Version - **2020.03**

Dernière mise-à-jour : 2020/12/31 10:14

DOF304 - Gestion du Réseau, des Services et d'une Architecture de Microservices

Contenu du Module

LAB #1 - Gestion du Réseau et des Services

1.1 - Présentation

Kubernetes impose des conditions pour l’implémentation d'un réseau :

- Les PODs sur un nœud peuvent communiquer avec tous les PODs sur tous le nœuds sans utiliser NAT,
- Les agents sur un nœud (par exemple kubelet) peuvent communiquer avec tous les PODs sur le nœud.

Important : La description technique et détaillée de l'approche réseau de Kubernetes peut être consultée à l'adresse : **https://kubernetes.io/docs/concepts/cluster-administration/networking/**.

Dans le cluster de ce cours, le réseau mis en place entre les noeuds Kubernetes est le 172.18.0.0/16 :

```
root@debian10:~# kubectl get nodes -o wide
NAME                 STATUS   ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION   CONTAINER-RUNTIME
kind-control-plane   Ready    master   26h   v1.19.1   172.18.0.5    <none>        Ubuntu Groovy Gorilla
(development branch)   4.19.0-6-amd64   containerd://1.4.0
kind-worker2         Ready    <none>   26h   v1.19.1   172.18.0.4    <none>        Ubuntu Groovy Gorilla
(development branch)   4.19.0-6-amd64   containerd://1.4.0
kind-worker3         Ready    <none>   26h   v1.19.1   172.18.0.3    <none>        Ubuntu Groovy Gorilla
(development branch)   4.19.0-6-amd64   containerd://1.4.0
```

Kind crée les noeuds en tant que conteneurs Docker dans l'hôte Debian_10 :

```
root@debian10:~# docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
NAMES
2cff99fff93b        kindest/node:v1.19.1   "/usr/local/bin/entr…"   27 hours ago        Up 27 hours
kind-worker
9b810f08fcc4        kindest/node:v1.19.1   "/usr/local/bin/entr…"   27 hours ago        Up 27 hours
kind-worker2
23dd96c58ceb        kindest/node:v1.19.1   "/usr/local/bin/entr…"   27 hours ago        Up 27 hours
```

```
127.0.0.1:44833->6443/tcp   kind-control-plane
1106161c7cd5        kindest/node:v1.19.1   "/usr/local/bin/entr…"   27 hours ago        Up 27 hours
kind-worker3
```

ainsi que le réseau **kind** de type **bridge** pour les relier :

```
root@debian10:~# docker network list
NETWORK ID          NAME                DRIVER              SCOPE
e6d50c85fcc4        bridge              bridge              local
471d983b1248        host                host                local
aac5f2655b24        kind                bridge              local
50b8123f99bf        none                null                local
```

La commande **docker network inspect** montre clairement la configuration de chaque conteneur :

```
root@debian10:~# docker network inspect kind
[
    {
        "Name": "kind",
        "Id": "aac5f2655b24912d0b7f88c538927edfc8464bd3c7f0b3a7fb438069d667eff7",
        "Created": "2020-11-30T14:12:57.708788165+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": true,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                },
                {
                    "Subnet": "fc00:f853:ccd:e793::/64",
```

```
                "Gateway": "fc00:f853:ccd:e793::1"
            }
        ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "1106161c7cd51db1694b7fe00e97636d5ed20be2cdfa56d32e3c335c29697539": {
            "Name": "kind-worker3",
            "EndpointID": "66a2356715009d227026231f0cbe66ccb718939a80e63945501c6d7e0d17b1c9",
            "MacAddress": "02:42:ac:12:00:03",
            "IPv4Address": "172.18.0.3/16",
            "IPv6Address": "fc00:f853:ccd:e793::3/64"
        },
                    "23dd96c58ceb0eaf5b5f0fcac1431825444dd91d860c894bece4690ac302b9ab": {
            "Name": "kind-control-plane",
            "EndpointID": "147e765d44fca78ee2eae41ea247aefefedfe1f97997f51240a0f8496b79f095",
            "MacAddress": "02:42:ac:12:00:05",
            "IPv4Address": "172.18.0.5/16",
            "IPv6Address": "fc00:f853:ccd:e793::5/64"
        },
        "2cff99fff93bf15f2c511da420cde9fc350823c4224a573a0d8aed5380a28300": {
            "Name": "kind-worker",
            "EndpointID": "76e472551da35af4713c3314c1ca93bce09d9d6905fc9ff44db9775c65951924",
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": "fc00:f853:ccd:e793::2/64"
        },
        "9b810f08fcc4a3fcc7755a691af724a3f115dbbc1f4a8656a6dc9be55f4ac1f9": {
```

```
                "Name": "kind-worker2",
                "EndpointID": "b2f3b842915c81211351d181fb13650ea3d4d0e631fa06c50f2a4906d436748e",
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": "fc00:f853:ccd:e793::4/64"
            }
        },
        "Options": {
            "com.docker.network.bridge.enable_ip_masquerade": "true"
        },
        "Labels": {}
    }
]
```

Sous Kubernetes, les adresses IP ne sont pas attachées aux conteneurs dans les PODs de chaque noeud mais aux PODs eux-mêmes :

```
root@debian10:~# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE           NOMINATED
NODE   READINESS GATES
myapp-deployment-6f899c7745-g6888   1/1     Running   0          8h    10.244.1.18   kind-worker2   <none>
<none>
myapp-deployment-6f899c7745-gdpp8   1/1     Running   0          8h    10.244.1.19   kind-worker2   <none>
<none>
myapp-deployment-6f899c7745-rtqhq   1/1     Running   0          8h    10.244.3.15   kind-worker3   <none>
<none>
```

