

Rapport de déploiement Kubernetes - Projet IC GROUP

Contexte

La société IC GROUP veut mettre en place un site vitrine avec accès direct à deux applications internes principales : Odoo 13.0 pour la gestion d'entreprise, et pgAdmin pour administrer PostgreSQL. Le but était de conteneuriser ces apps, les déployer dans Minikube avec persistance des données et lien via une webapp codée en Flask. Tout devait être opérationnel dans un namespace Kubernetes spécifique, avec une bonne séparation des ressources et du code.

Conteneurisation de la webapp Flask

J'ai commencé par cloner le repo du site vitrine :

```
cours_helm@helm-LOV:~$ mkdir mini-projet-kube
cours_helm@helm-LOV:~$ cd mini-projet-kube/
cours_helm@helm-LOV:~/mini-projet-kube$ git clone https://github.com/sadofrazer/ic-webapp.git
Clonage dans 'ic-webapp'...
remote: Enumerating objects: 96, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 96 (delta 32), reused 32 (delta 32), pack-reused 57 (from 1)
Dépaquetage des objets: 100% (96/96), 1.14 Mio | 8.74 Mio/s, fait.
cours_helm@helm-LOV:~/mini-projet-kube$ cd ic-webapp
```

Ensuite j'ai créé un Dockerfile avec l'image de base python:3.6-alpine, mis /opt comme répertoire de travail, installé Flask avec pip, exposé le port 8080 et ajouté les variables d'environnement ODOO_URL et PGADMIN_URL. Enfin, j'ai mis en ENTRYPOINT le lancement de app.py.

```

Dockerfile
1 FROM python:3.6-alpine
2
3 ENV FLASK_ENV=production \
4     PYTHONDONTWRITEBYTECODE=1 \
5     PYTHONUNBUFFERED=1 \
6     ODOO_URL=https://www.odoo.com \
7     PGADMIN_URL=https://www.pgadmin.org
8
9 RUN addgroup -S webgroup && adduser -S webapp -G webgroup
10
11 WORKDIR /opt
12 COPY . /opt
13 RUN chown -R webapp:webgroup /opt
14
15 RUN pip install --no-cache-dir Flask
16
17 EXPOSE 8080
18
19 USER webapp
20
21 ENTRYPOINT ["python"]
22 CMD ["app.py"]
23
24

```

```

cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ vim Dockerfile
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ docker build -t ic-webapp:1.0 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  2.891MB
Step 1/11 : FROM python:3.6-alpine
3.6-alpine: Pulling from library/python
59bf1c3509f3: Pull complete
3786870f2876: Pull complete
9acb0e804800e: Pull complete
52bedcb3e853: Pull complete
b064415ed3d7: Pull complete
Digest: sha256:579978dec4602646fe1262f02b96371779bfb0294e92c91392707fa999c0c989

```

Après build de l'image, j'ai préparé le push vers Docker Hub. J'ai vérifié que j'étais bien connecté :

```

cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/
Username: m4daa9
Password:
WARNING! Your password will be stored unencrypted in /home/cours_helm/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded

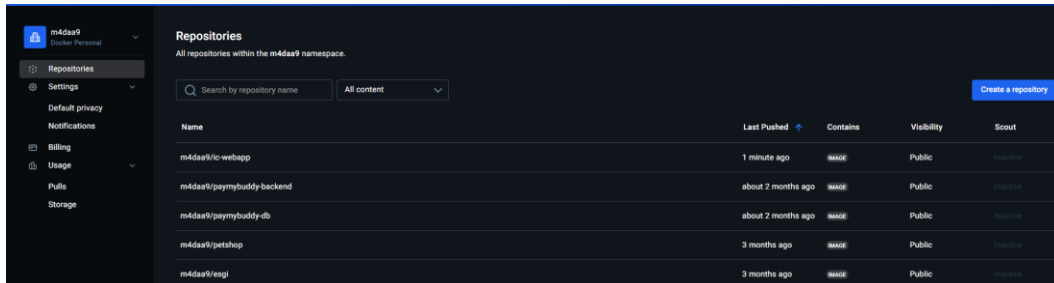
```

Puis j'ai push l'image :

```

cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ docker tag ic-webapp:1.0 m4daa9/ic-webapp:1.0
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ docker push m4daa9/ic-webapp:1.0
The push refers to repository [docker.io/m4daa9/ic-webapp]
7da42b2e99c6: Pushed
5fff586cdc4c: Pushed
b593aa325cb1: Pushed
d4124910f5de: Pushed
3156423bd38f: Mounted from library/python
efa76becf38b: Mounted from library/python
671e3248113c: Mounted from library/python
1965cfbef2ab: Mounted from library/python
8d3ac3489996: Mounted from library/python
1.0: digest: sha256:aaa675ab0e0b55ae8dd81ca41c2f197595f6296d0621843495522a082dba38ca size: 2210
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$

```



The screenshot shows the Docker Hub 'Repositories' page for the user 'm4daa9'. The page lists several repositories, including 'm4daa9/ic-webapp', 'm4daa9/paymybuddy-backend', 'm4daa9/paymybuddy-db', 'm4daa9/petshop', and 'm4daa9/regi'. Each repository entry shows its last push time, a 'Contains' button, its visibility (Public), and a 'Scout' button.

