Version - **2025.01**

Last update : 2025/01/19 14:20

DOE305 - Network, Service and Microservices Architecture Management

Curriculum

LAB #1 - Network and Service Management

1.1 - Overview of Network Extensions

Kubernetes imposes certain conditions on the implementation of a network:

- PODs on one node can communicate with all PODs on all nodes without using NAT,
- Agents on a node (e.g. kubelet) can communicate with all PODs on the node.

> **Important**: A detailed technical description of the Kubernetes networking approach can be found at : **https://kubernetes.io/docs/concepts/cluster-administration/networking/**.

When installing the cluster, we specified the use of a network extension called **Calico**, taken from the following list:

- **Calico**,
- **Cilium**,
- **Flannel**,
- **Kube-router**,
- **Romana**,
- **WeaveNet**,
- **Antrea**,
- **kube-ovn**,

- Channel (uses Flannel for network and Calico for firewall).

> **Important**: A comparative study of network extensions for Kubernetes can be found at : **https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-10gbit-s-network-updated-august-2020-6e1b757b9e49**.

1.2 - DNS K8s

Overview

DNS services for the cluster using the **Calico** plugin are provided by **CoreDNS** :

```
root@kubemaster:~# kubectl get deployments -n kube-system
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
calico-kube-controllers   1/1     1            1           12d
coredns                   2/2     2            2           12d
metrics-server            1/1     1            1           11d

root@kubemaster:~# kubectl get service -n kube-system
NAME             TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                  AGE
kube-dns         ClusterIP   10.96.0.10    <none>        53/UDP,53/TCP,9153/TCP   12d
metrics-server   ClusterIP   10.98.89.81   <none>        443/TCP                  11d
```

All pods are assigned a host name in the following format:

```
pod_ip_address_formated_as_xxx-xxx-xxx-xxx.namespace.pod.cluster.local
```

Implementation

To test the DNS, create the file **dnstest.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi dnstest.yaml
root@kubemaster:~# cat dnstest.yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox-dnstest
spec:
  containers:
    - name: busybox
      image: radial/busyboxplus:curl
      command: ['sh', '-c', 'while true; do sleep 3600; done']
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-dnstest
spec:
  containers:
  - name: nginx
    image: nginx:1.19.2
    ports:
    - containerPort: 80
```

Important: Note that this file will create two pods - **busybox-dnstest** and **nginx-dnstest**.

Create the two pods using the file:

```
root@kubemaster:~# kubectl create -f dnstest.yaml
pod/busybox-dnstest created
pod/nginx-dnstest created
```

Copy the IP address of the **nginx-test** pod:

```
root@kubemaster:~# kubectl get pods nginx-dnstest -o wide
NAME            READY   STATUS    RESTARTS   AGE   IP               NODE                         NOMINATED NODE
READINESS GATES
nginx-dnstest   1/1     Running   0          48s   192.168.150.33   kubenode2.ittraining.loc   <none>
<none>
```

Run the **curl <copied IP address>** command in the **busybox-dnstest** container:

```
root@kubemaster:~# kubectl exec busybox-dnstest -- curl 192.168.150.33
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

```
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
100   612  100   612    0     0   533k      0 --:--:-- --:--:-- --:--:--  597k
```

> **Important**: Note that **busybox-dnstest** was able to contact **nginx-dnstest** using its IP address.

Now use K8s DNS to resolve the **nginx-dnstest** pod hostname:

```
root@kubemaster:~# kubectl exec busybox-dnstest -- nslookup 192-168-150-33.default.pod.cluster.local
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      192-168-150-33.default.pod.cluster.local
Address 1: 192.168.150.33
```

> **Important**: Note that the host name has been resolved using K8s DNS.

Now run the **curl <hostname_of_pod_nginx_dnstest>** command in the **busybox-dnstest** container:

```
root@kubemaster:~# kubectl exec busybox-dnstest -- curl 192-168-150-33.default.pod.cluster.local
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
```

```
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
...
<title>Welcome to nginx!</title>
...
100   612  100   612    0     0   355k      0 --:--:-- --:--:-- --:--:--  597k
```

Important: Note that **busybox-dnstest** was able to contact **nginx-dnstest** using its host name.

1.3 - Network Policies

Overview

A **NetworkPolicy** is a K8s object that controls communication to and from pods.

The components of a NetworkPolicy are :

- **from and to Selectors**,
  - the **from selector** operates on **Ingress** traffic,
    - the word Ingress indicates network traffic to a pod,
  - the **to selector** operates on **Egress** traffic,
    - Egress indicates traffic received from a pod.

From and to Selectors use **Types**:

- **podSelector**,
  - A podSelector can select pods using Labels,
  - by default, a pod is not isolated in the cluster. However, as soon as a podSelector selects a pod, it is considered isolated and can only communicate using **NetworkPolicies**,

- **namespaceSelector**,
  - a namespaceSelector can select nameSpaces using Labels,
- **ipBlock**,
  - an IPBlock can select pods using a range of IP addresses in CIDR format.

In addition to the above Types, it is also possible to specify :

- **Ports**,
  - ports specify the port number and protocol,
  - network traffic is only accepted if the rules specified by Type **and** the port/protocol are satisfied.

Implementation

To understand this better, create a NameSpace called **nptest** :

```
root@kubemaster:~# kubectl create namespace nptest
namespace/nptest created
```

Label this NameSpace :

```
root@kubemaster:~# kubectl label namespace nptest lab=nptest
namespace/nptest labeled
```

Important: Note the label **lab=nptest**.

Now create the **npnginx.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi npnginx.yaml
root@kubemaster:~# cat npnginx.yaml
apiVersion: v1
kind: Pod
metadata:
  name: npnginx
  namespace: nptest
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

Important: Note the **app: nginx** tag.

Create the npnginx pod:

```
root@kubemaster:~# kubectl create -f npnginx.yaml
pod/npnginx created
```

Now create the **npbusybox.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi npbusybox.yaml
root@kubemaster:~# cat npbusybox.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: npbusybox
  namespace: nptest
  labels:
    app: client
spec:
  containers:
    - name: busybox
      image: radial/busyboxplus:curl
      command: ['sh', '-c', 'while true; do sleep 5; done']
```

Important: Note the **app: client** label.

Create the **npbusybox** pod:

```
root@kubemaster:~# kubectl create -f npbusybox.yaml
pod/npbusybox created
```

View the information on the two pods created:

```
root@kubemaster:~# kubectl get pods -n nptest -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP               NODE                        NOMINATED NODE
READINESS GATES
npbusybox   1/1     Running   0          48s     192.168.150.35   kubenode2.ittraining.loc   <none>
<none>
npnginx     1/1     Running   0          4m13s   192.168.239.33   kubenode1.ittraining.loc   <none>
<none>
```

Copy the IP address of the **npnginx** node and create a variable called **NGINX_IP** :

```
root@kubemaster:~# NGINX_IP=192.168.239.33

root@kubemaster:~# echo $NGINX_IP
192.168.239.33
```

Test the communication between **npbusybox** and **npnginx** :

```
root@kubemaster:~# kubectl exec -n nptest npbusybox -- curl $NGINX_IP
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   615  100   615    0     0  78977      0 --:--:-- --:--:-- --:--:-- 87857
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Important: Remember: by default, a pod is not isolated in the cluster. The communication

was therefore successful.