Il est possible d'obtenir les CIDR des PODs en utilisant la commande suivante :

```
root@debian10:~# kubectl get nodes -o jsonpath='{.items[*].spec.podCIDR}'
10.244.0.0/24 10.244.1.0/24 10.244.3.0/24root@debian10:~#
```

Notez que les adresses **10.244.1.x** sont associées aux PODs sur kind-worker2 tandis que les adresses **10.244.3.x** sont associées aux PODs sur kind-worker3. Ces adresses sont issues du réseau **10.244.0.0/16** stipulé par l'option **–pod-network-cidr** lors de l'initialisation du maître du cluster :

```
root@debian10:~# kubectl cluster-info dump | grep -m 1 cluster-cidr
                            "--cluster-cidr=10.244.0.0/16",
```

L'extension réseau utilisée par kind n'est pas une des classiques déjà présentées. En fait il s'agit d'une extension propre au projet dénommée **kindnet** :

```
root@debian10:~# kubectl get pods -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
coredns-f9fd979d6-hd5sh                      1/1     Running   0          26h
coredns-f9fd979d6-q7tcx                      1/1     Running   0          26h
etcd-kind-control-plane                      1/1     Running   0          26h
kindnet-2vgnb                                1/1     Running   0          26h
kindnet-6x6pk                                1/1     Running   0          26h
kindnet-snk42                                1/1     Running   0          26h
kube-apiserver-kind-control-plane            1/1     Running   0          26h
kube-controller-manager-kind-control-plane   1/1     Running   0          26h
kube-proxy-lkljb                             1/1     Running   0          26h
kube-proxy-mfgcf                             1/1     Running   0          26h
kube-proxy-wl4mk                             1/1     Running   0          26h
kube-scheduler-kind-control-plane            1/1     Running   0          26h
```

En sachant que dans chaque POD existe un conteneur Nginx, testez si vous pouvez afficher la page d'accueil de Nginx en vous connectant à kind-worker2 et kind-worker2 :

```
root@debian10:~# curl 172.18.0.3
curl: (7) Failed to connect to 172.18.0.3 port 80: Connection refused
root@debian10:~# curl 172.18.0.4
curl: (7) Failed to connect to 172.18.0.4 port 80: Connection refused
```

Important : Notez l'échec de la connexion.

Testez maintenant si vous pouvez afficher la page d'accueil de Nginx en vous connectant à un des PODs :

```
root@debian10:~# curl 10.244.1.18
^C
```

Important : Retenez donc qu'à ce stade il n'est pas possible d'afficher la page d'accueil de Nginx en vous connectant de l'extérieur du cluster.

Lors de l'installation du cluster contenant kubemaster, kubenode1 et kubenode2 nous avons spécifié l'utilisation d'une extension réseau appelée **Calico**, issue de la liste suivante :

- **Calico**,
- **Cilium**,
- **Flannel**,
- **Kube-router**,
- **Romana**,
- **WeaveNet**,
- **Antrea**,
- **kube-ovn**,
- Canal (utilise Flannel pour le réseau et Calico pour le pare-feu).

Important : Une étude comparative des extensions réseau pour Kubernetes peut être trouvée à la page : **https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-10gbit-s-network-updated-august-2020-6e1b757b9e49**.

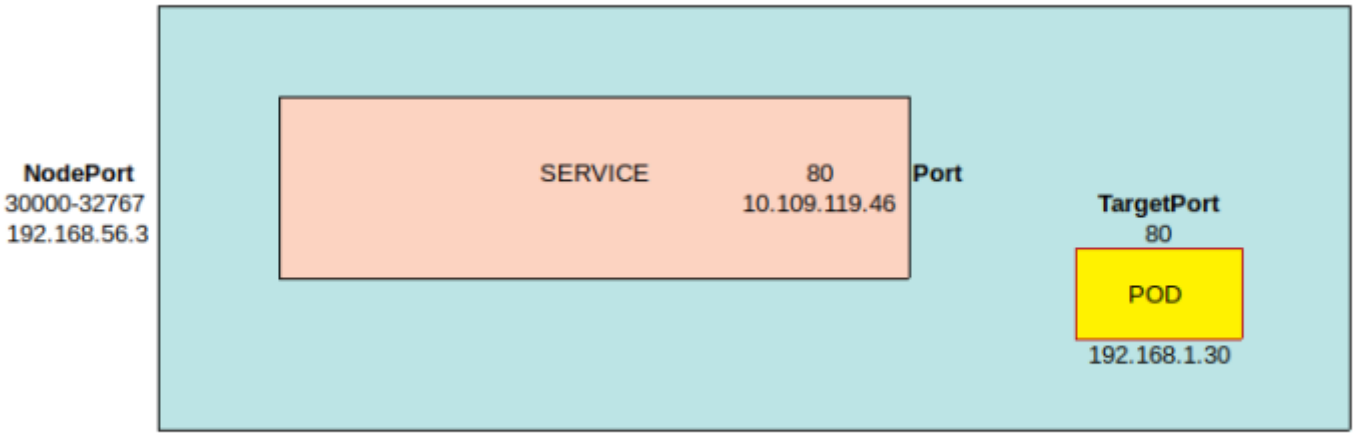
Ces extensions permettent la mise en place de Services :

- NodePort,
  - Ce Service rend un POD accessible sur un port du nœud le contenant,
- ClusterIP
  - Ce Service crée une adresse IP virtuelle afin de permettre la communication entre de services différents dans le cluster, par exemple des serveurs front-end avec des serveurs back-end,
- LoadBalancer
  - Ce service provisionne un équilibrage de charge pour une application dans certains fournisseurs de Cloud publique tels **A**mazon **W**eb **S**ervices et **G**oogle **C**loud **P**latform.

