



---

# PROJET KUBERNETES

---

Groupe Girls



**Membres :**  
**BEJAoui Yosser**  
**CHENG Xin Jie**  
**SAMB Weuly**

### Table des matières

Introduction .....	2
Avant Défi .....	2
Défi 1 .....	3
Défi 2 .....	5
Étape 1 : Comprendre le Deployment .....	6
Étape 2 : Configurer le fichier YAML du Deployment .....	6
Réponse aux questions du défi 2 .....	8
Défi 3 .....	9
Étape 1 : Préparer le projet Django .....	9
Étape 2 : Créer l'Image Docker pour Django .....	9
Étape 3 : Déployer la Base de Données PostgreSQL .....	10
Étape 4 : Déployer le Projet Django dans Kubernetes .....	11
Étape 5 : Lancer les Deployments dans Kubernetes .....	13
Étape 6 : Accéder à l'Application depuis machine locale .....	13
Réponse aux questions du défi 3 .....	14
Défi 4 .....	15
Étape 1 : Configurer un Ingress pour le site .....	15
Étape 2 : Modifier le projet Django .....	15
Étape 3 : Accéder au site web .....	16
Défi 5 .....	18
Étape 1 : Créer deux images Docker pour API et Public .....	18
Étape 2 : Déployer les Applications avec Kubernetes .....	19
Étape 3 : Configurer l'Ingress pour accéder aux applications .....	20
Étape 4 : Accéder au site web .....	20
Défi 6 .....	22
Étape 1 : Créer une chart Helm .....	22
Étape 2 : Modifier le chart Helm .....	22
Étape 3 : Modifier le fichier values.yaml .....	28
Étape 4 : Installer le chart Helm .....	28
Étape 5 : Accéder au site web .....	29
Conclusion .....	30
Annexes .....	31
Table des Illustrations .....	31

# Introduction

Ce projet a pour objectif de développer nos compétences pratiques sur Kubernetes, une plateforme d'orchestration de conteneurs, en déployant des applications dans un environnement Kubernetes. À travers plusieurs défis, nous avons appris à gérer des applications conteneurisées, à configurer les services Kubernetes pour exposer ces applications, et à optimiser l'utilisation des ressources dans le cluster.

Les étapes du projet ont permis de comprendre les concepts clés de Kubernetes, tels que les Pods, Deployments, Services, et Ingress, ainsi que l'utilisation de Docker pour containeriser les applications et Helm pour automatiser le déploiement. Ce projet a aussi impliqué l'intégration de bases de données et la gestion de la communication entre différents services dans un environnement Kubernetes.

## Avant Défi

Avant de commencer les défis, nous avons préparé l'environnement Kubernetes en suivant les étapes suivantes.

Créer un dossier `.kube` dans le dossier Home :

```
mkdir ~/.kube
```

Déplacer le fichier téléchargé à l'adresse `~/.kube/config`:

```
mv ~/Downloads/csc8567.yaml ~/.kube/config
```

Lister les pods en cours d'exécutions dans un cluster :

```
kubectl get pods -n u-cptz2
```

```
xinjie@MacBook-XinJie .kube % kubectl get pods -n u-cptz2
No resources found in u-cptz2 namespace.
```

car pour le moment, on n'a pas encore déployé un Pod.

## Défi 1

L'objectif de ce défi est de déployer une application web simple dans un Pod à partir d'une image Docker, et de rendre cette application accessible via un navigateur en utilisant le port forwarding.

On a d'abord téléchargé une image Docker depuis Docker Hub vers la machine locale :

```
docker pull xhelozs/csc8567:v1
```

On crée un Deployment nommé **deploy1** dans notre namespace **u-cptz2**, en utilisant l'image Docker téléchargé précédemment :

```
kubectl create deployment deploy1 --image=xhelozs/csc8567:v1 -n u-cptz2
```

Ce Deployment va créer un Pod qui exécute l'image Docker **xhelozs/csc8567:v1**.

```
xinjie@MacBook-XinJie .kube % kubectl create deployment deploy1 --image=xhelozs/csc8567:v1 -n u-cptz2
deployment.apps/deploy1 created
```

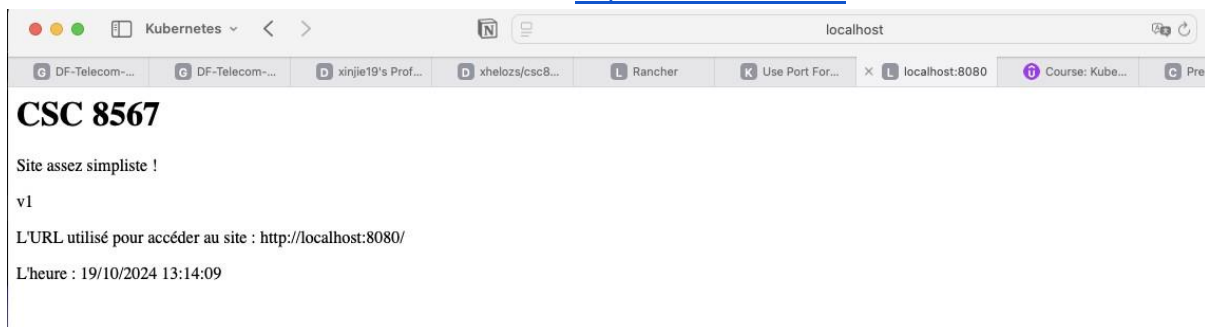
On effectue un port-forwarding pour permettre l'accès à un Pod depuis la machine locale:

```
kubectl port-forward pods/deploy1-6ff764d787-b6w5r 8080:5000 -n u-cptz2
```

**8080:5000** : on dirige le **port 5000** du Pod vers le **port 8080** sur la machine locale (D'après le Dockerfile sur GitHub, on voit que l'application utilise Flask, et que par défaut Flask utilise le port 5000)

Cette commande redirige les requêtes envoyées à **localhost:8080** sur la machine locale vers le port **5000** du Pod **deploy1-6ff764d787-b6w5r** qui s'exécute dans le namespace **u-cptz2**.

Le site est maintenant accessible via le lien <http://localhost:8080/>



Les commandes utiles :

- Pour afficher les Pod :  
`kubectl get pod -n u-cptz2`
- Pour afficher les Deployment :  
`kubectl get deployment -n u-cptz2`
- Pour afficher les Nodes : `kubectl get node -n u-cptz2`
- Pour afficher les logs :

```
kubectl logs <nom de pod> -n <namespace> kubectl
logs deploy1-6ff764d787-b6w5r -n u-cptz2
```

## Projet Kubernetes - Girls

```
xinjie@MacBook-XinJie .kube % kubectl get pod -n u-cptz2
NAME                                READY   STATUS    RESTARTS   AGE
deploy1-6ff764d787-b6w5r           1/1     Running   0           25h

xinjie@MacBook-XinJie .kube % kubectl get deployment -n u-cptz2
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
deploy1   1/1     1            1           25h

xinjie@MacBook-XinJie .kube % kubectl logs deploy1-6ff764d787-b6w5r -n u-cptz2
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.42.4.9:5000
Press CTRL+C to quit
127.0.0.1 - - [18/Oct/2024 12:16:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2024 12:16:11] "GET /favicon.ico HTTP/1.1" 404 -

xinjie@MacBook-XinJie .kube % kubectl get node -n u-cptz2
NAME                STATUS   ROLES                                AGE   VERSION
kube-agent-1        Ready    <none>                               20d   v1.30.5+rke2r1
kube-agent-2        Ready    <none>                               20d   v1.30.5+rke2r1
kube-master-1       Ready    control-plane,etcd,master           20d   v1.30.5+rke2r1
kube-master-2       Ready    control-plane,etcd,master           20d   v1.30.5+rke2r1
kube-master-3       Ready    control-plane,etcd,master           20d   v1.30.5+rke2r1

xinjie@MacBook-XinJie .kube % kubectl get pod deploy1-6ff764d787-b6w5r -o wide -n u-cptz2
NAME                                READY   STATUS    RESTARTS   AGE   IP        Pods   NODE
deploy1-6ff764d787-b6w5r           1/1     Running   0           46h   10.42.4.9 kube-agent-2
```

(le Pod est exécuté dans node agent kube-agent-2)

Schéma Défi 1 :

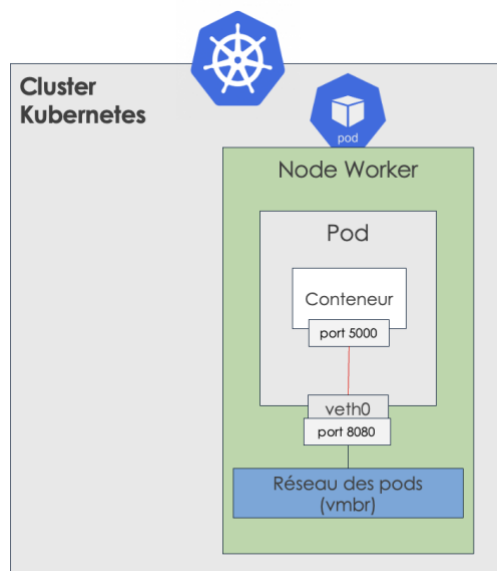


Figure 1: Schéma illustratif du défi 1

## Défi 2

L'objectif de ce défi est de créer un Deployment avec 3 répliques de pods utilisant l'image csc8567, puis d'exposer ces pods via un service ClusterIP pour les rendre accessibles à l'intérieur du cluster. Ensuite, le service doit être accessible depuis la machine locale en utilisant un proxy Kubernetes. Enfin, on doit allouer et limiter les ressources CPU et mémoire pour chaque Pod.