Now create the **mynetworkpolicy.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi mynetworkpolicy.yaml
root@kubemaster:~# cat mynetworkpolicy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: mynetworkpolicy
  namespace: nptest
spec:
  podSelector:
    matchLabels:
      app: nginx
  policyTypes:
    - Ingress
    - Egress
```

Important: Note the **app: nginx** tag. The policy therefore applies to the **npnginx** pod.

Now create the NetworkPolicy :

```
root@kubemaster:~# kubectl create -f mynetworkpolicy.yaml
```

```
networkpolicy.networking.k8s.io/mynetworkpolicy created
```

Test the communication between **npbusybox** and **npnginx** again:

```
root@kubemaster:~# kubectl exec -n nptest npbusybox -- curl $NGINX_IP
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:24 --:--:--     0^C
```

> **Important**: Note that NetworkPolicy blocks communication. Also note the use of **^C** to terminate the process.

Now edit the NetworkPolicy:

```
root@kubemaster:~# kubectl edit networkpolicy -n nptest mynetworkpolicy

# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  creationTimestamp: "2022-09-16T13:24:29Z"
  generation: 1
  name: mynetworkpolicy
  namespace: nptest
  resourceVersion: "1490105"
  uid: b130f09f-2ab1-4dc6-9059-95f900234be3
spec:
  podSelector:
```

```
    matchLabels:
      app: nginx
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          lab: nptest
    ports:
    - protocol: TCP
      port: 80
status: {}
:wq

root@kubemaster:~# kubectl edit networkpolicy -n nptest mynetworkpolicy
networkpolicy.networking.k8s.io/mynetworkpolicy edited
```

Important: Note the creation of the **ingress** rule. This rule uses a namespaceSelector to allow traffic from pods in a NameSpace with a **lab: nptest** label. The ports rule allows traffic on port 80/tcp.

Test communication between **npbusybox** and **npnginx** again:

```
root@kubemaster:~# kubectl exec -n nptest npbusybox -- curl $NGINX_IP
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   615  100   615    0     0   531k      0 --:--:-- --:--:-- --:--:--  600k
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Important: Note that the communication was successful.

1.4 - Services

Overview

K8s services are:

- NodePort,
  - This service makes a POD accessible on a port of the node containing it,
- ClusterIP

    - This service creates a virtual IP address to enable communication between different services in the cluster, e.g. front-end servers with back-end servers,
- LoadBalancer
    - This service provides load balancing for an application in certain public Cloud providers such as **A**mazon **W**eb **S**ervices and **G**oogle **C**loud **P**latform.
- ExternalName
    - Not part of CKA certification.

Implementation

Start by creating the **myapp-deployment** :

```
root@kubemaster:~# kubectl create -f deployment-definition.yaml
deployment.apps/myapp-deployment created
```

Check the status of the pods:

```
root@kubemaster:~# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS        AGE    IP               NODE
NOMINATED NODE   READINESS GATES
busybox-dnstest                     1/1     Running   0               4h9m   192.168.150.34
kubenode2.ittraining.loc   <none>           <none>
myapp-deployment-7c4d4f7fc6-2km9n   1/1     Running   0               83s    192.168.239.34
kubenode1.ittraining.loc   <none>           <none>
myapp-deployment-7c4d4f7fc6-7pts7   1/1     Running   0               83s    192.168.239.35
kubenode1.ittraining.loc   <none>           <none>
myapp-deployment-7c4d4f7fc6-9pw5x   1/1     Running   0               83s    192.168.150.36
kubenode2.ittraining.loc   <none>           <none>
mydaemonset-hmdhp                   1/1     Running   1 (7h29m ago)   23h    192.168.239.32
kubenode1.ittraining.loc   <none>           <none>
mydaemonset-kmf4z                   1/1     Running   1               23h    192.168.150.31
kubenode2.ittraining.loc   <none>           <none>
nginx-dnstest                       1/1     Running   0               4h9m   192.168.150.33
```

```
kubenode2.ittraining.loc   <none>           <none>
```

> **Important**: Note that the **192.168.239.x** addresses are associated with PODs on kubenode1, while the **192.168.150.x** addresses are associated with PODs on kubenode2. These addresses come from the **192.168.0.0/16** network stipulated by the **–pod-network-cidr** option during controller initialization.

Knowing that a Nginx container exists in each POD, test whether you can display the Nginx home page by connecting to kubenode1 and kubenode2 from your Gateway:

```
trainee@kubemaster:~$ exit
déconnexion
Connection to 10.0.2.65 closed.
trainee@gateway:~$ curl 192.168.56.3
curl: (7) Failed to connect to 192.168.56.3 port 80: Connection refused
trainee@gateway:~$ curl 192.168.56.4
curl: (7) Failed to connect to 192.168.56.4 port 80: Connection refused
```

> **Important**: Note the connection failure.

Now test whether you can display the Nginx home page by connecting to one of the PODs **from your Gateway** :

```
trainee@gateway:~$ curl 192.168.239.34
^C
```

Connect to **kubemaster** :

```
trainee@gateway:~$ ssh -l trainee 192.168.56.2
```

```
trainee@192.168.56.2's password: trainee
Linux kubemaster.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jul 13 15:45:46 2022 from 10.0.2.40
trainee@kubemaster:~$ su -
Password: fenestros
root@kubemaster:~#
```

Of course, it is possible to display the page by connecting to one of the PODs **inside** the cluster:

```
root@kubemaster:~# curl 192.168.239.34
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

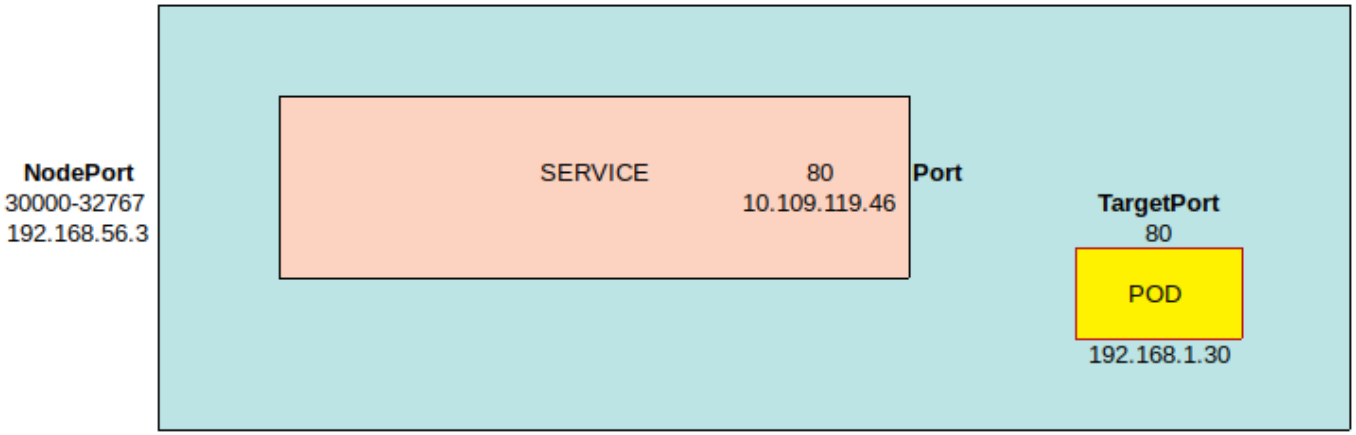
```
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

> **Important**: Note that at this stage, it is not possible to display the Nginx home page when connecting from outside the cluster.