Name	Last Pushed	Contains	Visibility	Scout
m4daa9/ic-webapp	1 minute ago	image	Public	inactive
m4daa9/paymybuddy-backend	about 2 months ago	image	Public	inactive
m4daa9/paymybuddy-db	about 2 months ago	image	Public	inactive
m4daa9/petshop	3 months ago	image	Public	inactive
m4daa9/regi	3 months ago	image	Public	inactive

Test local du conteneur

J'ai testé l'application en local avec la commande docker run avec les variables d'environnement définies et le port 8080 exposé. Cela a permis de vérifier que la webapp redirigeait bien vers Odoo et pgAdmin.

```

cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp$ docker run -d --name test-ic-webapp \
> -e OD00_URL=https://www.odoo.com \
> -e PGADMIN_URL=https://www.pgadmin.org \
> -p 8080:8080 ic-webapp:1.0
a1f8e4e49c3e76d412ad3eef77d8affd268536e84bd4116b4c81b7596fa4f1d4

```

Déploiement Kubernetes

Une fois tout prêt, j'ai appliqué les fichiers YAML avec kubectl. Le namespace icgroup est bien utilisé, toutes les ressources ont le label env=prod.

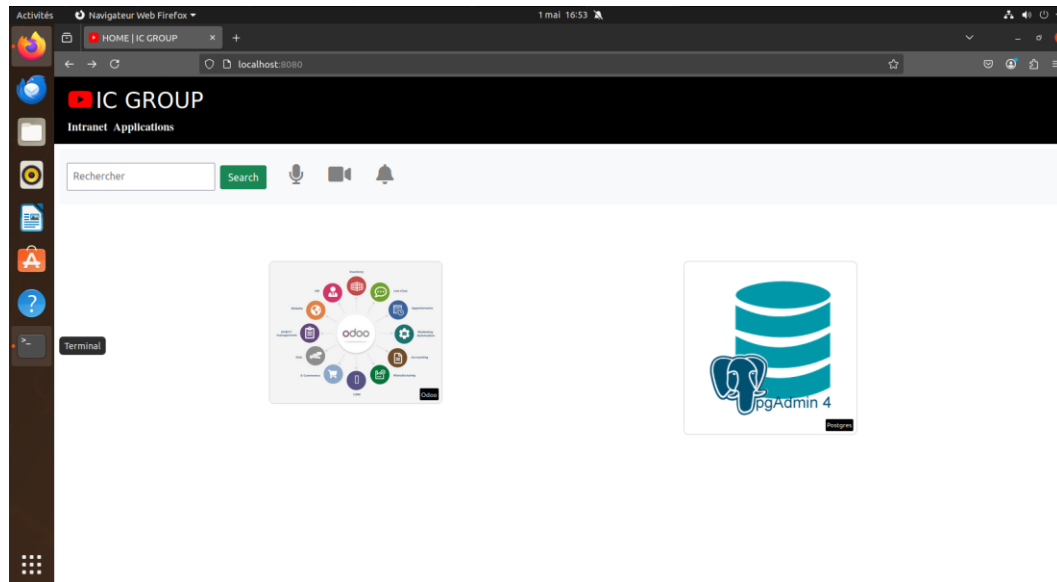
```

cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f secret/namespace.yaml
namespace/icgroup created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f secret/secret.yaml
secret/db-secret created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f secret/configmap-webapp.yaml
configmap/webapp-config created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f secret/pgadmin-servers-configmap.yaml
configmap/pgadmin-servers created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f postgres/pvc.yaml
persistentvolumeclaim/postgres-pvc created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f postgres/deployment.yaml
deployment.apps/postgres created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f postgres/service.yaml
service/postgres created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f odoo/pvc.yaml
persistentvolumeclaim/odoo-pvc created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f odoo/deployment.yaml
deployment.apps/odoo created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f odoo/service.yaml
service/odoo created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f pgadmin/pvc.yaml
persistentvolumeclaim/pgadmin-pvc created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f pgadmin/deployment.yaml
deployment.apps/pgadmin created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f pgadmin/service.yaml
service/pgadmin created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f ic-webapp/deployment.yaml
deployment.apps/ic-webapp created
cours_helm@helm-LOV:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl apply -f ic-webapp/service.yaml
service/ic-webapp created