Ce défi nous guide dans l'apprentissage des bases de Kubernetes, en nous demandant de déployer une application Docker et de configurer son accès dans un environnement Kubernetes. Voici ce qu'il nous permet de découvrir et d'accomplir ensemble :

### 1. Créer et Gérer un Déploiement Kubernetes

- Objectif : Nous apprenons à utiliser un Deployment pour gérer l'application Docker (image `csc8567`). Cela nous permet de créer plusieurs répliques du pod pour assurer que l'application est toujours disponible, même si l'un des pods rencontre un problème.
- En pratique : Kubernetes s'assure automatiquement que trois répliques de notre pod sont en cours d'exécution, et le Deployment gère le redémarrage en cas de défaillance.

### 2. Exposer l'Application avec un Service ClusterIP et le Proxy

- Objectif : Nous découvrons comment rendre notre application accessible dans le cluster avec un Service de type ClusterIP.
- En pratique : Nous configurons un Service pour que les autres applications dans le cluster puissent y accéder. Puis, grâce au proxy (`kubectl proxy`), nous pouvons tester notre application depuis notre machine locale sans la rendre accessible publiquement.

### 3. Allouer et Limiter les Ressources pour chaque Pod

- Objectif : Ce défi nous enseigne à gérer efficacement les ressources, en spécifiant la quantité de CPU et de mémoire RAM dont chaque pod a besoin.
- En pratique : Nous fixons des valeurs de requests (pour allouer des ressources minimales) et de limits (pour éviter la surconsommation). Cela nous permet de maintenir une utilisation optimale des ressources dans le cluster, assurant que chaque pod fonctionne correctement sans nuire aux autres.

### 4. Utilisation d'un Proxy pour Accéder à l'Application

- Objectif : Utiliser `kubectl proxy` pour accéder aux services internes dans le cluster, ce qui nous permet de tester l'application localement.
- En pratique : En lançant `kubectl proxy`, nous redirigeons les requêtes locales vers notre service dans Kubernetes, ce qui nous permet de vérifier que tout fonctionne correctement avant de l'exposer éventuellement à l'extérieur du cluster.

### Étape 1 : Comprendre le Deployment

Un **Deployment** en Kubernetes gère la création, la mise à jour et l'échelle d'un ensemble de pods identiques. En définissant un **Deployment**, on peut spécifier :

- Le nombre de réplicas (pods identiques)
- L'image Docker à utiliser
- Les ressources demandées et limites pour chaque pod

### Étape 2 : Configurer le fichier YAML du Deployment

1. Ouvrir le dossier kube
2. Créer un fichier deployment.yaml

Le contenu du fichier deployment.yaml:

```
GNU nano 7.2 deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: csc8567-deployment
  namespace: u-cptz2 # Mettez ici le namespace correct
spec:
  replicas: 3 # Nombre de répliques
  selector:
    matchLabels:
      app: csc8567-app
  template:
    metadata:
      labels:
        app: csc8567-app
    spec:
      containers:
      - name: csc8567-container
        image: xhelozs/csc8567:v1 # Image Docker
        resources:
          requests:
            cpu: "100m" # 1/10 CPU
            memory: "100Mi" # 100 Mo de RAM
          limits:
            cpu: "200m" # 1/5 CPU
            memory: "200Mi" # 200 Mo de RAM
```

3. Créer un fichier service.yaml

```
! app-service.yaml > {} spec
io.k8s.api.core.v1.Service (v1@service.json)
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: app-service
5    namespace: u-cptz2
6  spec:
7    selector:
8      app: my-app
9    ports:
10     - protocol: TCP
11       port: 80 # Port sur lequel le service va écouter
12       targetPort: 5000 # Le port où ton application écoute dans le pod
13   type: ClusterIP
```

### 4. Créer le deployment pour les 3 pods

```
yosser@yosser:~/k8e$ kubectl apply -f deployment.yaml
```

### 5. Créer le deployment pour le service

```
yosser@yosser:~/k8e$ kubectl apply -f service.yaml
```

### 6. Lance le proxy

```
yosser@yosser:~/k8e$ kubectl proxy
W1031 15:22:39.114689 25135 proxy.go:172] Your kube context contains a server path /k8s/clusters/local, use --append-server-path to automatically append the path to each request
Starting to serve on 127.0.0.1:8001
```

Afficher les pods et service :

```
xinjie@MacBook-XinJie..k8e % kubectl get pod -n u-cptz2
NAME                                READY   STATUS    RESTARTS   AGE
app-deployment-5c799888c9-7k4gd     1/1     Running   0           3m22s
app-deployment-5c799888c9-h6dbg     1/1     Running   0           3m23s
app-deployment-5c799888c9-pnxch     1/1     Running   0           3m22s
deploy1-6ff764d787-b6w5r            1/1     Running   0           4d19h

xinjie@MacBook-XinJie..k8e % kubectl get service -n u-cptz2
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
app-service   ClusterIP   10.43.42.223 <none>        80/TCP     2m8s
```

### 7. Accéder au site via ce lien <http://127.0.0.1:8001/api/v1/namespaces/u-cptz2/services/csc8567-service/proxy/>

← → ↻ 127.0.0.1:8001/api/v1/namespaces/u-cptz2/services/csc8567-service/proxy/

## CSC 8567

Site assez simpliste !

v1

L'URL utilisé pour accéder au site : <http://csc8567.luxbulb.org/>

L'heure : 31/10/2024 14:23:30



## Réponse aux questions du défi 2

1. Quel est le but d'un service ?

Un service permet de rendre les applications disponibles au sein du cluster Kubernetes ou à l'extérieur, et il gère la répartition de la charge entre les Pods derrière lui.

2. Quelle est la différence entre les services ClusterIP et NodePort ?

Un service de type **ClusterIP** expose le service à l'intérieur du cluster uniquement, c'est-à-dire que seul le trafic provenant d'autres Pods ou services du même cluster peut accéder à ce service. Alors qu'un service de type **NodePort** permet d'exposer le service en dehors du cluster.