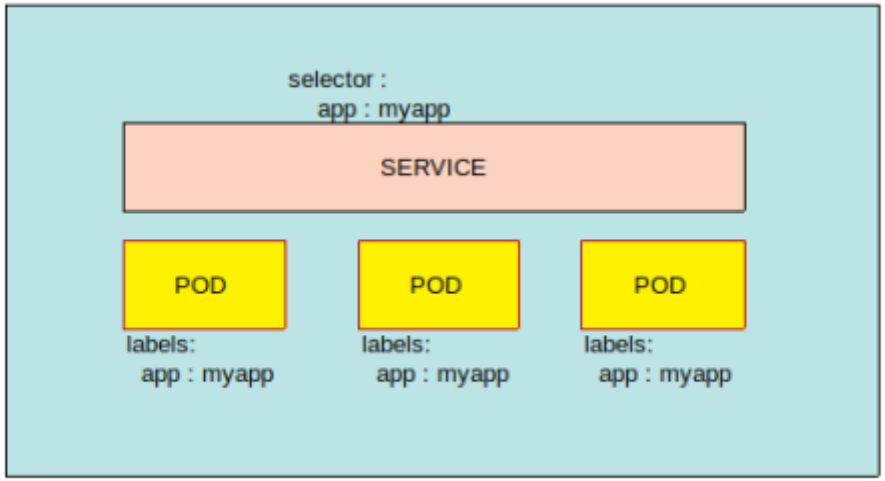
1.2 - Le Service NodePort

Le Service NodePort définit trois ports :

- **TargetPort** : le port sur le POD,
- **Port** : le port sur le Service lié à un IP du Cluster,
- **NodePort** : le port sur le Nœud issu de la plage 30000-32767.

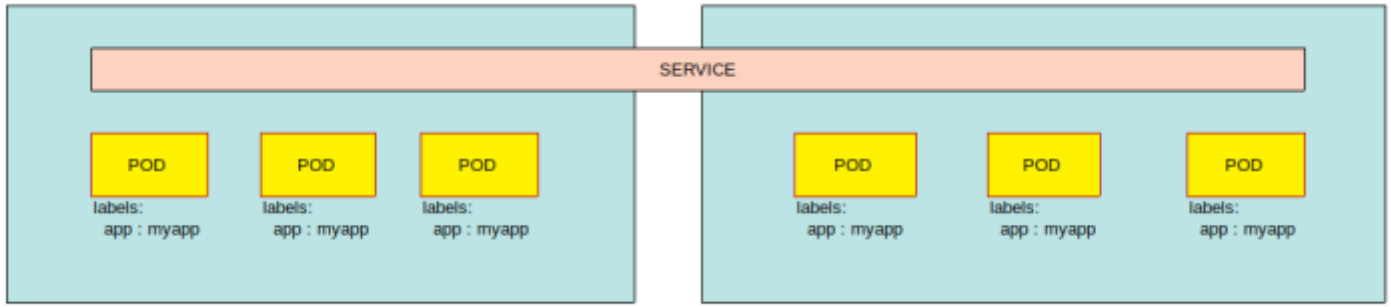


Si dans le même nœud, plusieurs PODs ont les étiquettes qui correspondent au **selector** du Service, le Service identifie les PODs et s'étend automatiquement pour englober tous les PODs. Les PODs sont appelés des **End-Points** :



Important : Notez que dans ce cas l'équilibrage de charge est automatique est utilise l'algorithme **Random** avec une affinité de session..

De même, quand les PODs sont distribués sur plusieurs nœuds, le Service s'étend pour tout englober :



Créez donc le fichier YAML **service-definition.yaml** :

```
root@debian10:~# vi service-definition.yaml
root@debian10:~# 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 : Notez que si le champ **type:** est manquant, sa valeur par défaut est **ClusterIP**. Notez aussi que dans **ports**, seul le champ **port** est obligatoire. Si le champ

targetPort est manquant, sa valeur par défaut est celle du champ **port**. Si le champ **nodePort** est manquant, sa valeur par défaut est le premier port disponible dans la plage entre **30 000** et **32 767**. Dernièrement, il est possible de spécifier de multiples définitions de ports dans le service.

Le champs **selector** contient les étiquettes des PODs concernés par la mise en place du Service :

```
root@debian10:~# 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
```

Créez le Service en utilisant le fichier **service-definition.yaml** :

```
root@debian10:~# kubectl create -f service-definition.yaml
service/myapp-service created
```

Constatez la création du Service :

```
root@debian10:~# kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        27h
```

```
myapp-service   NodePort    10.96.228.251   <none>        80:30008/TCP   25s
```

Important : Notez que le Service a une adresse IP du cluster et qu'il a exposé le port **30 008**.

Testez maintenant si vous pouvez afficher la page d'accueil de Nginx en vous connectant à un des PODs en utilisant le service NodePort :