```

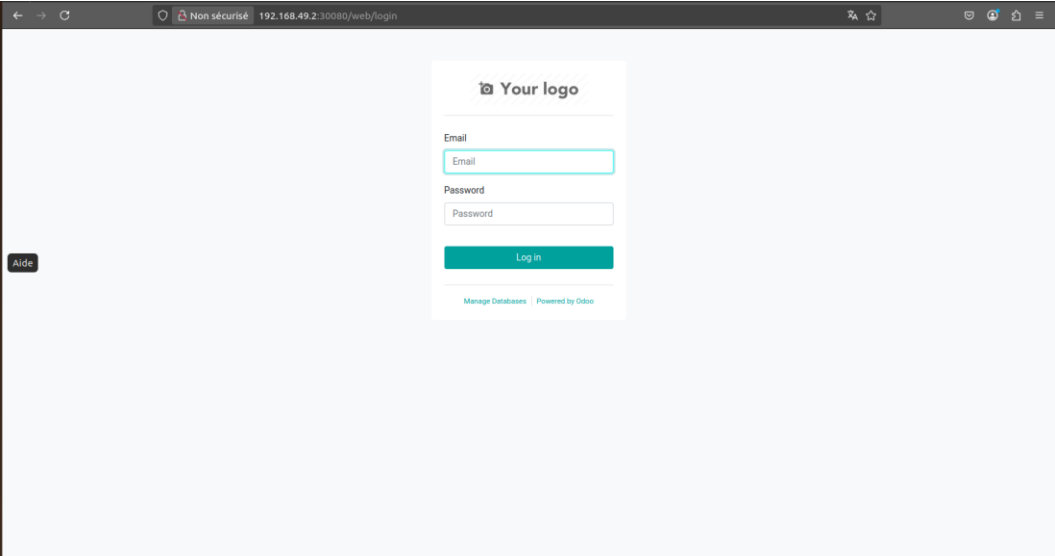
Webapp vitrine

Une fois déployée dans Kubernetes, la webapp s'affiche bien avec les liens dynamiques.



Déploiement de Odoo

Odoo est bien fonctionnel, j'ai pu accéder à l'interface et créer une base.



Create Database



Master Password

.....

Database Name

victor_database

Email

vlemoine8@myges.fr

Password

.....



Phone number

Language

English (US)



Country

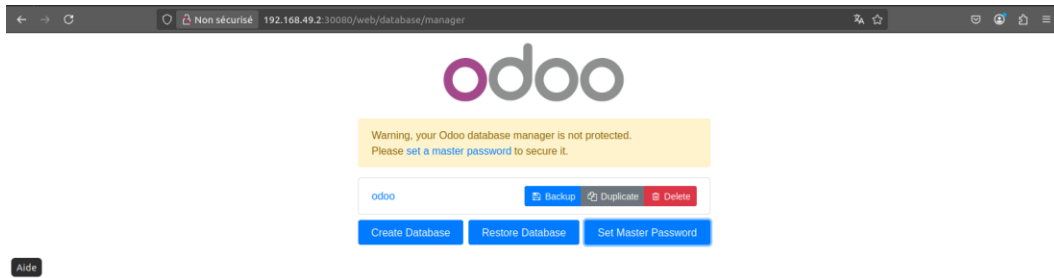


Demo data



To enhance your experience, some data may be sent to Odoo online services.
See our [Privacy Policy](#).

Continue



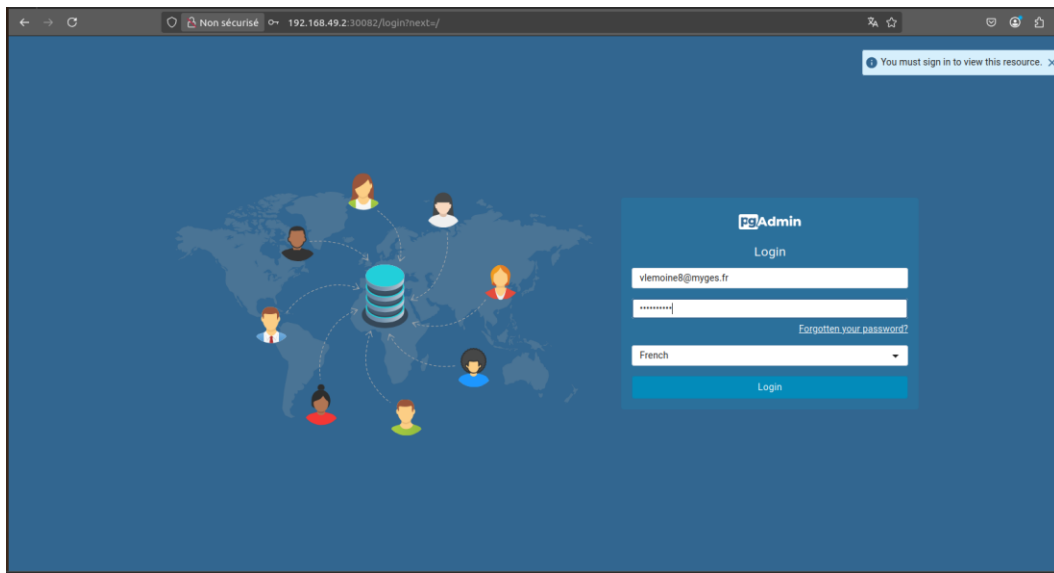
```
cours_helme@helme-L0V:~/mini-projet-kube/ic-webapp/Kubernetes$ kubectl exec -n icgroup -it deploy/postgres -- psql -U ESGI -c '\l'
```