3. Schéma du défi 2 :

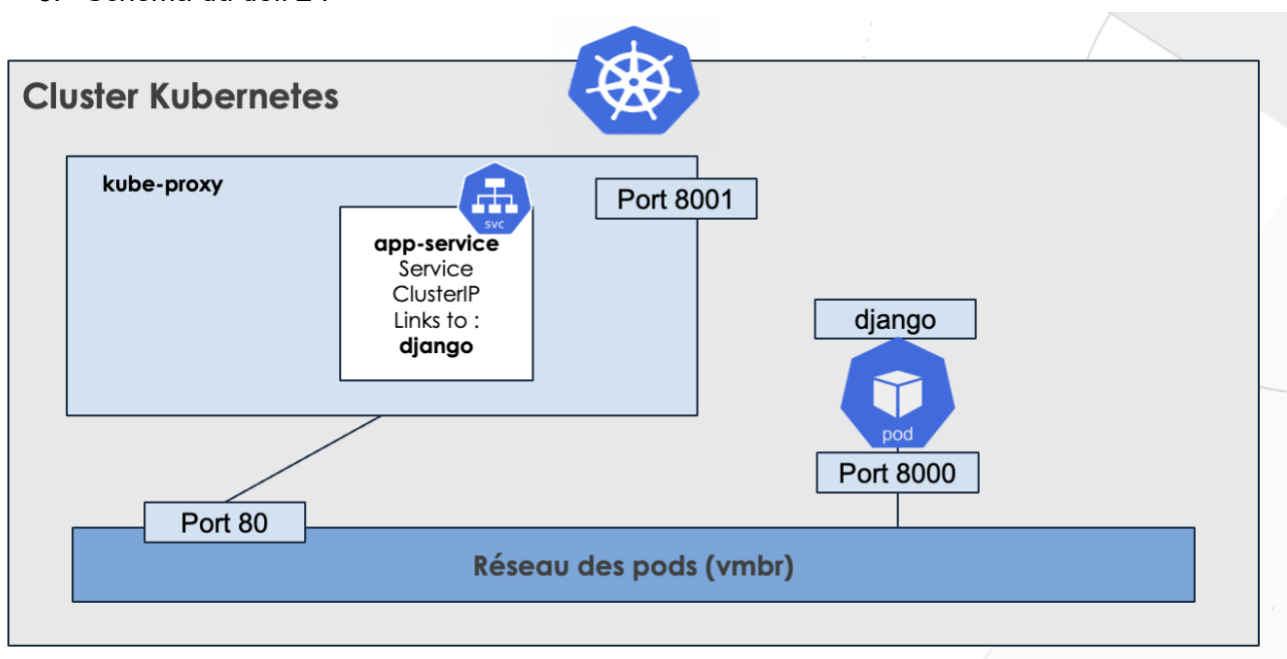


Figure 2: Schéma illustratif du défi 2

## Défi 3

L'objectif principal de ce défi est de **déployer une application Django avec une base de données PostgreSQL** dans un environnement Kubernetes. Ce défi nous permet d'apprendre à :

1. **Conteneuriser le projet Django**, incluant les applications public et api, dans une image Docker.
2. **Publier cette image sur Docker Hub**, pour qu'elle soit accessible et réutilisable.
3. **Déployer cette image dans Kubernetes**, en utilisant un Deployment pour le site Django et un autre pour la base de données PostgreSQL.
4. **Utiliser des services ClusterIP pour la communication interne**, permettant à l'application Django de communiquer avec PostgreSQL dans le cluster.

### Étape 1 : Préparer le projet Django

On utilise le projet Django qu'on a créé lors de la première partie du cours. On fait un copier-coller du répertoire de ce projet dans notre répertoire `.kube`.

### Étape 2 : Créer l'Image Docker pour Django

#### 1. Écrire un Dockerfile pour le projet Django

On crée un nouveau fichier Dockerfile qui combine les deux applications (public et api) en un seul pour créer une seule image Docker.

```
Dockerfile.kbt U X
projet-web-tsp > Dockerfile.kbt > ...
1  # Utilisation d'une image de base légère Python
2  FROM python:3.8-alpine
3
4  # Définir des variables d'environnement
5  ENV PYTHONUNBUFFERED=1
6
7  # Définir le répertoire de travail
8  WORKDIR /projet-web-tsp/django-site/todolist
9
10 # Installer les dépendances système requises
11 RUN apk add --no-cache gcc musl-dev python3-dev libffi-dev
12
13 # Copier les fichiers du projet Django dans le conteneur
14 COPY ./django-site/todolist /projet-web-tsp/django-site/todolist
15
16 # Installer les dépendances Python
17 RUN pip install --upgrade pip && \
18     pip install -r ./requirements.txt
19
20 # Exposer le port sur lequel Django fonctionne
21 EXPOSE 8000
22
23 # Commande pour démarrer le serveur Django
24 CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
25
```

## Projet Kubernetes - Girls

Modification sur fichier settings.py :

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', 'localhost:8001', 'csc8567.luxbulb.org',  
                 'django.girls.csc8567.luxbulb.org', 'frontend', 'proxy', 'api']  
  
DATABASES = {  
    'default': dj_database_url.config(  
        default='postgres://{{os.getenv('POSTGRES_USER', 'xinjie')}}:{{os.getenv('POSTGRES_PASSWORD', 'xinjie')}}@{{os.getenv('DB_HOST', 'db-service')}}:5432/{{os.getenv('POSTGRES_DB', 'todolist')}}'  
    )  
}
```

Pour la partie **DATABASES**, on configure la connexion de Django à PostgreSQL en utilisant les variables d'environnement.

### 2. Construire l'image Docker et pousser l'image sur Docker Hub

La commande pour construire l'image Docker et pousser sur Docker Hub :

```
docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64 -t xinjie19/kubeapipub:<version> -f projet-  
web-tsp/Dockerfile.kbt --push projet-web-tsp
```

```
xinjie@MacBook-XinJie .kube % docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64 -t xinjie19/kubeapipub:v8 -f projet-web-tsp/Dockerfile.kbt --push projet-web-tsp
```

Après avoir poussé sur Docker Hub, on fait un pull pour récupérer l'image :

```
xinjie@MacBook-XinJie .kube % docker pull xinjie19/kubeapipub:v8  
v8: Pulling from xinjie19/kubeapipub
```

## Étape 3 : Déployer la Base de Données PostgreSQL

### 1. On crée un Deployment pour PostgreSQL :

On écrit un fichier de configuration YAML pour déployer PostgreSQL dans Kubernetes qui inclure :

- Un Deployment pour gérer le Pod de PostgreSQL.
- Un Service de type ClusterIP pour permettre à Django de se connecter à PostgreSQL en interne.

### 2. On configure les variables d'environnement pour PostgreSQL :

On ajoute les variables d'environnement dans le fichier de Deployment **dbdeploy.yaml** pour définir les informations de connexion, comme le nom de la base de données, l'utilisateur et le mot de passe.

```
! dbdeploy.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: dbdeploy
5    namespace: u-cptz2
6  spec:
7    replicas: 1
8    selector:
9      matchLabels:
10       app: postgres
11  template:
12    metadata:
13      labels:
14       app: postgres
15    spec:
16      containers:
17      - name: postgres
18        image: postgres:13
19        env:
20        - name: POSTGRES_DB
21          value: todolist # Nom de la base de données
22        - name: POSTGRES_USER
23          value: xinjie # Nom d'utilisateur
24        - name: POSTGRES_PASSWORD
25          value: xinjie # Mot de passe
26        ports:
27        - containerPort: 5432
28        volumeMounts:
29        - name: postgres-storage
30          mountPath: /var/lib/postgresql/data
31      volumes:
32      - name: postgres-storage
33        emptyDir: {}

! dbservice.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: dbservice
5    namespace: u-cptz2
6  spec:
7    selector:
8      app: postgres
9    ports:
10     - protocol: TCP
11       port: 5432 # Le port sur lequel PostgreSQL écoute
12       targetPort: 5432
13    type: ClusterIP
```

## Étape 4 : Déployer le Projet Django dans Kubernetes

### 1. On crée un Deployment pour Django :

On écrit un fichier de configuration YAML pour déployer ton projet Django qui inclure : •

Un **Deployment** pour gérer le Pod de Django.

- Un **Service de type ClusterIP** pour permettre d'accéder à l'application depuis kubeproxy et d'autres composants internes.

### 2. On passe les variables d'environnement nécessaires :

On ajoute les informations de connexion à PostgreSQL (host, utilisateur, mot de passe) en tant que variables d'environnement dans le Deployment Django.

```
! servicedjango.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: servicedjango
5    namespace: u-cptz2
6  spec:
7    selector:
8      app: web
9    ports:
10     - protocol: TCP
11       port: 80      # Port sur lequel le service va écouter
12       targetPort: 8000 # Le port où l'application Django écoute dans le pod
13     type: ClusterIP
14

! deploydjango.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: deploydjango
5    namespace: u-cptz2
6  spec:
7    replicas: 1
8    selector:
9      matchLabels:
10       app: web
11    template:
12      metadata:
13        labels:
14         app: web
15      spec:
16        containers:
17         - name: app-container
18           image: xinjie19/kubeapipub:v8
19           ports:
20             - containerPort: 8000
21           env:
22             - name: POSTGRES_USER
23               value: "xinjie"
24             - name: POSTGRES_PASSWORD
25               value: "xinjie"
26             - name: DB_HOST
27               value: "dbservice"
28             - name: POSTGRES_DB
29               value: "todolist"
30           command: ["sh", "-c", "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"]
31        resources:
32          requests:
33            cpu: "100m"
34            memory: "100Mi"
35          limits:
36            cpu: "200m"
37            memory: "200Mi"
```

### Étapes pour connecter le Pod Django au Pod PostgreSQL dans Kubernetes

- dbservice.yaml** crée un Service appelé **dbservice** qui rend PostgreSQL accessible aux autres Pods dans le cluster.
- Dans **deploydjango.yaml**, la variable **DB\_HOST=dbservice** dit à Django de se connecter à PostgreSQL via ce Service.
- Les autres variables d'environnement (**POSTGRES\_USER**, **POSTGRES\_PASSWORD**, **POSTGRES\_DB**) donnent à Django les identifiants pour se connecter à la base de données.

En résumé : **DB\_HOST=dbservice** relie le Pod Django au Pod PostgreSQL en passant par le Service **dbservice**.

### Étape 5 : Lancer les Deployments dans Kubernetes

1. On applique les configurations :