```
root@debian10:~# curl 172.18.0.3:30008
<!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>
```

1.3 - Le Service ClusterIP

Le Service **ClusterIP** permet de regrouper les PODs offrant le même service afin de faciliter la communication, par exemple :

- 3 PODs front-end = une adresse ClusterIP,
- 3 PODs back-end = une autre adresse ClusterIP.

Pour créer un Service ClusterIP, créez le fichier **clusterip-definition.yaml** :

```
root@debian10:~# vi clusterip-definition.yaml
root@debian10:~# cat clusterip-definition.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: back-end

spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
  selector:
    app: myapp
    type: front-end
```

Créez le Service en utilisant le fichier **clusterip-definition.yaml** :

```
root@debian10:~# kubectl create -f clusterip-definition.yaml
```

```
service/back-end created
```

Vérifiez maintenant la présence du Service :

```
root@debian10:~# kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
back-end        ClusterIP   10.96.188.7     <none>        80/TCP         14s
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        45h
myapp-service   NodePort    10.96.228.251   <none>        80:30008/TCP   17h
```

Supprimez maintenant les Services créés :

```
root@debian10:~# kubectl delete service myapp-service
service "myapp-service" deleted
root@debian10:~# kubectl delete service back-end
service "back-end" deleted
```

Dernièrement supprimez le Deployment **myapp-deployment** :

```
root@debian10:~# kubectl delete deployment myapp-deployment
deployment.apps "myapp-deployment" deleted
```

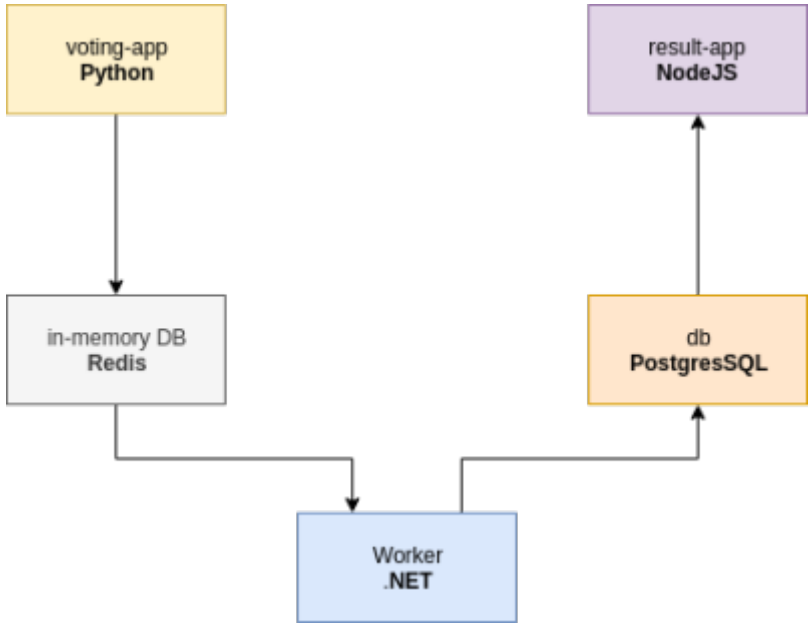
Vérifiez qu'il ne reste que le service par défaut **kubernetes** :