The NodePort Service

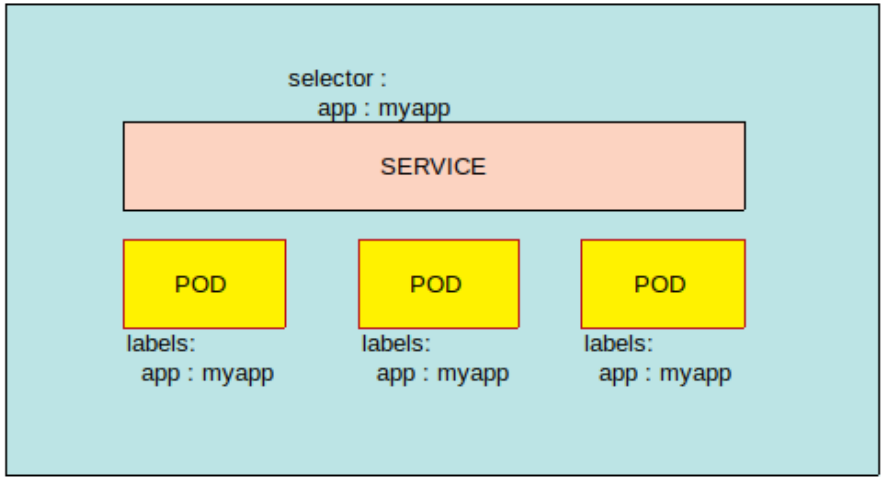
The NodePort Service defines three ports:

- **TargetPort**: the port on the POD,
- **Port**: the port on the Service linked to a Cluster IP,
- **NodePort** : the port on the Node from the range 30000-32767.



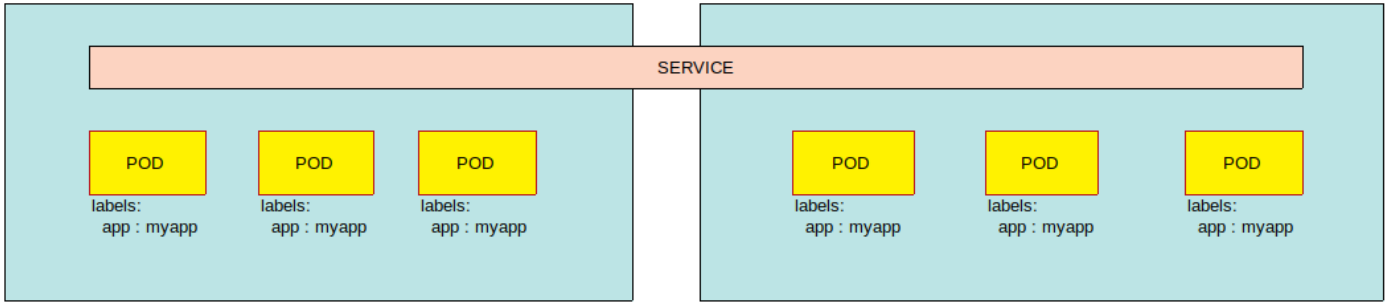
}

If several PODs in the same node have labels that match the Service's **selector**, the Service identifies the PODs and automatically expands to include all PODs. PODs are called **End-Points**:



Important: Note that in this case, load balancing is automatic and uses the **Random** algorithm with a session affinity...

Similarly, when PODs are distributed across several nodes, the Service extends to encompass all of them:



Create the YAML file **service-definition.yaml** :

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi service-definition.yaml
root@kubemaster:~# cat service-definition.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service

spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

Important: Note that if the **type:** field is missing, its default value is **ClusterIP**. Also note that in **ports**, only the **port** field is mandatory. If the **targetPort** field is missing, its default value is that of the **port** field. If the **nodePort** field is missing, its default value is the first available port in the range **30,000** to **32,767**. Finally, it is possible to specify multiple port definitions in the service.

The **selector** field contains the labels of the PODs concerned by the Service setup:

```
root@kubemaster:~# cat pod-definition.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Create the Service using the **service-definition.yaml** file:

```
root@kubemaster:~# kubectl create -f service-definition.yaml
service/myapp-service created
```

Note that the service has been created:

```
root@kubemaster:~# kubectl get services
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP        26h
myapp-service   NodePort    10.97.228.14   <none>        80:30008/TCP   13s
```

Important: Note that the Service has a cluster IP address and has exposed port **30,008**.

Now test whether you can display the Nginx home page by connecting to one of the PODs from your Gateway using the exposed port:

```
root@kubemaster:~# exit
déconnexion
```

```
trainee@kubemaster:~$ exit
déconnexion
Connection to 192.168.56.2 closed.

trainee@gateway:~$ curl 192.168.56.3:30008
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

trainee@gateway:~$ curl 192.168.56.4:30008
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

The ClusterIP Service

The **ClusterIP** service groups together PODs offering the same service to facilitate communication between pods within the cluster.

To create a ClusterIP Service, create the file **clusterip-example.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi clusterip-example.yaml
root@kubemaster:~# cat clusterip-example.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentclusterip
spec:
  replicas: 3
  selector:
    matchLabels:
      app: clusteripexample
  template:
    metadata:
      labels:
        app: clusteripexample
    spec:
      containers:
      - name: nginx
        image: nginx:1.19.1
        ports:
        - containerPort: 80
```

Create a deployment using the **clusterip-example.yaml** file:

```
root@kubemaster:~# kubectl create -f clusterip-example.yaml
deployment.apps/deploymentclusterip created
```

Now create a ClusterIP service to expose the pods in the **deploymentclusterip** deplyment:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi clusterip-service.yaml
root@kubemaster:~# cat clusterip-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: clusteripservice
spec:
  type: ClusterIP
  selector:
    app: clusteripexample
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Create a service using the **clusterip-service.yaml** file:

```
root@kubemaster:~# kubectl create -f clusterip-service.yaml
service/clusteripservice created

root@kubemaster:~# kubectl get services
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
clusteripservice   ClusterIP   10.109.80.217   <none>        80/TCP    5s
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP   12d
```

View the service's EndPoints using the following command:

```
root@kubemaster:~# kubectl get endpoints clusteripservice
NAME               ENDPOINTS                                               AGE
clusteripservice   192.168.150.39:80,192.168.150.40:80,192.168.239.38:80   114s
```

Now create a pod that will use the **clusteripservice** service:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi clusterippod.yaml
root@kubemaster:~# cat clusterippod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: clusterippod
spec:
  containers:
  - name: busybox
    image: radial/busyboxplus:curl
    command: ['sh', '-c', 'while true; do sleep 10; done']
```