```
kubectl apply -f dbdeploy.yaml
kubectl apply -f dbservice.yaml
kubectl apply -f deploydjango.yaml
kubectl apply -f servicedjango.yaml
```

2. On vérifie les statuts des Pods et Services :

```
xinjie@MacBook-XinJie .kube % kubectl get pods -n u-cptz2
NAME                                READY   STATUS    RESTARTS   AGE
app-deployment-5c799888c9-7k4gd     1/1     Running   0           18d
app-deployment-5c799888c9-h6dbg     1/1     Running   0           18d
app-deployment-5c799888c9-pnxch     1/1     Running   0           18d
csc8567-deployment-7454488cdb-nm7f5 1/1     Running   0           10d
csc8567-deployment-7454488cdb-qn242 1/1     Running   0           10d
csc8567-deployment-7454488cdb-qsfw8 1/1     Running   0           10d
dbdeploy-75db678446-5m5k5           1/1     Running   0           18d
deploy1-6ff764d787-b6w5r            1/1     Running   0           23d
deploydjango-cbc9c586b-z78nt        1/1     Running   0           4d6h

xinjie@MacBook-XinJie .kube % kubectl get services -n u-cptz2
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
app-service   ClusterIP   10.43.42.223 <none>         80/TCP     18d
csc8567-service ClusterIP   10.43.107.40 <none>         80/TCP     10d
dbservice     ClusterIP   10.43.8.162  <none>         5432/TCP   18d
postgres-service ClusterIP   10.43.186.212 <none>         5432/TCP   18d
servicedjango ClusterIP   10.43.51.96  <none>         80/TCP     18d
```

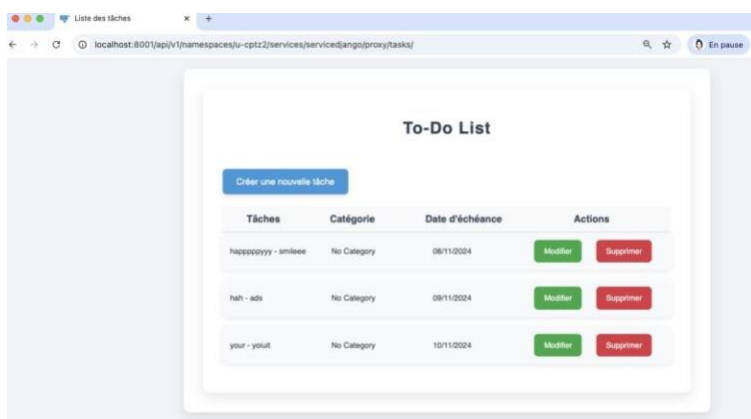
### Étape 6 : Accéder à l'Application depuis machine locale

Après avoir lancé les Deployments, on utilise `kubectl proxy` pour accéder à notre application Django depuis notre machine locale.

```
xinjie@MacBook-XinJie .kube % kubectl proxy
```

Le site est accessible via le lien :

<http://localhost:8001/api/v1/namespaces/u-cptz2/services/servicedjango/proxy/>



## Réponse aux questions du défi 3

- a. Quelle est la différence entre un service **ClusterIP** et **NodePort** ?

**ClusterIP** est utilisé pour la communication interne dans le cluster, tandis que **NodePort** permet d'exposer des services en dehors du cluster.

- b. Quelle critique pouvez-vous donner vis-à-vis de l'utilisation d'un Pod pour la base de données ?

**Manque de persistance des données** : Si un Pod contenant une base de données est redémarré, toutes les données non stockées de manière persistante dans des volumes externes seront perdues.

**Réinitialisation et gestion des pannes** : Si un Pod contenant la base de données échoue ou est supprimé, Kubernetes tentera de redémarrer ce Pod. Cependant, dans de nombreuses configurations de bases de données, ce type de redémarrage pourrait entraîner des problèmes de corruption de données ou des indisponibilités prolongées, car la base de données peut avoir des processus de démarrage ou de récupération spécifiques.

- c. Le schéma :

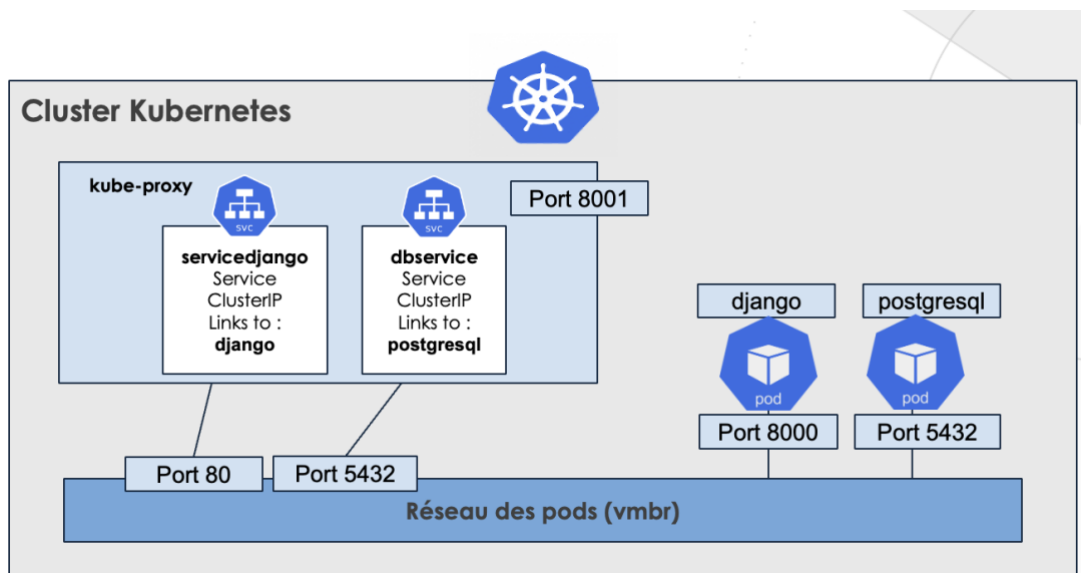


Figure 3: Schéma illustratif du défi 3



## Défi 4

**Objectif :** L'objectif de ce défi est de rendre notre site Django accessible depuis Internet en configurant un Ingress dans Kubernetes.

### Étape 1 : Configurer un Ingress pour le site

1. Créer un fichier `ingress-django.yaml` :

```
! ingress-django.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: ingress-django
5    namespace: u-cptz2
6    annotations:
7      nginx.ingress.kubernetes.io/rewrite-target: /
8  spec:
9    rules:
10     - host: django.girls.csc8567.luxbulb.org
11       http:
12         paths:
13           - path: /
14             pathType: Prefix
15             backend:
16               service:
17                 name: servicedjango # Nom de service Django
18                 port:
19                   number: 80
20     tls:
21     - hosts:
22       - django.girls.csc8567.luxbulb.org
23       secretName: tls-secret # Utilisé pour le HTTPS
24
```

On utilise les services web et base des données qu'on a créé au Défi 3.

2. Appliquer la configuration :

```
xinjie@MacBook-XinJie .kube % kubectl apply -f ingress-django.yaml
ingress.networking.k8s.io/ingress-django created .votre_nom_de_gro
```

### Étape 2 : Modifier le projet Django

1. Ajouter le domaine dans la liste `ALLOWED_HOST` de fichier `settings.py` :

```
django.girls.csc8567.luxbulb.org
```



## Projet Kubernetes - Girls

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', 'localhost:8001', 'csc8567.luxbulb.org',  
                  'django.girls.csc8567.luxbulb.org', 'frontend', 'proxy', 'api']
```

Puisqu'on a fait une modification dans le projet Django, on doit créer une nouvelle version image Docker et la déployer dans Kubernetes pour que les changements soient pris en compte :

