

$ kubectl get secret domain-pki -o yaml
apiVersion: v1
data:
  tls.crt: LS0tLS1CRUdJ...GSUNBVEUtLS0tLQo=
  tls.key: LS0tLS1CRUdJ...TESBLRVktLS0tLQo=
kind: Secret
Metadata:
  name: domain-pki
  ...
type: kubernetes.io/tls
```

Secret de type TLS : utilisation

```
$ cat pod-secret-tls.yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: proxy
```

```
spec:
```

```
  containers:
```

```
  - name: proxy
```

```
    image: nginx:1.12.2
```

```
    volumeMounts:
```

```
    - name: tls
```

```
      mountPath: "/etc/ssl/certs/"
```

```
  volumes:
```

```
  - name: tls
```

```
    secret:
```

```
      secretName: domain-pki
```



Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur le Secret domain-pki

Secret de type TLS : utilisation

```
# Lancement du Pod
$ kubectl create -f pod-secret-tls.yaml
pod "proxy" created

# Lancement d'un shell interactif dans le container
$ kubectl exec -ti proxy -- sh
/ # ls /etc/ssl/certs
tls.crt  tls.key
```

ConfigMap

Création à partir d'un fichier

```
$ cat nginx.conf
user www-data;
worker_processes 4;
pid /run/nginx.pid;
events {
    worker_connections 768;
}
http {
    server {
        listen *:8000;
        location / {
            proxy_pass http://localhost;
        }
    }
}
```



```
$ kubectl create configmap nginx-config --from-file=./nginx.conf
configmap "nginx-config" created
```


Création à partir d'un fichier

```
$ kubectl get cm nginx-config -o yaml
apiVersion: v1
data:
  nginx.conf: |
    user www-data;
    worker_processes 4;
    ...
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-03T15:35:46Z
  name: nginx-config
  namespace: default
  resourceVersion: "635910"
  selfLink: /api/v1/namespaces/default/configmaps/nginx-config
  uid: 8ac176fc-1ef8-11e8-ba7f-080027f0e385
```

Création à partir d'un fichier d'environnement

```
# Fichier d'environnement constitué de couples key=value
$ cat app.env
log_level=WARN
env=production
cache=redis

# Création d'une ConfigMap à partir du fichier .env
$ kubectl create configmap app-config-env --from-env-file=./app.env
configmap "app-config-env" created

$ kubectl get cm app-config-env -o yaml
apiVersion: v1
data:
  cache: redis
  env: production
  log_level: WARN
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-04T14:45:09Z
  name: app-config-env
  namespace: default
  ...
```

Création à partir de valeurs littérales

```
# Utilisation de l'option --from-literal pour chaque couple clé=valeur
$ kubectl create configmap app-config-lit \
  --from-literal=log_level=WARM \
  --from-literal=env=production \
  --from-literal=cache=redis
configmap "app-config-lit" created

$ kubectl get cm app-config-lit -o yaml
apiVersion: v1
data:
  cache: redis
  env: production
  log_level: WARM
kind: ConfigMap
metadata:
  creationTimestamp: 2018-03-04T14:49:51Z
  name: app-config-lit
  namespace: default
  ...
```

Utilisation dans un Pod : volume

```
$ cat pod-config-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: proxy
    image: nginx:1.12.2
    ports:
    - containerPort: 8000
    volumeMounts:
    - name: config
      mountPath: "/etc/nginx/"
  - name: api
    image: lucj/city:1.0
    ports:
    - containerPort: 80
  volumes:
  - name: config
    configMap:
      name: nginx-config
```

Montage du volume dans le container sur le point de montage spécifié

Définition d'un volume basé sur la ConfigMap nginx-config



Utilisation dans un Pod : volume

```
# Création du Pod www
$ kubectl create -f pod-config-volume.yaml
pod "www" created

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti www --container proxy -- bash
root@www:/# cat /etc/nginx/nginx.conf
user www-data;
worker_processes 4;
pid /run/nginx.pid;
events {
    worker_connections 768;
}
http {
    server {
        listen *:8000;
        location / {
            proxy_pass http://localhost;
        }
    }
}
```

Utilisation dans un Pod : volume

```
# Lancement d'un shell interactif dans le container api du Pod www
```

```
$ kubectl exec -ti www --container api -- sh
```

```
/app # apk update && apk add curl
```

```
/app # curl localhost
```

```
{"message":"www suggests to visit Robjazzpaw"}
```

```
# Lancement d'un shell interactif dans le container proxy du Pod www
```

```
$ kubectl exec -ti www --container proxy -- bash
```

```
root@www:/# apt-get update && apt-get install -y curl
```

```
root@www:/# curl localhost:8000
```

```
{"message":"www suggests to visit Tubogbaj"}
```

Utilisation dans un Pod : variable d'environnement

```
$ cat pod-config-env.yaml
apiVersion: v1
kind: Pod
metadata:
  name: w3
spec:
  containers:
  - name: www
    image: nginx:1.12.2
    env:
    - name: LOG_LEVEL
      valueFrom:
        configMapKeyRef:
          name: app_config-lit
          key: log_level
    - name: CACHE
      valueFrom:
        configMapKeyRef:
          name: app-config-env
          key: cache
```

Utilisation dans un Pod : variable d'environnement