Create the pod using the **clusterippod.yaml** file:

```
root@kubemaster:~# kubectl create -f clusterippod.yaml
pod/clusterippod created
```

Check that the **clusterippod** pod is running:

```
root@kubemaster:~# kubectl get pod clusterippod
NAME           READY   STATUS    RESTARTS   AGE
clusterippod   1/1     Running   0          2m28s
```

Check the **clusteripservice** inside the **clusterippod** pod:

```
root@kubemaster:~# kubectl exec clusterippod -- curl clusteripservice
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

```
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
100   612  100   612    0     0   6224      0 --:--:-- --:--:-- --:--:--  6652
```

1.5 - Services and the k8s DNS

Before continuing, clean up the cluster:

```
root@kubemaster:~# kubectl delete service myapp-service
service “myapp-service” deleted

root@kubemaster:~# kubectl delete deployment myapp-deployment
deployment.extensions “myapp-deployment” deleted

root@kubemaster:~# kubectl delete daemonset mydaemonset
daemonset.apps “mydaemonset” deleted
```

```
root@kubemaster:~# kubectl delete pods busybox-dnstest nginx-dnstest
pod “busybox-dnstest” deleted
pod “nginx-dnstest” deleted
```

Overview

Each K8s service is assigned a FQDN in the form :

```
service-name.namespace.svc.cluster-name-domain.example
```

Note that :

- The default **cluster-domain-name.example** is **cluster.local**.
- The FQDN can be used to reach a service from any NameSpace.
- Pods in the same NameSpace as the service can reach it using its short name, i.e. **service-name**.

Implementation

View the **clusteripservice** service created earlier:

```
root@kubemaster:~# kubectl get service clusteripservice
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
clusteripservice   ClusterIP   10.109.80.217   <none>        80/TCP    12m
```

as well as the pods present in the cluster:

```
root@kubemaster:~# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
clusterippod                          1/1     Running   0          11m
deploymentclusterip-7776dc8d55-bmfjl   1/1     Running   0          15m
deploymentclusterip-7776dc8d55-pgmcg   1/1     Running   0          15m
```

```
deploymentclusterip-7776dc8d55-qvphh   1/1     Running   0          15m
```

View the FQDN of the **clusteripservice** using the **clusterippod** pod:

```
root@kubemaster:~# kubectl exec clusterippod -- nslookup 10.109.80.217
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: 10.109.80.217
Address 1: 10.109.80.217 clusteripservice.default.svc.cluster.local
```

> **Important**: Note that the FQDN of the service is **clusteripservice.default.svc.cluster.local**.

Check communication with the service using its IP address:

```
root@kubemaster:~# kubectl exec clusterippod -- curl 10.109.80.217
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   612  100   612    0     0  35322      0 --:--:--<!DOCTYPE html>:--     0
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
```

```
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
 --:--:-- --:--:-- 36000
```

Check the communication with the service using its short name:

```
root@kubemaster:~# kubectl exec clusterippod -- curl clusteripservice
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   612  100   612    0     0  81404      0 --:--:-- --:--:-- --:--:--  597k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

```
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

> **Important**: Note that the communication was successful because the **clusterippod** pod and the **clusteripservice** service are in the same namespace.

Verify the communication with the service using its FQDN:

```
root@kubemaster:~# kubectl exec clusterippod -- curl clusteripservice.default.svc.cluster.local
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   612  100   612    0     0   269k      0 --:--:-- --:--:-- --:--:--  597k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
```

```
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Now check the communication with the service using its short name from the **npbusybox** pod in the **nptest** namespace:

```
root@kubemaster:~# kubectl exec -n nptest npbusybox -- curl clusteripservice
curl: (6) Couldn't resolve host 'clusteripservice'
command terminated with exit code 6
```

Important: Note that the communication was unsuccessful because the **npbusybox** pod and the **clusteripservice** service are not in the same namespace.

Now check the communication with the service using its FQDN from the **npbusybox** pod in the **nptest** namespace:

```
root@kubemaster:~# kubectl exec -n nptest npbusybox -- curl clusteripservice.default.svc.cluster.local
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

```
100   612  100   612    0     0   291k      0 --:--:-- --:--:-- --:--:--  597k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Important: Note that the communication was successful thanks to the use of the service's FQDN.

1.6 - K8s Ingress management

Overview

An Ingress is a K8s object that manages access to services from outside the cluster. An Ingress is capable of more functionality than a simple NodePort service, for example:

- SSL,
- load balancing,
- name-based virtual hosts.

Ingress doesn't do anything on its own. It needs an **Ingress Controller** to function. Setting up and configuring an Ingress Controller is not part of the CKA certification.

Implementation

Start by creating the **myingress.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi myingress.yaml
root@kubemaster:~# cat myingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - http:
```

```
      paths:
      - path: /somepath
        pathType: Prefix
        backend:
          service:
            name: clusteripservice
            port:
              number: 80
```

> **Important**: Note that in this Ingress file we have a rule that defines a **path**. Requests that reference the path, for example http://<endpoint>/somepath, will be routed to the **backend**. In this example, the backend is a service, **clusteripservice**, listening on port **80**.

Now create the Ingress :

```
root@kubemaster:~# kubectl create -f myingress.yaml
ingress.networking.k8s.io/my-ingress created
```

Now consult Ingress:

```
root@kubemaster:~# kubectl describe ingress my-ingress
Name:             my-ingress
Labels:           <none>
Namespace:        default
Address:
Ingress Class:    <none>
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
```

```
  *
              /somepath   clusteripservice:80 (192.168.150.39:80,192.168.150.40:80,192.168.239.38:80)
Annotations:  <none>
Events:       <none>
```

Important: Note that the endpoints of the **clusteripservice** are displayed in the command output.

Now edit the **clusterip-service.yaml** file and add a **name** line in the **ports** section:

```
root@kubemaster:~# vi clusterip-service.yaml
root@kubemaster:~# cat clusterip-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: clusteripservice
spec:
  type: ClusterIP
  selector:
    app: clusteripexample
  ports:
    - name: myingress
      protocol: TCP
      port: 80
      targetPort: 80
```

Important: Note that the name can be any string.

Apply the clusteripservice modification:

```
root@kubemaster:~# kubectl apply -f clusterip-service.yaml
Warning: resource services/clusteripservice is missing the kubectl.kubernetes.io/last-applied-configuration
annotation which is required by kubectl apply. kubectl apply should only be used on resources created
declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched
automatically.
service/clusteripservice configured
```

Important: Note that the error is unimportant.

Now edit the **myingress.yaml** file and add a **name** line in the **ports** section, deleting the **number: 80** line:

```
root@kubemaster:~# cat myingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - http:
      paths:
      - path: /somepath
        pathType: Prefix
        backend:
          service:
            name: clusteripservice
            port:
              name: myingress
```

Apply the Ingress modification:

```
root@kubemaster:~# kubectl apply -f myingress.yaml
Warning: resource ingresses/my-ingress is missing the kubectl.kubernetes.io/last-applied-configuration annotation
which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by
either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
ingress.networking.k8s.io/my-ingress configured
```

Important: Note that the error is unimportant.

Now check the Ingress :

```
root@kubemaster:~# kubectl describe ingress my-ingress
Name:             my-ingress
Labels:           <none>
Namespace:        default
Address:
Ingress Class:    <none>
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /somepath   clusteripservice:myingress (192.168.150.39:80,192.168.150.40:80,192.168.239.38:80)
Annotations:  <none>
Events:       <none>
```

Important: Note that Ingress can still find the backend by using the name **myingress**.

LAB #2 - Managing a Microservices Architecture

Before continuing, clean up the cluster:

```
root@kubemaster:~# kubectl delete service clusteripservice
service “clusteripservice” deleted

root@kubemaster:~# kubectl delete deployment deploymentclusterip
deployment.apps “deploymentclusterip” deleted

root@kubemaster:~# kubectl delete ingress my-ingress
ingress.networking.k8s.io “my-ingress” deleted

root@kubemaster:~# kubectl delete pod clusterippod
pod “clusterippod” deleted
```

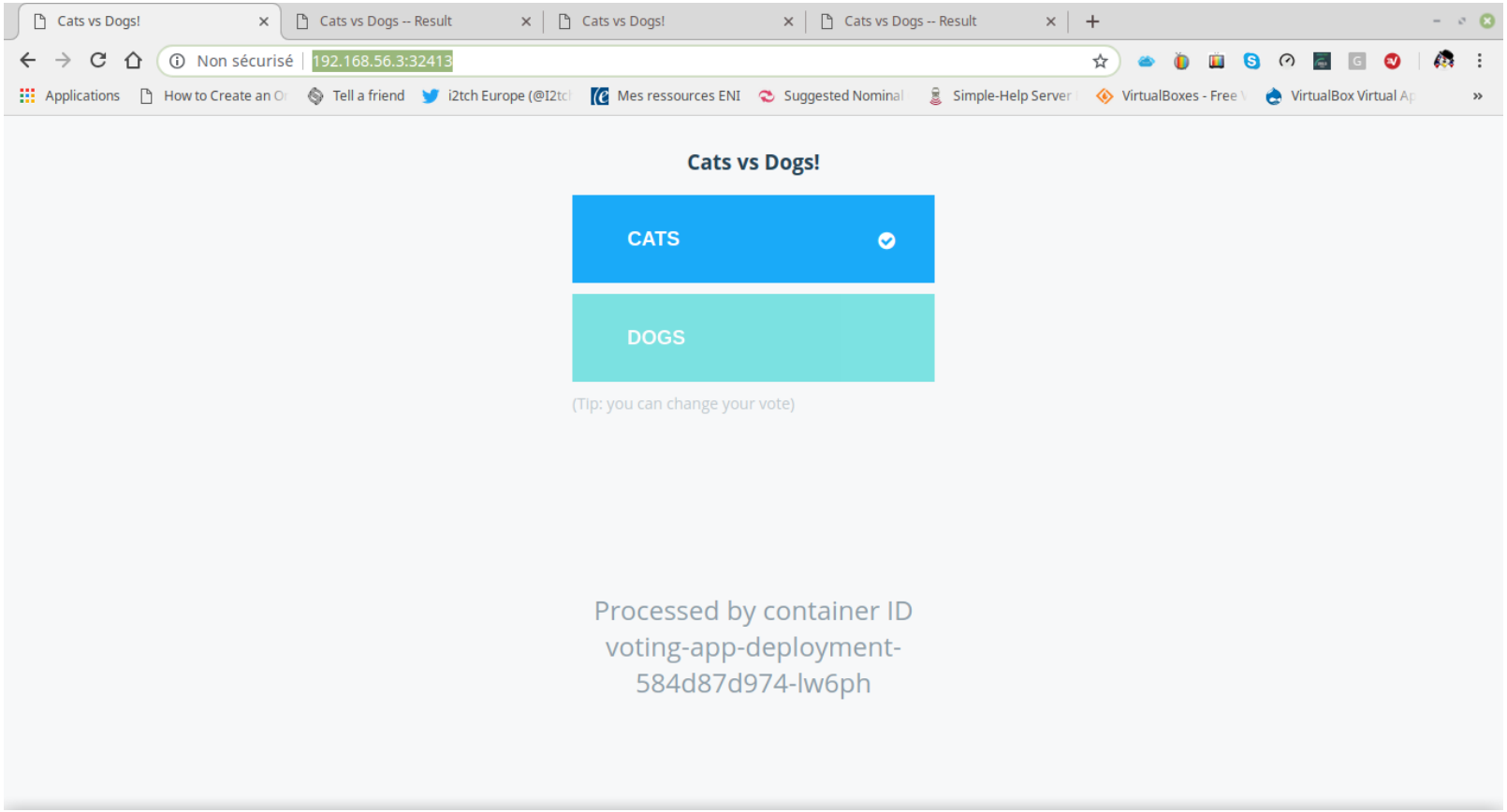
Check that only the default **kubernetes** service remains:

```
root@kubemaster:~# kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   13d
```

2.1 - Overview

You're going to set up a simple application in the form of microservices, developed by Docker, and called **demo-voting-app**:

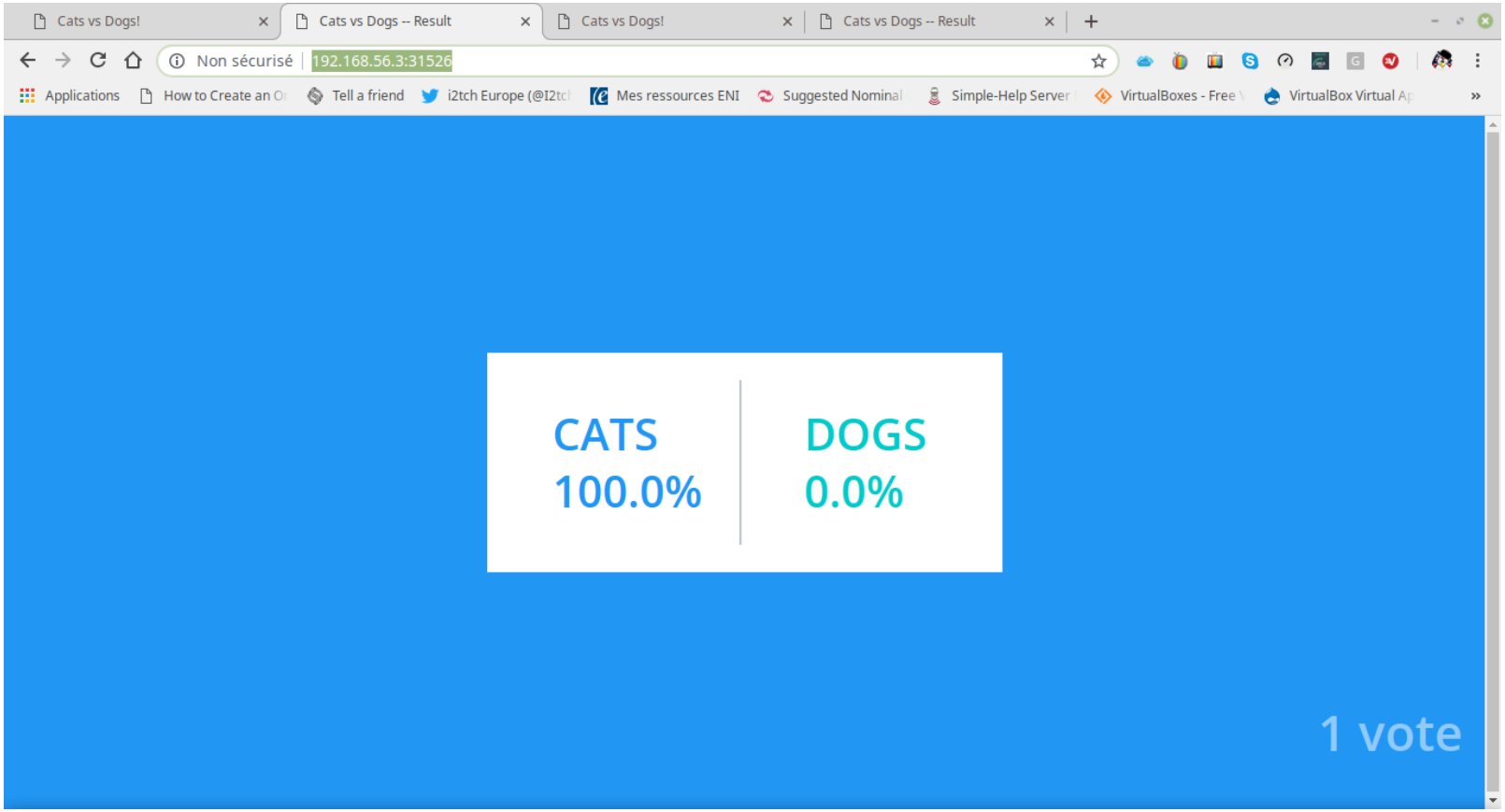
In this application, the **voting-app** container lets you vote for **cats** or **dogs**. This application runs under Python and provides an HTML interface:



}

When voting, the result is stored in **Redis** in an in-memory database. The result is then passed to the **Worker** container running under .NET, which updates the persistent database in the **db** container running under PostgreSQL.

The **result-app** application running under NodeJS then reads the table from the PostgreSQL database and displays the result in HTML form:



}

2.2 - Creating Deployments

Create the **myapp** directory. Go to this directory and create the file **voting-app-deployment.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# mkdir myapp

root@kubemaster:~# cd myapp

root@kubemaster:~/app# vi voting-app-deployment.yaml

root@kubemaster:~/app# cat voting-app-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: voting-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app-pod
        app: demo-voting-app

    spec:
      containers:
      - name: voting-app
        image: dockersamples/examplevotingapp_vote
        ports:
        - containerPort: 80
```

Important : This file describes a Deployment. Note that the Deployment creates **a** replica of the POD specified by **template** containing a container named **voting-app** which uses port 80 and is created from the image **dockersamples/examplevotingapp_vote**.

Now create the **redis-deployment.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi redis-deployment.yaml
root@kubemaster:~/app# cat redis-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: redis-pod
      app: demo-voting-app
  template:
    metadata:
      name: redis pod
      labels:
        name: redis-pod
```

```
        app: demo-voting-app

    spec:
      containers:
      - name: redis
        image: redis
        ports:
        - containerPort: 6379
```

> **Important** : This file describes a Deployment. Note that the Deployment creates **a** replica of the POD specified by **template** containing a container named **redis** which uses port 6379 and is created from the **redis** image.

Create the file **worker-deployment.yaml**:

> To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi worker-deployment.yaml
root@kubemaster:~/app# cat worker-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker-app-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      name: worker-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: worker-app-pod
      labels:
        name: worker-app-pod
        app: demo-voting-app

    spec:
      containers:
      - name: worker-app
        image: dockersamples/examplevotingapp_worker
```

Important : This file describes a Deployment. Note that the Deployment creates **a** replica of the POD specified by **template** containing a container called **worker-app** which is created from the **dockersamples/examplevotingapp_worker** image.

Next, create the file **postgres-deployment.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi postgres-deployment.yaml
root@kubemaster:~/app# cat postgres-deployment.yaml
---
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: postgres-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: postgres-pod
      app: demo-voting-app
  template:
    metadata:
      name: postgres pod
      labels:
        name: postgres-pod
        app: demo-voting-app

    spec:
      containers:
      - name: postgres
        image: postgres:9.4
        env:
        - name: POSTGRES_USER
          value: postgres
        - name: POSTGRES_PASSWORD
          value: postgres
        ports:
        - containerPort: 5432
```

Important : This file describes a Deployment. Note that the Deployment creates **a** replica of the POD specified by **template** containing a container named **postgres** which uses port 5432 and is created from the **postgres:9.4** image.

Finally, create the file **result-app-deployment.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi result-app-deployment.yaml
root@kubemaster:~/app# cat result-app-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: result-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app-pod
        app: demo-voting-app

    spec:
      containers:
      - name: result-app
        image: dockersamples/examplevotingapp_result
        ports:
```

```
        - containerPort: 80
```

Important : This file describes a Deployment. Note that the Deployment creates **a** replica of the POD specified by **template** containing a container named **result-app** which uses port 80 and is created from the **dockersamples/examplevotingapp_result** image.

2.3 - Creating Services

Now create the **redis-service.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi redis-service.yaml
root@kubemaster:~/app# cat redis-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    name: redis-service
    app: demo-voting-app

spec:
  ports:
  - port: 6379
    targetPort: 6379
```

```
  selector:
    name: redis-pod
    app: demo-voting-app
```

Important : This file describes a **ClusterIP** Service. Note that the Service exposes port **6379** on any POD with the name **redis-pod**.

Next, create the file **postgres-service.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi postgres-service.yaml
root@kubemaster:~/app# cat postgres-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: db
  labels:
    name: db-service
    app: demo-voting-app

spec:
  ports:
  - port: 5432
    targetPort: 5432
  selector:
    name: postgres-pod
```

```
    app: demo-voting-app
```

Important : This file describes a **ClusterIP** Service. Note that the Service exposes port **5432** on any POD with the name **postgres-pod**.

Create the file **voting-app-service.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi voting-app-service.yaml
root@kubemaster:~/app# cat voting-app-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: voting-service
  labels:
    name: voting-service
    app: demo-voting-app

spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
  selector:
    name: voting-app-pod
    app: demo-voting-app
```

Important : This file describes a **NodePort** Service. Note that the Service exposes port **80** on any POD with the name **voting-app-pod**.

Finally, create the file **result-app-service.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~/app# vi result-app-service.yaml
root@kubemaster:~/app# cat result-app-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: result-service
  labels:
    name: result-service
    app: demo-voting-app

spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
  selector:
    name: result-app-pod
    app: demo-voting-app
```

Important : This file describes a **NodePort** Service. Note that the Service exposes port **80** on any POD with the name **result-app-pod**.