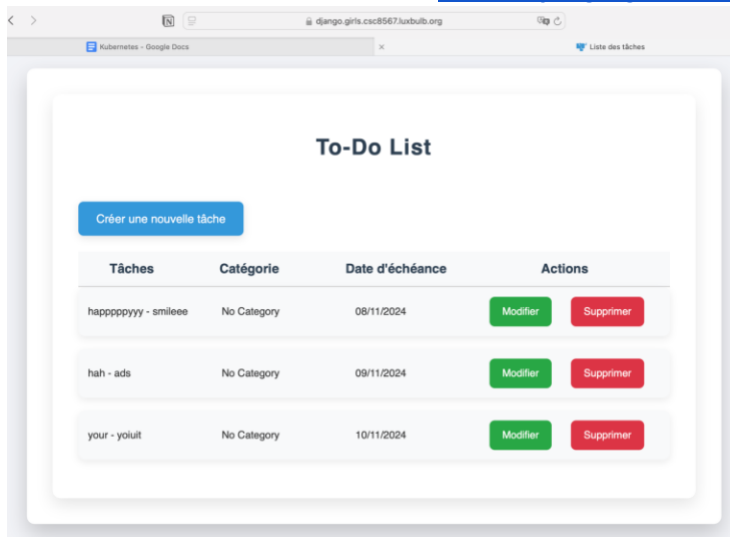
```
docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64  
-t  
xinjie19/kubeapipub:v8 -f projet-web-tsp/Dockerfile.kbt --push projet-web-tsp  
xinjie@MacBook-XinJie .kube % docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64 -t xinjie19/kubeapipub:v8 -f projet-web-tsp/Dockerfile.kbt --push projet-web-tsp  
docker pull xinjie19/kubeapipub:v8
```

### 2. Mettre à jour la version de l'image dans le fichier Deployment deploydjango.yaml :

```
spec:  
  containers:  
  - name: app-container  
    image: xinjie19/kubeapipub:v8  
    ports:  
    - containerPort: 8000  
  
xinjie@MacBook-XinJie .kube % kubectl apply -f deploydjango.yaml -n u-cptz2  
deployment.apps/deploydjango configured
```

## Étape 3 : Accéder au site web

Le site est accessible via le lien : <https://django.girls.csc8567.luxbulb.org/>



## Projet Kubernetes - Girls

```
xinjie@MacBook-XinJie .kube % kubectl get ingress -n u-cptz2
```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
ingress-django	nginx	django.girls.csc8567.luxbulb.org	157.159.11.201	80, 443	4d8h

Le trafic entrant sur l'IP **157.159.11.201** sur les ports 80 (HTTP) et 443 (HTTPS) sera géré par le contrôleur d'Ingress, qui appliquera les règles définies dans le fichier **ingressdjango.yaml**.

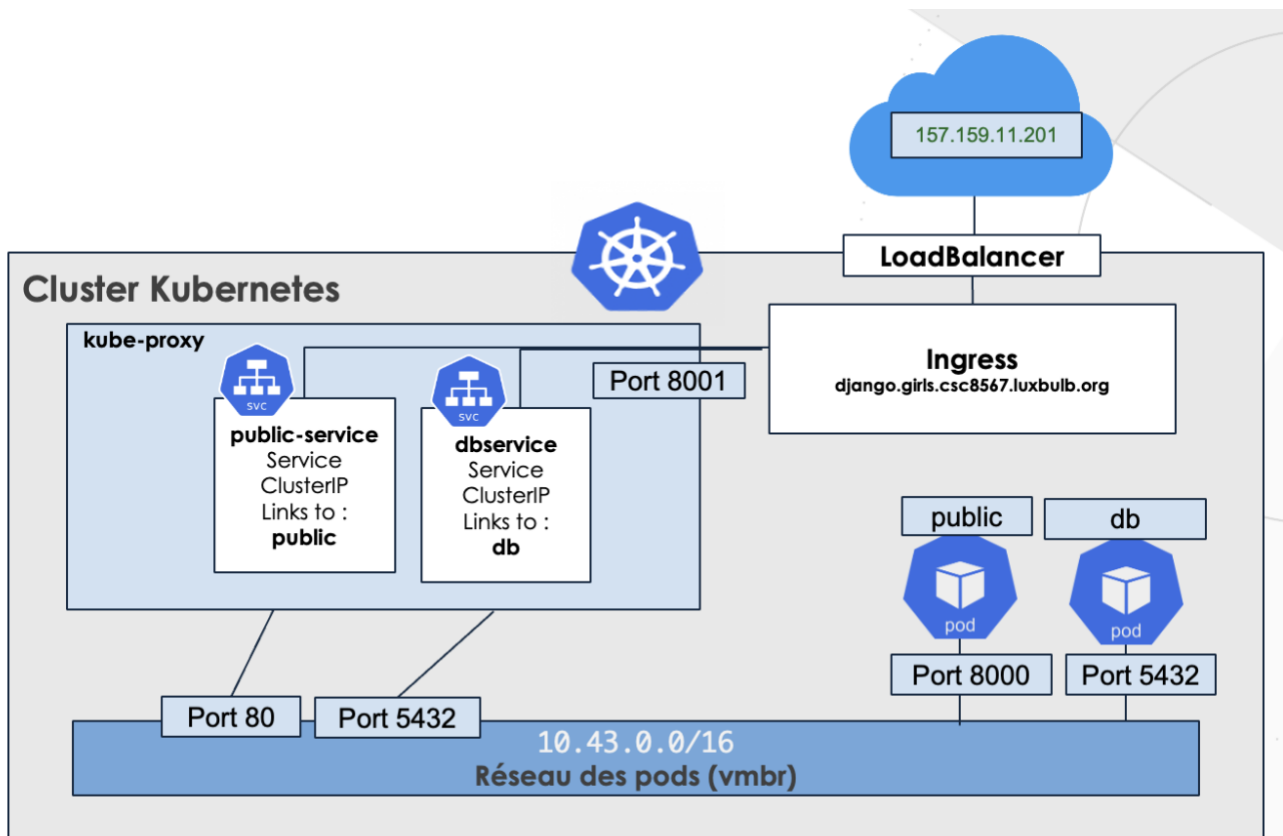


Figure 4: Schéma illustratif du défi 4

# Défi 5

**Objectif :** L'objectif principal de ce défi est de concevoir une infrastructure Kubernetes complète, adaptée à une application composée de plusieurs services indépendants. Cela inclut le déploiement séparé des applications API et Public, avec une gestion indépendante des pods, services, et ressources. La mise en place d'une base de données isolée, également déployée et gérée par Kubernetes.

## Étape 1 : Créer deux images Docker pour API et Public

On utilise les deux Dockerfile créées dans la première partie du module. Ensuite on construit les images Docker et les envoyer vers Docker Hub :

```
docker buildx build --platform linux/amd64,linux/arm64 -t xinjie19/kubeapi:v5 -f projet-webtsp/django-site/todolist/api/Dockerfile.api --push projet-web-tsp
docker buildx build --platform linux/amd64,linux/arm64 -t xinjie19/kubepub:v4 -f projet-webtsp/django-site/todolist/public/Dockerfile.front --push projet-web-tsp
```

```
projet-web-tsp > django-site > todolist > public > Dockerfile.front > ...
1 FROM python:3.8-alpine
2
3 # Set environment variables
4 ENV PYTHONUNBUFFERED=1
5
6 # Set the working directory
7 WORKDIR /projet-web-tsp/django-site/todolist
8
9 # Install system dependencies required for building packages like backports.zoneinfo
10 RUN apk add --no-cache gcc musl-dev python3-dev libffi-dev
11
12 # Copy the Django project files into the container
13 COPY ./django-site/todolist /projet-web-tsp/django-site/todolist
14
15 # Install Python dependencies
16 RUN pip install --upgrade pip && \
17     pip install -r ./requirements.txt
18
19 EXPOSE 8000
20
21 # Command to start the Django development server
22 CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
23
```

```
projet-web-tsp > django-site > todolist > api > Dockerfile.api > ...
1 FROM python:3.8-alpine
2
3 # Set environment variables
4 ENV PYTHONUNBUFFERED=1
5
6 # Set the working directory
7 WORKDIR /projet-web-tsp/django-site/todolist
8
9 # Install system dependencies required for building packages like backports.zoneinfo
10 RUN apk add --no-cache gcc musl-dev python3-dev libffi-dev
11
12 # Copy the Django project files into the container
13 COPY ./django-site/todolist /projet-web-tsp/django-site/todolist
14
15 # Install Python dependencies
16 RUN pip install --upgrade pip && \
17     pip install -r ./requirements.txt
18
19 # Exposer le port
20 EXPOSE 8000
21
22 # Commande pour démarrer l'API
23 CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
24
```

# Étape 2 : Déployer les Applications avec Kubernetes

## 1. Déploiement pour le Frontend Public et l'API

On crée un fichier Deployment `api-deploy.yaml` pour le le déploiement de l'API et un fichier Deployment `public-deploy.yaml` pour le le déploiement du frontend.