```
root@debian10:~# kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   45h
```

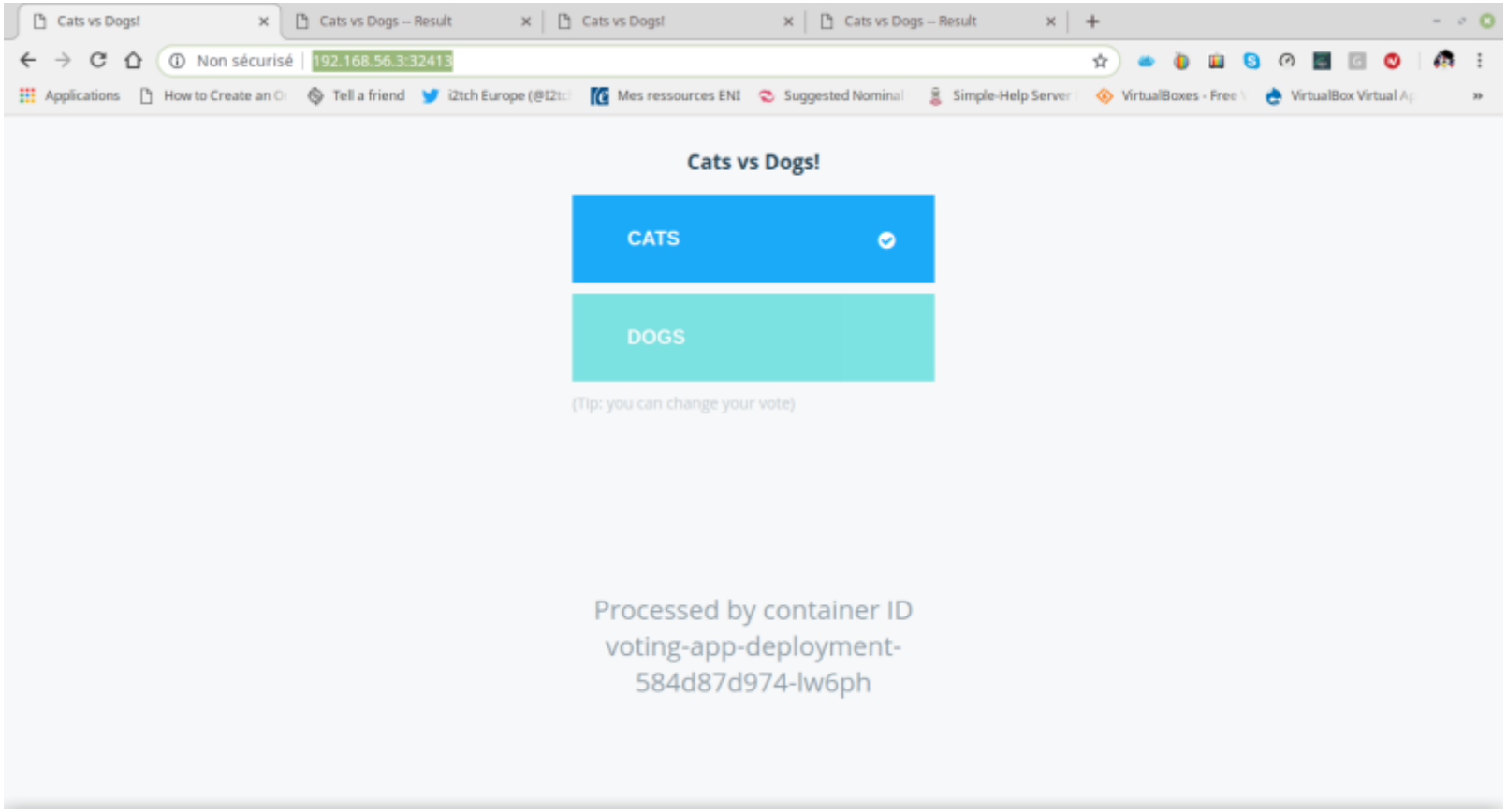
LAB #2 - Gestion d'une Architecture de Microservices

2.1 - Présentation

Vous allez mettre en place une application simple, appelé **demo-voting-app** et développé par Docker, sous forme de microservices :

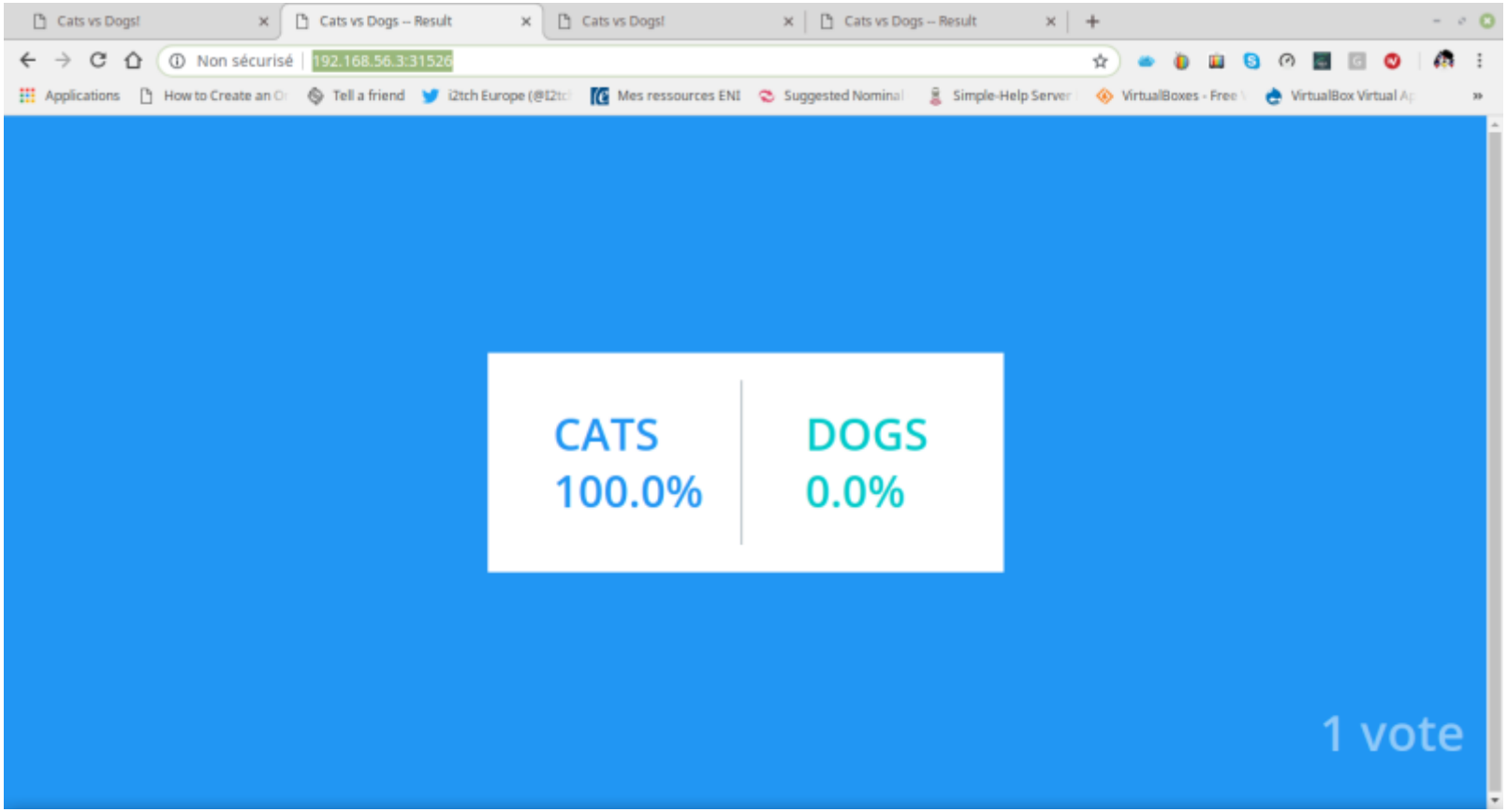


Dans cette application le conteneur **voting-app** permet de voter pour des **chats** ou des **chiens**. Cette application tourne sous Python et fournit une interace HTML :



Lors de la vote, le résultat de celle-ci est stocké dans **Redis** dans une base de données en mémoire. Le résultat est ensuite passé au conteneur **Worker** qui tourne sous .NET et qui met à jour la base de données persistante dans le conteneur **db** qui tourne sous PostgreSQL.

L'application **result-app** qui tourne sous NodeJS lit ensuite la table dans la base de données PostgreSQL et affiche le résultat sous forme HTML :



Cette application peut être mise en place sous docker avec les commandes suivantes :