2.4 - Deploying the Application

Check that you have created all the necessary YAML files:

```
root@kubemaster:~/myapp# ls
postgres-deployment.yaml redis-deployment.yaml result-app-deployment.yaml voting-app-deployment.yaml worker-
deployment.yaml
postgres-service.yaml redis-service.yaml result-app-service.yaml voting-app-service.yaml
```

Then use the **kubectl create** command:

```
root@kubemaster:~/myapp# kubectl create -f .
deployment.apps/postgres-deployment created
service/db created
deployment.apps/redis-deployment created
service/redis created
deployment.apps/result-app-deployment created
service/result-service created
deployment.apps/voting-app-deployment created
service/voting-service created
deployment.apps/worker-app-deployment created
```

Important: Note the use of the **.** character to indicate any file in the current directory.

Wait until all Deployments are **READY** (7 to 10 minutes):

```
root@kubemaster:~/myapp# kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
postgres-deployment      1/1     1            1           51m
redis-deployment         1/1     1            1           51m
result-app-deployment    1/1     1            1           51m
voting-app-deployment    1/1     1            1           51m
worker-app-deployment    1/1     1            1           51m
```

Next, check the status of the PODs:

```
root@kubemaster:~/myapp# kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
postgres-deployment-5b8bd66778-j99zz    1/1     Running   0          51m
redis-deployment-67d4c466c4-9wzfn       1/1     Running   0          51m
result-app-deployment-b8f9dc967-nzbgd   1/1     Running   0          51m
voting-app-deployment-669dccccfb-jpn6h  1/1     Running   0          51m
worker-app-deployment-559f7749b6-jh86r  1/1     Running   0          51m
```

and the list of Services :

```
root@kubemaster:~/myapp# kubectl get services
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db               ClusterIP   10.107.90.45     <none>        5432/TCP       24h
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP        4d9h
redis            ClusterIP   10.102.154.105   <none>        6379/TCP       24h
result-service   NodePort    10.103.192.107   <none>        80:31526/TCP   24h
voting-service   NodePort    10.96.42.244     <none>        80:32413/TCP   24h
```

In the case of the example in this course, the application now looks like the following diagram:

2.5 - Scaling Up

Edit the **voting-app-deployment.yaml** file and change the value of the **replicas** field from 1 to 3 :

```
root@kubemaster:~/app# vi voting-app-deployment.yaml
root@kubemaster:~/app# cat voting-app-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 3
  selector:
    matchLabels:
      name: voting-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app-pod
        app: demo-voting-app

    spec:
      containers:
      - name: voting-app
        image: dockersamples/examplevotingapp_vote
        ports:
        - containerPort: 80
```

Edit the **result-app-deployment.yaml** file and change the value of the **replicas** field from 1 to 3 :

```
root@kubemaster:~/app# vi result-app-deployment.yaml
root@kubemaster:~/app# cat result-app-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-deployment
  labels:
    app: demo-voting-app
spec:
  replicas: 3
  selector:
    matchLabels:
      name: result-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app-pod
        app: demo-voting-app

    spec:
      containers:
      - name: result-app
        image: dockersamples/examplevotingapp_result
        ports:
        - containerPort: 80
```

Apply the changes using the **kubectl apply** command:

```
root@kubemaster:~/myapp# kubectl apply -f voting-app-deployment.yaml
```

```
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment.apps/voting-app-deployment configured

root@kubemaster:~/myapp# kubectl apply -f result-app-deployment.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment.apps/result-app-deployment configured
```

Then check the Deployments :

```
root@kubemaster:~/myapp# kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
postgres-deployment     1/1     1            1           23h
redis-deployment        1/1     1            1           23h
result-app-deployment   3/3     3            3           23h
voting-app-deployment   3/3     3            3           23h
worker-app-deployment   1/1     1            1           23h
```

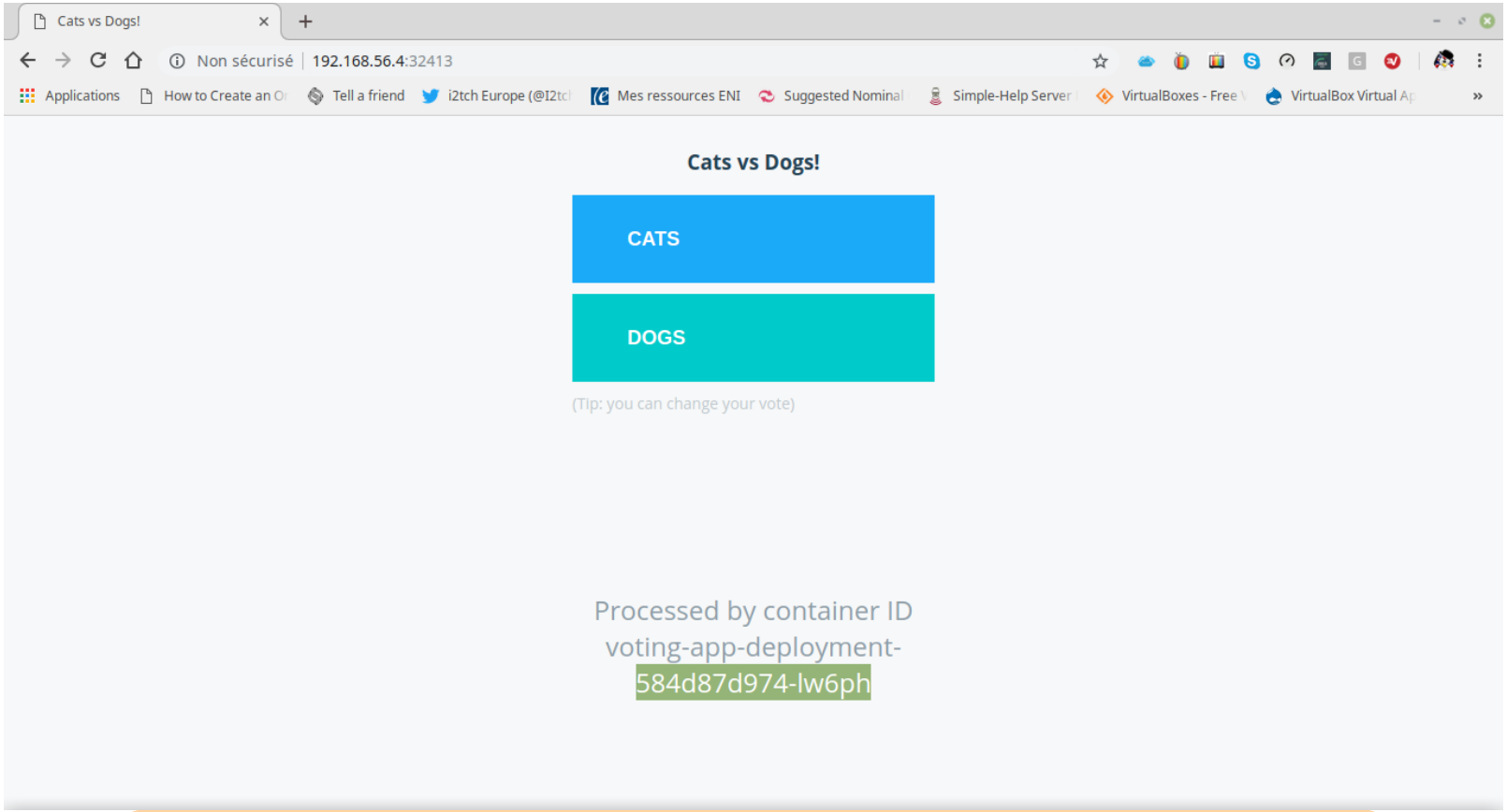
as well as the PODs :

```
root@kubemaster:~/myapp# kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE     IP                NODE
NOMINATED NODE   READINESS GATES
postgres-deployment-5b8bd66778-j99zz     1/1     Running   1          169m    192.168.35.83     kubenode2
<none>           <none>
redis-deployment-67d4c466c4-9wzfn        1/1     Running   1          169m    192.168.205.217   kubenode1
<none>           <none>
result-app-deployment-b8f9dc967-nzbgd    1/1     Running   1          169m    192.168.205.218   kubenode1
<none>           <none>
result-app-deployment-b8f9dc967-r84k6    1/1     Running   0          2m36s   192.168.35.86     kubenode2
<none>           <none>
result-app-deployment-b8f9dc967-zbsk2    1/1     Running   0          2m36s   192.168.35.85     kubenode2
<none>           <none>
voting-app-deployment-669dccccfb-jpn6h   1/1     Running   1          169m    192.168.35.82     kubenode2
<none>           <none>
```

```
voting-app-deployment-669dccccfb-ktd7d   1/1     Running   0          2m50s   192.168.35.84     kubenode2
<none>           <none>
voting-app-deployment-669dccccfb-x868p   1/1     Running   0          2m50s   192.168.205.219   kubenode1
<none>           <none>
worker-app-deployment-559f7749b6-jh86r   1/1     Running   2          169m    192.168.205.216   kubenode1
<none>           <none>
```