Pour les Deployments, on utilise 3 réplicas, avec les mêmes allocations/limitations de ressources que Défi 2 (`100m` de CPU, `100Mi` de mémoire pour les requêtes, et `200m`, `200Mi` pour les limites).

```
! api-deploy.yaml
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: api-deploy
5   namespace: u-cpt22
6 spec:
7   replicas: 3
8   selector:
9     matchLabels:
10      app: api
11   template:
12     metadata:
13       labels:
14         app: api
15     spec:
16       containers:
17       - name: api-container
18         image: xinjie19/kubeapi:v5
19         env:
20         - name: DB_NAME
21           value: "todolist"          # Nom de la base de données
22         - name: DB_USER
23           value: "xinjie"           # Nom d'utilisateur de la base de données
24         - name: DB_PASSWORD
25           value: "xinjie"           # Mot de passe de l'utilisateur
26         - name: DB_HOST
27           value: "dbservice"        # Nom du service de la base de données
28         - name: DB_PORT
29           value: "5432"             # Port de la base de données
30         command: ["sh", "-c", "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"]
31       resources:
32         requests:
33           cpu: "100m"
34           memory: "100Mi"
35         limits:
36           cpu: "200m"
37           memory: "200Mi"
38
```

```
! public-deploy.yaml
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: public-deploy
5   namespace: u-cpt22
6 spec:
7   replicas: 3
8   selector:
9     matchLabels:
10      app: public
11   template:
12     metadata:
13       labels:
14         app: public
15     spec:
16       containers:
17       - name: public-container
18         image: xinjie19/kubepub:v5
19         env:
20         - name: DB_NAME
21           value: "todolist"          # Nom de la base de données
22         - name: DB_USER
23           value: "xinjie"           # Nom d'utilisateur de la base de données
24         - name: DB_PASSWORD
25           value: "xinjie"           # Mot de passe de l'utilisateur
26         - name: DB_HOST
27           value: "dbservice"        # Nom du service de la base de données
28         - name: DB_PORT
29           value: "5432"             # Port de la base de données
30         command: ["sh", "-c", "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"]
31       resources:
32         requests:
33           cpu: "100m"
34           memory: "100Mi"
35         limits:
36           cpu: "200m"
37           memory: "200Mi"
38
```

## Projet Kubernetes - Girls

### 2. Déploiement pour les bases des données

On utilise les mêmes fichiers `dbdeploy.yaml` et `dbservice.yaml` qu'on a créés lors de Défi 4.

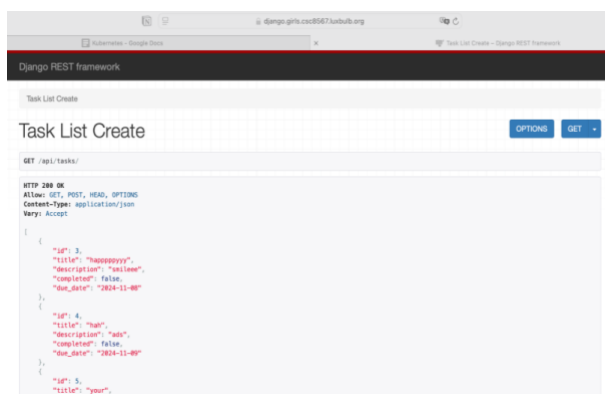
## Étape 3 : Configurer l'Ingress pour accéder aux applications

Ce fichier Ingress redirigera les requêtes vers `/api` et `/public` vers les services API et Public respectivement.

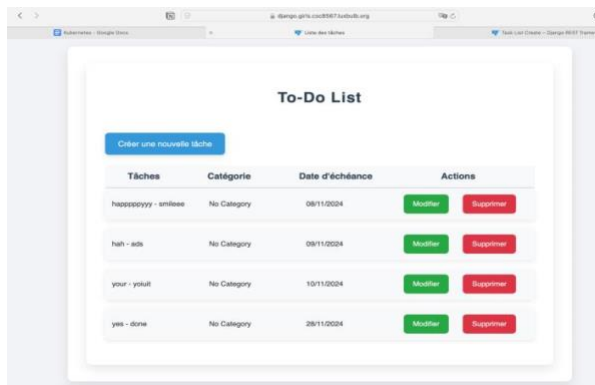
```
! ingress-pubapi.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: ingress-app
5    namespace: u-cptz2
6  spec:
7    rules:
8      - host: django.girls.csc8567.luxbulb.org
9        http:
10         paths:
11           - path: /api
12             pathType: Prefix
13             backend:
14               service:
15                 name: api-service
16                 port:
17                   number: 80
18           - path: /public
19             pathType: Prefix
20             backend:
21               service:
22                 name: public-service
23                 port:
24                   number: 80
25           - path: /static
26             pathType: Prefix
27             backend:
28               service:
29                 name: public-service
30                 port:
31                   number: 80
32         tls:
33           - hosts:
34             - django.girls.csc8567.luxbulb.org
35             secretName: tls-secret
36
```

## Étape 4 : Accéder au site web

- Pour accéder à l'API : <https://django.girls.csc8567.luxbulb.org/api>
- Pour accéder au frontend public : <https://django.girls.csc8567.luxbulb.org/public>



## Projet Kubernetes - Girls



On vérifie les services :

```
xinjie@MacBook-XinJie .k8s % kubectl get pods -n u-cptz2
NAME                                READY   STATUS    RESTARTS   AGE
api-deploy-8446c9dbd9-2g7zs        1/1     Running   0           18m
api-deploy-8446c9dbd9-g5vrz        1/1     Running   0           18m
api-deploy-8446c9dbd9-qd6xk        1/1     Running   0           18m
db-deploy-75db678446-5m5k5         1/1     Running   0           19d
public-deploy-7d6ff8bf5b-4wqhl     1/1     Running   0           19m
public-deploy-7d6ff8bf5b-8rh66     1/1     Running   0           19m
public-deploy-7d6ff8bf5b-k8zjn     1/1     Running   0           19m
xinjie@MacBook-XinJie .k8s %
xinjie@MacBook-XinJie .k8s % kubectl get services -n u-cptz2
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
api-service         ClusterIP   10.43.189.153 <none>        80/TCP     176m
app-service         ClusterIP   10.43.42.223  <none>        80/TCP     19d
csc8567-service     ClusterIP   10.43.107.40  <none>        80/TCP     11d
db-service          ClusterIP   10.43.8.162   <none>        5432/TCP   19d
postgres-service    ClusterIP   10.43.186.212 <none>        5432/TCP   19d
public-service      ClusterIP   10.43.182.249 <none>        80/TCP     175m
servicedjango       ClusterIP   10.43.51.96   <none>        80/TCP     19d
xinjie@MacBook-XinJie .k8s %
xinjie@MacBook-XinJie .k8s % kubectl get ingress -n u-cptz2
NAME          CLASS    HOSTS                                ADDRESS          PORTS    AGE
ingress-app   nginx    django.girls.csc8567.luxbulb.org     157.159.11.201  80, 443  173m
ingress-django nginx    django.girls.csc8567.luxbulb.org     157.159.11.201  80, 443  4d23h
```

Le schéma :

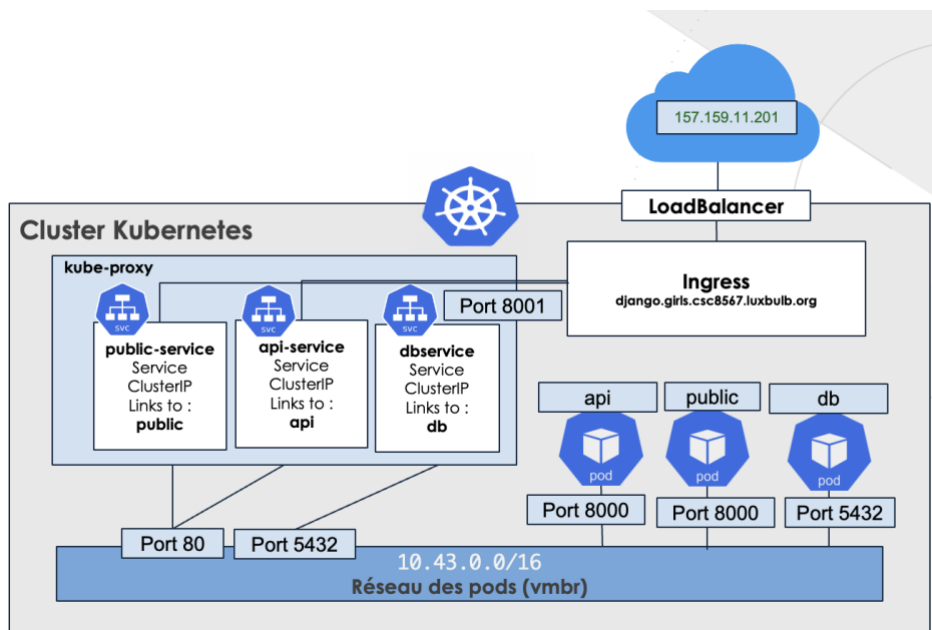


Figure 5: Schéma illustratif du défi 5

## Défi 6

**Objectif :** L'objectif principal de ce défi est d'automatiser le déploiement de l'infrastructure réalisée au Défi 5 grâce à Helm. Helm est un outil qui facilite la gestion des applications Kubernetes en les emballant dans des "charts". En plus, on utilise ConfigMaps pour stocker et gérer des informations de configuration telles que les détails de connexion à la base de données (DB\_NAME, DB\_USER, etc.).