```
docker run -d --name=redis redis
docker run -d --name=db -e POSTGRES_PASSWORD=postgres -e POSTGRES_USER=postgres postgres:9.4
docker run -d --name=vote -p 5000:80 --link redis:redis dockersamples/examplevotingapp_vote
docker run -d --name=result -p 5001:80 --link db:db dockersamples/examplevotingapp_result
docker run -d --name=worker --link db:db ---link redis:redis dockersamples/examplevotingapp_worker
```

Par contre, Docker annonce le retrait éventuel de l'option **–lien** et indique qu'il vaudrait mieux utiliser des réseaux pour assurer la communication entre les conteneurs :

“Warning: The –link flag is a legacy feature of Docker. It may eventually be removed. Unless you absolutely need to continue using it, we recommend that you use user-defined networks to facilitate communication between two containers instead of using –link. One feature that user-defined networks do not support that you can do with –link is sharing environment variables between containers. However, you can use other mechanisms such as volumes to share environment variables between containers in a more controlled way.”

Cette application peut être mise en place sous docker swarm avec les commandes suivantes :

```
docker@manager1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
ENGINE VERSION
vwshwppuaoze785gy12k0gh62 *   manager1            Ready               Active              Leader
18.09.3
t0rjtq76j35mbn44olp0t3yeq     worker1             Ready               Active
18.09.3
udv7w988tepuba7pf6rb5k1o3     worker2             Ready               Active
18.09.3
uz2m26qe0hdf7lplb9a5m0ysv     worker3             Ready               Active
18.09.3
sfig9atrbgzt41sjxhj95wfgu     worker4             Ready               Active
18.09.3
56az1cupssf9uqx9h0yvbmydw     worker5             Ready               Active
18.09.3
```

```
docker@manager1:~$ vi docker-stack.yml
docker@manager1:~$ cat docker-stack.yml
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
```

```
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
```

```
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
```

```
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
  db-data:
```

```
docker@manager1:~$ docker stack deploy -c docker-stack.yml app
Creating network app_backend
Creating network app_frontend
Creating network app_default
Creating service app_worker
Creating service app_visualizer
Creating service app_redis
Creating service app_db
Creating service app_vote
Creating service app_result
```

2.2 - Création des Deployments

Créez le répertoire **myapp**. Placez-vous dans ce répertoire et créez le fichier **voting-app-deployment.yaml** :

```
root@debian10:~# mkdir myapp
root@debian10:~# cd myapp
root@debian10:~/myapp# vi voting-app-deployment.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Deployment. Notez que le Deployment crée **un** replica du POD spécifié par **template** contenant un conteneur dénommé **voting-app** qui utilise le port 80 et qui est créé à partir de l'image **dockersamples/examplevotingapp_vote**.

Créez maintenant le fichier **redis-deployment.yaml** :

```
root@debian10:~/myapp# vi redis-deployment.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Deployment. Notez que le Deployment crée **un** replica du POD spécifié par **template**

contenant un conteneur dénommé **redis** qui utilise le port 6379 et qui est créé à partir de l'image **redis**.

Créez le fichier **worker-deployment.yaml** :

```
root@debian10:~/myapp# vi worker-deployment.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Deployment. Notez que le Deployment crée **un** replica du POD spécifié par **template** contenant un conteneur dénommé **worker-app** qui est créé à partir de l'image **dockersamples/examplevotingapp_worker**.

Créez ensuite le fichier **postgres-deployment.yaml** :

```
root@debian10:~/myapp# vi postgres-deployment.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Deployment. Notez que le Deployment crée **un** replica du POD spécifié par **template** contenant un conteneur dénommé **postgres** qui utilise le port 5432 et qui est créé à partir de l'image **postgres:9.4**.

Dernièrement, créez le fichier **result-app-deployment.yaml** :

```
root@debian10:~/myapp# vi result-app-deployment.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Deployment. Notez que le Deployment crée **un** replica du POD spécifié par **template** contenant un conteneur dénommé **result-app** qui utilise le port 80 et qui est créé à partir de l'image **dockersamples/examplevotingapp_result**.

2.3 - Création des Services

Créez maintenant le fichier **redis-service.yaml** :

```
root@debian10:~/myapp# vi redis-service.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Service **ClusterIP**. Notez que le Service expose le port **6379** sur tout POD ayant le nom **redis-pod**.

Créez ensuite le fichier **postgres-service.yaml** :

```
root@debian10:~/myapp# vi postgres-service.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Service **ClusterIP**. Notez que le Service expose le port **5432** sur tout POD ayant le nom **postgres-pod**.