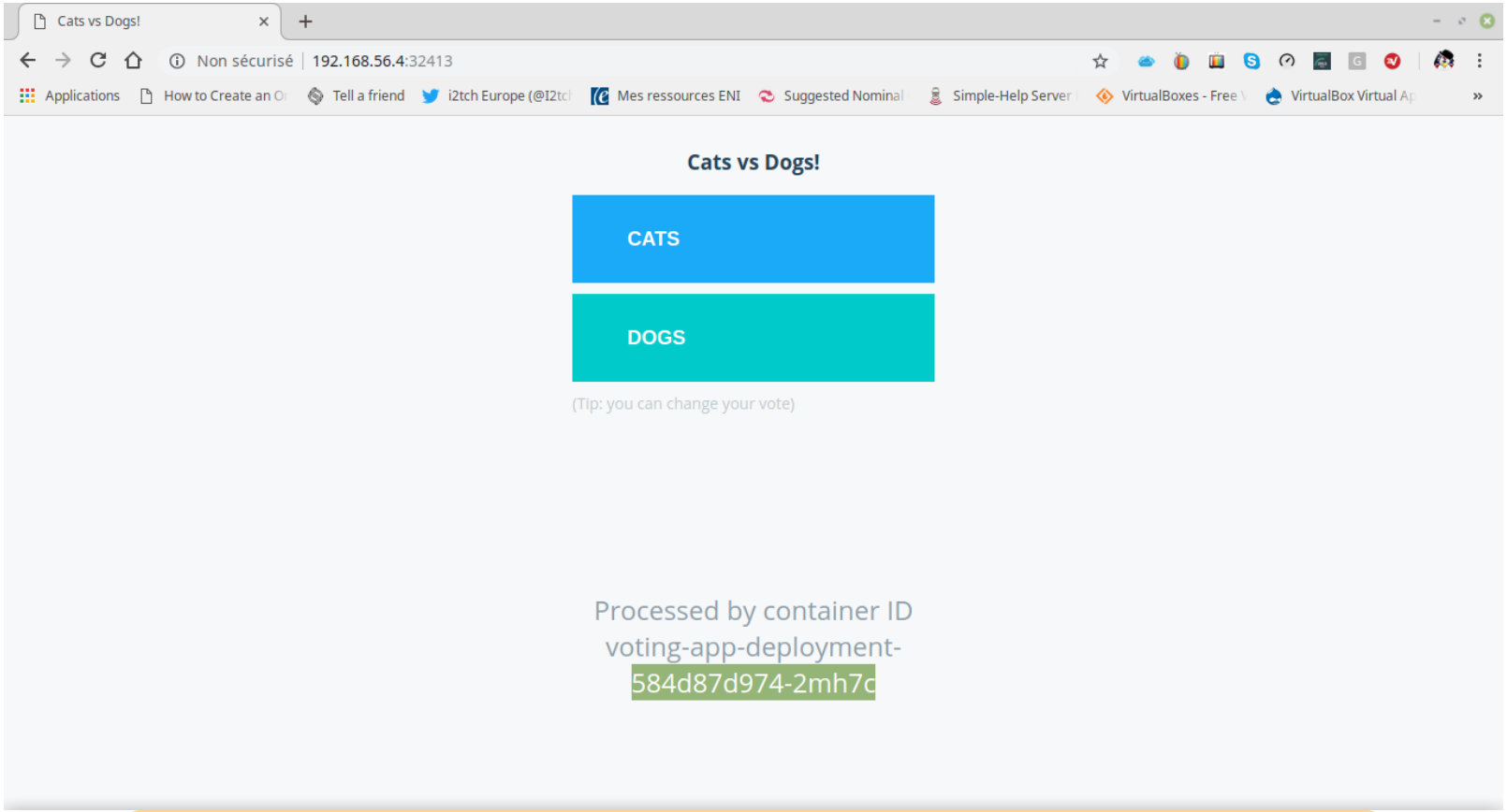
In the case of the example in this course, the application now looks like the following diagram:

Return to the browser on your host machine and refresh the voting-app page:



Important: Note the POD that served the page.

Refresh the page again:



Important: Note that the POD that served the page has changed.

Note that this POD change does not indicate load balancing. You'd have to set up another virtual machine under, say, HAProxy to achieve load balancing.

On the other hand, in the case of an application on GCP, for example, you need to modify the following two files by changing the **type** field value from NodePort to **LoadBalancer** and then configure an instance of GCP's native Load Balancer:

```
root@kubemaster:~/app# vi voting-app-service.yaml
```

```
root@kubemaster:~/app# cat voting-app-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: voting-service
  labels:
    name: voting-service
    app: demo-voting-app

spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    name: voting-app-pod
    app: demo-voting-app
```

Important: This file describes a **LoadBalancer** Service. Note that the Service exposes port **80** on any POD with the name **voting-app-pod**.

Finally, create the file **result-app-service.yaml** :

```
root@kubemaster:~/app# vi result-app-service.yaml
root@kubemaster:~/app# cat result-app-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: result-service
```

```
  labels:
    name: result-service
    app: demo-voting-app

spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    name: result-app-pod
    app: demo-voting-app
```