### Étape 1 : Créer une chart Helm

On crée une structure de fichiers pour le chart Helm dans le dossier `django-chart`.

```
xinjie@MacBook-XinJie .kube % helm create django-chart
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /Users/xinjie/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /Users/xinjie/.kube/config
Creating django-chart
```

Après avoir créé le chart avec la commande `helm create`, les fichiers ci-dessous sont créés automatiquement dans le dossier `django-chart` :

```
xinjie@MacBook-XinJie .kube % tree django-chart
django-chart
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── api-deployment.yaml
│   ├── configmap.yaml
│   ├── db-deployment
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── public-deployment.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

4 directories, 13 files
```

### Étape 2 : Modifier le chart Helm

#### 1. Configurer les déploiements (API, Public et DB)

Dans `templates/api-deployment.yaml`, `public-deployment.yaml` et `db-deployment.yaml`, on crée des déploiements séparés pour l'API et le frontend public et la base des données, basés sur les configurations dynamiques de `values.yaml`.



public-deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.deployments.public.name }}
  namespace: {{ .Values.namespace }}
  labels:
    app: {{ .Values.deployments.public.appLabel }}
    helm.sh/chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    app.kubernetes.io/name: public
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: {{ .Values.deployments.public.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.deployments.public.appLabel }}
  template:
    metadata:
      labels:
        app: {{ .Values.deployments.public.appLabel }}
    spec:
      containers:
        - name: public-container
          image: {{ .Values.deployments.public.image }}
          env:
            - name: DB_NAME
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_NAME
            - name: DB_USER
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_USER
            - name: DB_USER
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_USER
            - name: DB_HOST
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_HOST
            - name: DB_PORT
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_PORT
            - name: DB_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_PASSWORD
          command:
            - sh
            - -c
            - python manage.py migrate && python manage.py runserver 0.0.0.0:8000
          resources:
            requests:
              cpu: {{ .Values.deployments.public.resources.requests.cpu }}
              memory: {{ .Values.deployments.public.resources.requests.memory }}
            limits:
              cpu: {{ .Values.deployments.public.resources.limits.cpu }}
              memory: {{ .Values.deployments.public.resources.limits.memory }}
```



## Projet Kubernetes - Girls

api-deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.deployments.api.name }}
  namespace: {{ .Values.namespace }}
  labels:
    app: {{ .Values.deployments.api.appLabel }}
    helm.sh/chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    app.kubernetes.io/name: api
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: {{ .Values.deployments.api.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.deployments.api.appLabel }}
  template:
    metadata:
      labels:
        app: {{ .Values.deployments.api.appLabel }}
    spec:
      containers:
        - name: api-container
          image: {{ .Values.deployments.api.image }}
          env:
            - name: DB_NAME
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: DB_NAME
```

```
        - name: DB_USER
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: DB_USER
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: DB_HOST
        - name: DB_PORT
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: DB_PORT
        - name: DB_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: DB_PASSWORD
      command:
        - sh
        - -c
        - python manage.py migrate && python manage.py runserver 0.0.0.0:8000
      resources:
        requests:
          cpu: {{ .Values.deployments.api.resources.requests.cpu }}
          memory: {{ .Values.deployments.api.resources.requests.memory }}
        limits:
          cpu: {{ .Values.deployments.api.resources.limits.cpu }}
          memory: {{ .Values.deployments.api.resources.limits.memory }}
```

## Projet Kubernetes - Girls

db-deployment.yaml :

django-chart > templates > db-deployment.yaml > {map} >

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: {{ .Values.deployments.db.name }}
5    namespace: {{ .Values.namespace }}
6    labels:
7      app: {{ .Values.deployments.db.appLabel }}
8  spec:
9    replicas: {{ .Values.deployments.db.replicas }}
10   selector:
11     matchLabels:
12       app: {{ .Values.deployments.db.appLabel }}
13   template:
14     metadata:
15       labels:
16         app: {{ .Values.deployments.db.appLabel }}
17     spec:
18       containers:
19         - name: postgres
20           image: {{ .Values.deployments.db.image }}
21           env:
22             - name: POSTGRES_DB
23               valueFrom:
24                 configMapKeyRef:
25                   name: db-config
26                   key: DB_NAME
```

```
    - name: POSTGRES_USER
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_USER
    - name: POSTGRES_PASSWORD
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_PASSWORD
  ports:
    - containerPort: 5432
  volumeMounts:
    - name: postgres-storage
      mountPath: /var/lib/postgresql/data
  volumes:
    - name: postgres-storage
      emptyDir: {}
```

### 2. Configurer les services (API et Public et DB)

On modifie `templates/service.yaml` pour inclure les services nécessaires, en fonction des variables `values.yaml`.

```
django-chart > templates > service.yaml > ...

1  # API Service
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: {{ .Values.deployments.api.serviceName }}
6    namespace: {{ .Values.namespace }}
7  spec:
8    selector:
9      app: {{ .Values.deployments.api.appLabel }}
10   ports:
11     - protocol: TCP
12       port: 80
13       targetPort: 8000
14   type: ClusterIP
15 ---
16 # Public Service
17 apiVersion: v1
18 kind: Service
19 metadata:
20   name: {{ .Values.deployments.public.serviceName }}
21   namespace: {{ .Values.namespace }}
22 spec:
23   selector:
24     app: {{ .Values.deployments.public.appLabel }}
25   ports:
26     - protocol: TCP
27       port: 80
28       targetPort: 8000
29   type: ClusterIP
30 ---
31 # DB Service
32 apiVersion: v1
33 kind: Service
34 metadata:
35   name: {{ .Values.deployments.db.serviceName }}
36   namespace: {{ .Values.namespace }}
37 spec:
38   selector:
39     app: {{ .Values.deployments.db.appLabel }}
40   ports:
41     - protocol: TCP
42       port: 5432
43       targetPort: 5432
44   type: ClusterIP
45
```

### 3. Configurer l'Ingress

On ajoute les routes pour accéder à l'API et au frontend public dans `templates/ingress.yaml`.

```
django-chart > templates > ingress.yaml > spec > rules > http > paths > - path: /api
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: {{ .Values.ingress.name }}
5    namespace: {{ .Values.namespace }}
6  spec:
7    rules:
8      - host: {{ .Values.ingress.host }}
9      http:
10         paths:
11           - path: /api
12             pathType: Prefix
13             backend:
14               service:
15                 name: {{ .Values.deployments.api.serviceName }}
16                 port:
17                   number: 80
18           - path: /public
19             pathType: Prefix
20             backend:
21               service:
22                 name: {{ .Values.deployments.public.serviceName }}
23                 port:
24                   number: 80
25           - path: /static
26             pathType: Prefix
27             backend:
28               service:
29                 name: {{ .Values.deployments.public.serviceName }}
30                 port:
31                   number: 80
32         tls:
33           - hosts:
34             - {{ .Values.ingress.host }}
35             secretName: {{ .Values.ingress.tlsSecret }}
36
```

### 4. Ajouter un ConfigMap pour les variables de la base de données

On crée `templates/configmap.yaml` pour stocker les informations de connexion :

```
django-chart > templates > configmap.yaml > {map}
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: db-config
5    namespace: {{ .Values.namespace }}
6  data:
7    DB_NAME: "{{ .Values.database.name }}"
8    DB_USER: "{{ .Values.database.user }}"
9    DB_HOST: "{{ .Values.database.host }}"
10   DB_PORT: "{{ .Values.database.port }}"
11   DB_PASSWORD: "{{ .Values.database.password }}"
12
```

### Étape 3 : Modifier le fichier values.yaml

On ajoute toutes les variables nécessaires dans le fichier `values.yaml`.