```
# Création du Pod www
$ kubectl create -f pod-config-env.yaml
pod "www" created

# Lancement d'un shell interactif dans le container proxy du Pod www
$ kubectl exec -ti w3 --container www -- bash
# / env
HOSTNAME=w3
NJS_VERSION=1.12.2.0.1.14-1~stretch
NGINX_VERSION=1.12.2-1~stretch
CACHE=production
LOG_LEVEL=WARM
...
```


Namespace

Création

```
# Création du namespaces development (option 1)
```

```
$ kubectl create namespace development
```

```
namespace "development" created
```

```
# Suppression du namespace
```

```
$ kubectl delete namespace/development
```

```
namespace "development" deleted
```

```
# Création du namespace development (option 2)
```

```
$ cat development.yaml
```

```
{  
  "kind": "Namespace",  
  "apiVersion": "v1",  
  "metadata": {  
    "name": "development",  
    "labels": {  
      "name": "development"  
    }  
  }  
}
```

```
$ kubectl create -f development.yaml
```

```
namespace "development" created
```

Utilisation

- Pod avec namespace spécifié dans les metadata

```
$ cat nginx-pod-dev.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: development
spec:
  containers:
  - name: www
    image: nginx:1.12.2
```

Utilisation

```
# Lancement d'un Pod dans le namespace development
$ kubectl create -f nginx-pod-dev.yaml
pod "nginx" created
```

```
# Liste des Pods dans le namespace default
$ kubectl get po
No resources found.
```

```
# Liste des Pods dans le namespace development
$ kubectl get po --namespace=development
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           17s
```

```
# Liste des Pods dans l'ensemble des namespaces
$ kubectl get po --all-namespaces
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           17s
...
```

Utilisation

```
# Création d'un Deployment dans le namespace development
$ kubectl run www --namespace development --replicas 2 --image nginx:1.12.2
```

```
# Liste des Deployments dans le namespace development
```

```
$ kubectl get deploy --namespace development
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
www	2	2	2	2	20s

```
# Liste des Pods dans le namespace development
```

```
$ kubectl get po --namespace development
```

NAME	READY	STATUS	RESTARTS	AGE
www-6b5dfc4699-8zk92	1/1	Running	0	30s
www-6b5dfc4699-xj5cg	1/1	Running	0	30s

Ingress

Ingress : routage par nom de domaine

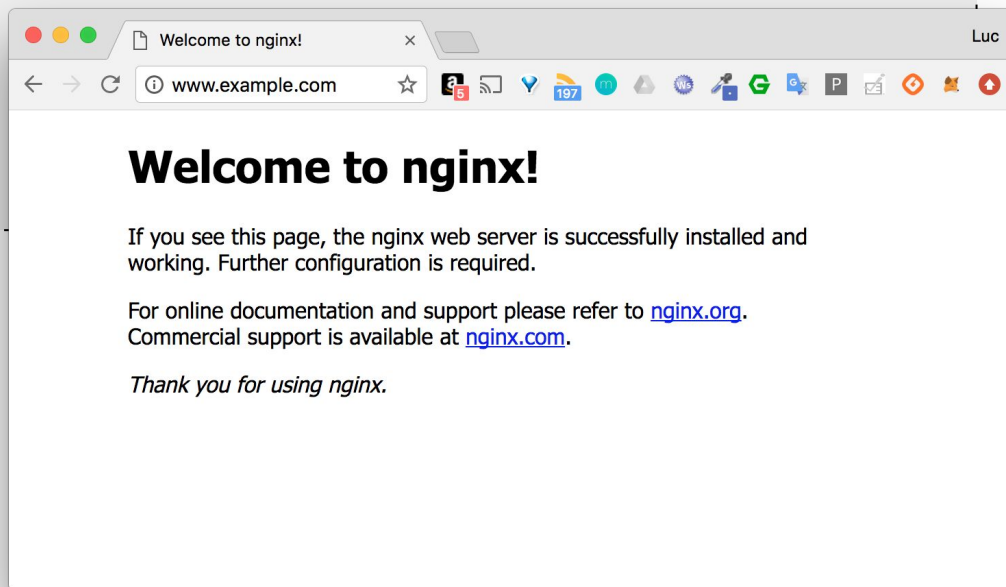
```
$ cat www-ingress-domain.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: www-domain
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
          serviceName: www
          servicePort: 80
```

Ingress : routage par nom de domaine

```
# Création d'un Deployment basé sur nginx
$ kubectl run www --image=nginx:1.12.2
deployment "www" created

# Exposition du Deployment via un Service
$ kubectl expose deployment www --port=80 --target-port=80
service "www" exposed

# Création de l'objet Ingress
$ kubectl create -f www-ingress-domain.yaml
ingress "www-domain" created
```



Ingress : routage via le path de la requête

```
$ cat www-ingress-path.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: www-path
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /www
        backend:
          serviceName: w3
          servicePort: 80
```

Ingress : routage via le path de la requête

```
# Création d'un Deployment basé sur nginx
$ kubectl run w3 --image=nginx:1.12.2
deployment "w3" created

# Exposition du Deployment via un Service
$ kubectl expose deployment w3 --port=80 --target-port=80
service "w3" exposed

# Création de l'objet Ingress
$ kubectl create -f www-ingress-domain.yaml
ingress "www-domain" created
```