Créez le fichier **voting-app-service.yaml** :

```
root@debian10:~/myapp# vi voting-app-service.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Service **NodePort**. Notez que le Service expose le port **80** sur tout POD ayant le nom **voting-app-pod**.

Dernièrement, créez le fichier **result-app-service.yaml** :

```
root@debian10:~/myapp# vi result-app-service.yaml
root@debian10:~/myapp# 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 : Ce fichier décrit un Service **NodePort**. Notez

que le Service expose le port **80** sur tout POD ayant le nom **result-app-pod**.

2.4 - Déployer l'Application

Vérifiez que vous avez créé les fichiers ***9** YAML necéssaires :

```
root@debian10:~/myapp# ls -l
total 36
-rw-r--r-- 1 root root 590 Dec 15 10:59 postgres-deployment.yaml
-rw-r--r-- 1 root root 222 Dec 15 11:03 postgres-service.yaml
-rw-r--r-- 1 root root 439 Dec 15 10:57 redis-deployment.yaml
-rw-r--r-- 1 root root 225 Dec 15 11:01 redis-service.yaml
-rw-r--r-- 1 root root 494 Dec 15 11:00 result-app-deployment.yaml
-rw-r--r-- 1 root root 253 Dec 15 11:05 result-app-service.yaml
-rw-r--r-- 1 root root 492 Dec 15 10:56 voting-app-deployment.yaml
-rw-r--r-- 1 root root 253 Dec 15 11:04 voting-app-service.yaml
-rw-r--r-- 1 root root 451 Dec 15 10:57 worker-deployment.yaml
```

Utilisez ensuite la commande **kubectl create** :

```
root@debian10:~/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 : Notez l'utilisation du caractère **.** qui indique tout fichier dans le répertoire courant.

Attendez que tous les Deployments soient **READY** (2 à 3 minutes) :

```
root@debian10:~/myapp# kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
postgres-deployment      0/1     1            0           27s
redis-deployment         0/1     1            0           27s
result-app-deployment    0/1     1            0           27s
voting-app-deployment    0/1     1            0           26s
worker-app-deployment    0/1     1            0           25s

root@debian10:~/myapp# kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
postgres-deployment      1/1     1            1           2m43s
redis-deployment         1/1     1            1           2m43s
result-app-deployment    1/1     1            1           2m43s
voting-app-deployment    1/1     1            1           2m42s
worker-app-deployment    1/1     1            1           2m41s
```

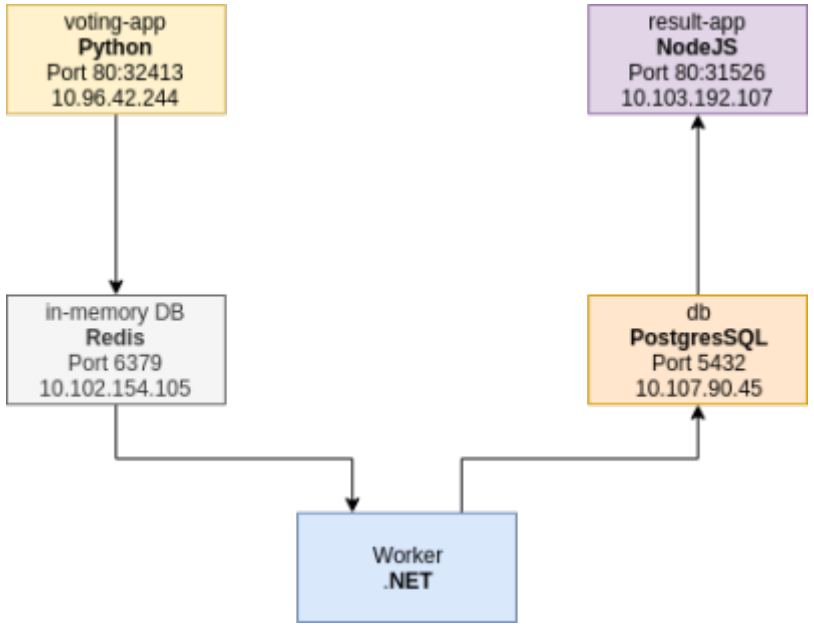
Contrôlez ensuite l'état des PODs :

```
root@debian10:~/myapp# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
postgres-deployment-746bc85b8-8lw6c   1/1     Running   0          3m34s
redis-deployment-64cff75679-8zqr8     1/1     Running   0          3m34s
result-app-deployment-7cdc94dfcd-nddsh   1/1     Running   0          3m34s
voting-app-deployment-678c67fc7-zcs6c    1/1     Running   0          3m33s
worker-app-deployment-767d5b67ff-sgj2x   1/1     Running   0          3m32s
```

ainsi que la liste des Services :

```
root@debian10:~/myapp# kubectl get services
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
db               ClusterIP   10.96.30.165   <none>        5432/TCP       4m2s
kubernetes       ClusterIP   10.96.0.1      <none>        443/TCP        46h
redis            ClusterIP   10.96.99.190   <none>        6379/TCP       4m2s
result-service   NodePort    10.96.128.82   <none>        80:31801/TCP   4m1s
voting-service   NodePort    10.96.73.238   <none>        80:30343/TCP   4m
```

Dans le cas donc de l'exemple dans ce cours, l'application ressemble maintenant au diagramme suivant :



2.5 - Tester l'Application

Connectez-vous à votre serveur cloud en utilisant X2Go. Ouvrez Oracle VirtualBox et double-cliquez sur la machine virtuelle **Debian_10**. Connectez-vous à la VM en tant que **trainee** avec le mot de passe **trainee** :

Testez ensuite votre application en vous connectant à http://172.18.0.4:30343 à partir du navigateur dans la machine virtuelle Debian_10.

Important : Modifiez les sockets en fonction de votre installation.

2.6 - Scaling Up

Éditez le fichier **voting-app-deployment.yaml** et modifiez la valeur du champ **replicas** de 1 à 3 :