```
django-chart > ! values.yaml > {} deployments > {} {
1 namespace: u-cptz2
2
3
4 database:
5   name: todoist
6   user: xinjie
7   password: xinjie
8   host: dbservice
9   port: "5432"
10
11 deployments:
12   api:
13     name: api-deploy
14     appLabel: api
15     image: xinjie19/kubeapi:v5
16     replicas: 3
17     resources:
18       requests:
19         cpu: "100m"
20         memory: "100Mi"
21       limits:
22         cpu: "200m"
23         memory: "200Mi"
24     serviceName: api-service
25
26   public:
27     name: public-deploy
28     appLabel: public
29     image: xinjie19/kubepub:v5
30     replicas: 3
31     resources:
32       requests:
33         cpu: "100m"
34         memory: "100Mi"
35       limits:
36         cpu: "200m"
37         memory: "200Mi"
38     serviceName: public-service
39
40 db:
41   name: db-deploy
42   appLabel: postgres
43   replicas: 1
44   image: postgres:13
45   serviceName: db-service
46   resources:
47     requests:
48       cpu: "100m"
49       memory: "100Mi"
50     limits:
51       cpu: "200m"
52       memory: "200Mi"
53
54 ingress:
55   name: ingress-app
56   host: django.girls.csc8567.luxbulb.org
57   tlsSecret: tls-secret
58
59 service:
60   type: ClusterIP
61   port: 80
62   name: django-service
63
64 serviceAccount:
65   create: true
66   name: ""
67   automount: true
68   annotations: {}
69
70 autoscaling:
71   enabled: false
72   minReplicas: 1
73   maxReplicas: 5
74   targetCPUUtilizationPercentage: 80
75   targetMemoryUtilizationPercentage: 75
76
77 serviceaccount:
78   create: false
79   name: default
80   automount: true
81 }
```

### Étape 4 : Installer le chart Helm

Pour installer le chart Helm :

```
helm install django-girls ./django-chart -n u-cptz2
```

```
xinjie@MacBook-XinJie .k8e % helm install django-girls ./django-chart -n u-cptz2
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /Users/xinjie/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /Users/xinjie/.kube/config
NAME: django-girls
LAST DEPLOYED: Sun Nov 17 01:52:12 2024
NAMESPACE: u-cptz2
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace u-cptz2 -l "app.kubernetes.io/name=django-chart,app.kubernetes.io/instance=django-girls" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace u-cptz2 $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace u-cptz2 port-forward $POD_NAME 8080:$CONTAINER_PORT
```



## Projet Kubernetes - Girls

```
xinjie@MacBook-XinJie .kube % kubectl get all -n u-cptz2
```

NAME	READY	STATUS	RESTARTS	AGE
pod/api-deploy-55dfd477bb-4rw2c	1/1	Running	1 (11h ago)	11h
pod/api-deploy-55dfd477bb-9j85t	1/1	Running	1 (11h ago)	11h
pod/api-deploy-55dfd477bb-x8jjz	1/1	Running	1 (11h ago)	11h
pod/db-deploy-7d9f955bfb-2vk29	1/1	Running	0	11h
pod/public-deploy-76d57db47c-59ls5	1/1	Running	0	11h
pod/public-deploy-76d57db47c-c9222	1/1	Running	1 (11h ago)	11h
pod/public-deploy-76d57db47c-gjbs5	1/1	Running	1 (11h ago)	11h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/api-service	ClusterIP	10.43.39.61	<none>	80/TCP	11h
service/app-service	ClusterIP	10.43.42.223	<none>	80/TCP	27d
service/csc8567-service	ClusterIP	10.43.107.40	<none>	80/TCP	19d
service/db-service	ClusterIP	10.43.44.88	<none>	5432/TCP	11h
service/dbservice	ClusterIP	10.43.8.162	<none>	5432/TCP	27d
service/postgres-service	ClusterIP	10.43.186.212	<none>	5432/TCP	27d
service/public-service	ClusterIP	10.43.238.121	<none>	80/TCP	11h
service/servicedjango	ClusterIP	10.43.51.96	<none>	80/TCP	27d

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/api-deploy	3/3	3	3	11h
deployment.apps/app-deployment	0/0	0	0	27d
deployment.apps/csc8567-deployment	0/0	0	0	19d
deployment.apps/db-deploy	1/1	1	1	11h
deployment.apps/dbdeploy	0/0	0	0	27d
deployment.apps/deploy1	0/0	0	0	32d
deployment.apps/deploydjango	0/0	0	0	27d
deployment.apps/public-deploy	3/3	3	3	11h

```
xinjie@MacBook-XinJie .kube % kubectl get deployments -n u-cptz2
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
app-deployment	0/0	0	0	27d
csc8567-deployment	0/0	0	0	19d
dbdeploy	0/0	0	0	27d
deploy1	0/0	0	0	32d
deploydjango	0/0	0	0	27d

```
xinjie@MacBook-XinJie .kube % helm install django-girls ./django-chart -n u-cptz2
```

```
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /Users/xinjie/.kube/config
```

```
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /Users/xinjie/.kube/config
```

```
NAME: django-girls
```

```
LAST DEPLOYED: Tue Nov 19 20:35:36 2024
```

```
NAMESPACE: u-cptz2
```

```
STATUS: deployed
```

```
REVISION: 1
```

```
NOTES:
```

```
1. Get the application URL by running these commands:
```

```
export POD_NAME=$(kubectl get pods --namespace u-cptz2 -l "app.kubernetes.io/name=django-chart,app.kubernetes.io/instance=django-girls" -o jsonpath="{.items[0].metadata.name}")
```

```
export CONTAINER_PORT=$(kubectl get pod --namespace u-cptz2 $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
```

```
echo "Visit http://127.0.0.1:8080 to use your application"
```

```
kubectl --namespace u-cptz2 port-forward $POD_NAME 8080:$CONTAINER_PORT
```

```
xinjie@MacBook-XinJie .kube %
```

```
xinjie@MacBook-XinJie .kube % kubectl get deployments -n u-cptz2
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
api-deploy	3/3	3	3	14s
app-deployment	0/0	0	0	27d
csc8567-deployment	0/0	0	0	19d
db-deploy	1/1	1	1	14s
dbdeploy	0/0	0	0	27d
deploy1	0/0	0	0	32d
deploydjango	0/0	0	0	27d
public-deploy	3/3	3	3	14s

## Étape 5 : Accéder au site web

On peut maintenant accéder aux sites via les liens suivants :

- API : <https://django.girls.csc8567.luxbulb.org/api>
- Frontend Public : <https://django.girls.csc8567.luxbulb.org/public>

## Conclusion

Au terme de ce projet Kubernetes, notre groupe a acquis une solide compréhension de la gestion des applications dans un environnement Kubernetes. Grâce à une série de défis pratiques, nous avons appris à déployer des applications web simples dans des Pods, à gérer des répliques avec des Deployments, et à exposer nos services via des ClusterIP ou Ingress pour rendre les applications accessibles à l'intérieur ou à l'extérieur du cluster.

En particulier, le projet a permis d'approfondir notre maîtrise des concepts essentiels de Kubernetes, tels que la gestion des ressources (CPU, mémoire), l'utilisation de Docker pour containeriser des applications, et l'automatisation du déploiement via Helm. Nous avons également travaillé avec des bases de données, configuré des services et utilisé des variables d'environnement pour relier différents composants d'une application, comme Django et PostgreSQL.

Les étapes pratiques comme la mise en place d'un Ingress pour rendre notre application accessible sur Internet, ainsi que l'utilisation de Helm pour automatiser le déploiement, ont constitué des moments clés pour renforcer nos compétences en gestion de l'infrastructure Kubernetes. Grâce à ce projet, nous avons non seulement appris à configurer des applications complexes, mais aussi à optimiser leur gestion et leur scalabilité dans un environnement de production.

# Annexes

## GitHub Défi :

<https://github.com/DF-Telecom-SudParis/CSC8567-Final>

## Site web gestion de cluster :

<https://csc8567.luxbulb.org/dashboard/auth/login>

Login : Girls

Mdp : YosXinWeulyFisa2024@

Namespace : u-cptz2

## GitHub partage codes :

<https://github.com/xinjiee19/kubernetes-tsp/tree/main>

# Table des Illustrations

Figure 1: Schéma illustratif du défi 1 .....	4
Figure 2: Schéma illustratif du défi 2 .....	8
Figure 3: Schéma illustratif du défi 3 .....	14
Figure 4: Schéma illustratif du défi 4 .....	17
Figure 5: Schéma illustratif du défi 5 .....	21