```
root@debian10:~/myapp# vi voting-app-deployment.yaml
root@debian10:~/myapp# 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
```

Éditez le fichier **result-app-deployment.yaml** et modifiez la valeur du champ **replicas** de 1 à 3 :

```
root@debian10:~/myapp# vi result-app-deployment.yaml
root@debian10:~/myapp# 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
```

Appliquez les modifications à l'aide de la commande **kubectl apply** :

```
root@debian10:~/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@debian10:~/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
```

Contrôlez ensuite les Deployments :

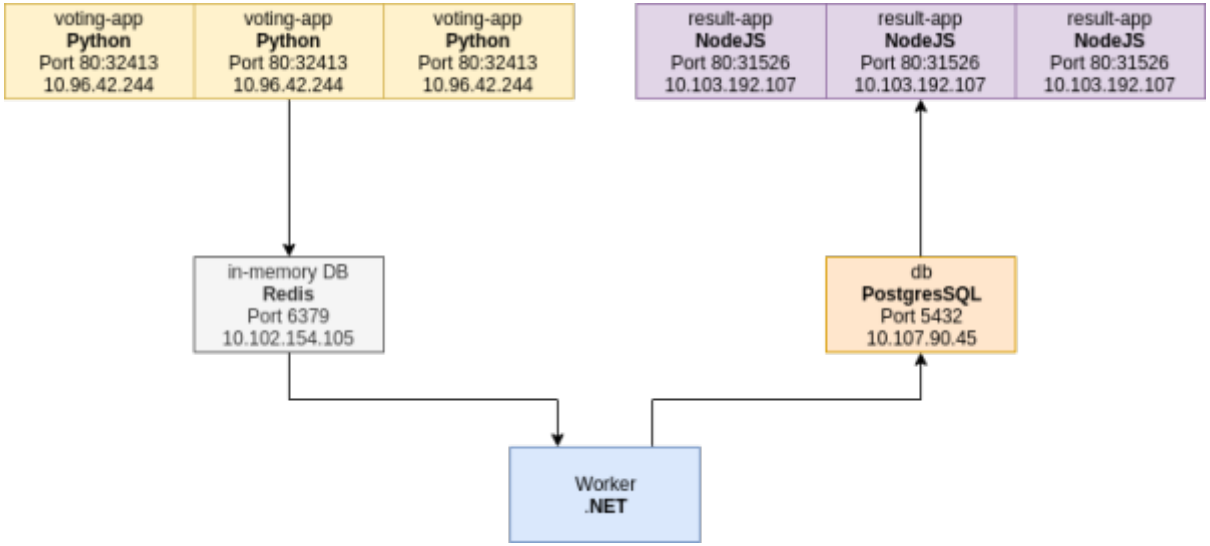
```
root@debian10:~/myapp# kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
postgres-deployment     1/1     1            1           68m
redis-deployment        1/1     1            1           68m
result-app-deployment   1/3     3            1           68m
voting-app-deployment   3/3     3            3           68m
worker-app-deployment   1/1     1            1           68m
```

ainsi que les PODs :

```
root@debian10:~/myapp# kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
postgres-deployment-746bc85b8-8lw6c    1/1     Running   1          69m
redis-deployment-64cff75679-8zqr8      1/1     Running   1          69m
```

```
result-app-deployment-7cdc94dfcd-nddsh   1/1     Running   1          69m
result-app-deployment-7cdc94dfcd-ntbdj   1/1     Running   0          54s
result-app-deployment-7cdc94dfcd-wsm2d   1/1     Running   0          54s
voting-app-deployment-678c67fc7-59q7z    1/1     Running   0          67s
voting-app-deployment-678c67fc7-sgczf    1/1     Running   0          67s
voting-app-deployment-678c67fc7-zcs6c    1/1     Running   1          69m
worker-app-deployment-767d5b67ff-sgj2x   1/1     Running   2          69m
```

Dans le cas de l'exemple dans ce cours, l'application ressemble maintenant au diagramme suivant :



Retournez sur le navigateur de votre VM Debian_10 et rafraichissez la page du voting-app :

Important : Notez le POD qui a servi la page.

Rafraîchissez la page de nouveau :

Important : Notez que le POD qui a servi la page a changé.

Notez que ce changement de POD n'indique pas un équilibrage de charge. En effet, sous VirtualBox, il faudrait mettre en place une autre machine virtuelle sous, par exemple, HAProxy pour obtenir l'équilibrage.

Par contre, dans le cas d'une application sur GCP par exemple, il convient de modifier les deux fichiers suivants en changeant la valeur de champ **type** de NodePort à **LoadBalancer** puis de configurer une instance du Load Balancer natif de GCP :

```
root@debian10:~/myapp# 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 : Ce fichier décrit un Service **LoadBalancer**. Notez que le Service expose le port **80** sur tout POD ayant le nom **voting-app-pod**.

Dernièrement, créez le fichier **result-app-service.yaml** :

```
root@debian10:~/myapp# 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
```