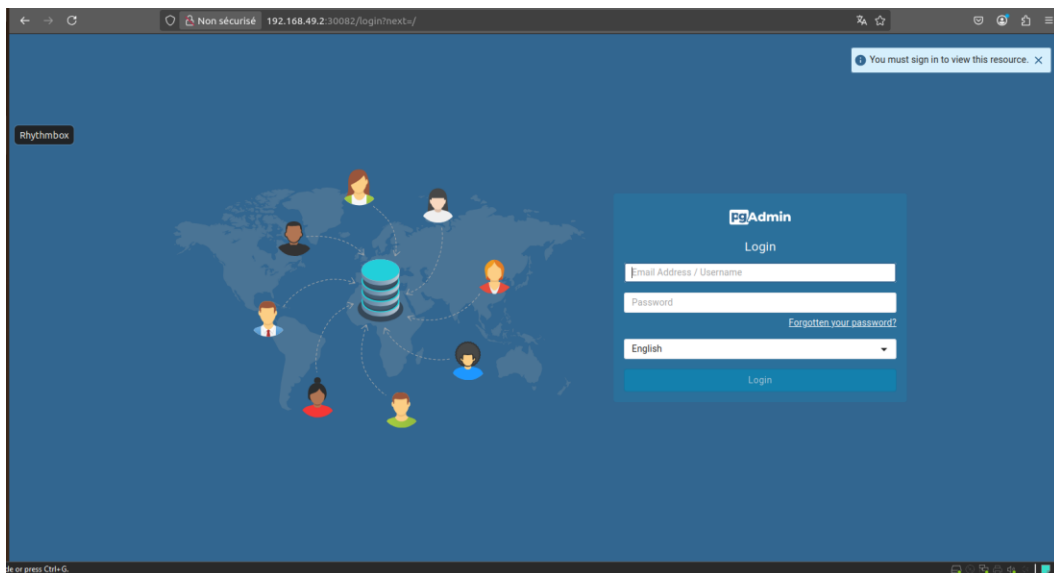
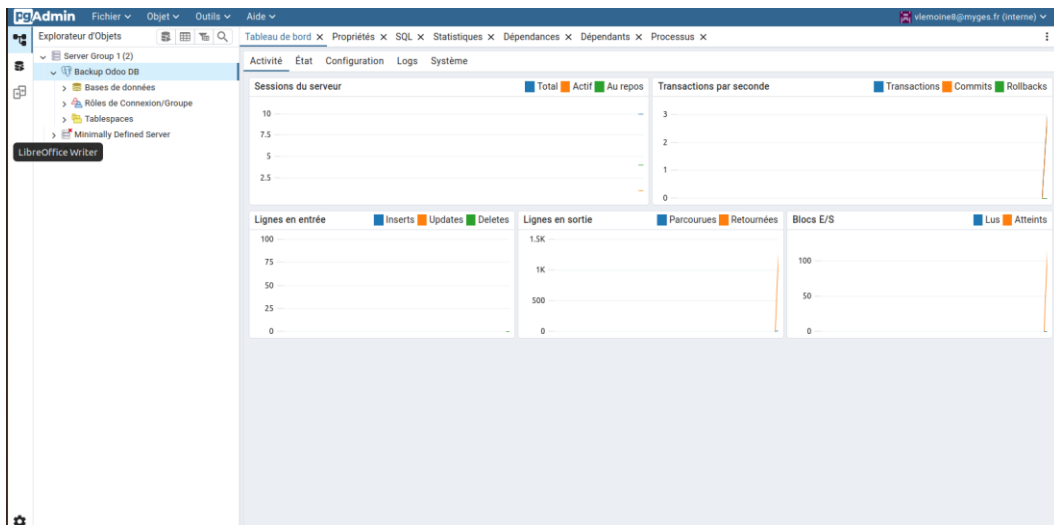
Name	Owner	Encoding	Collate	Ctype	Access privileges
ESGI	ESGI	UTF8	en_US.utf8	en_US.utf8	
database_test	ESGI	UTF8	C	en_US.utf8	
odoo	ESGI	UTF8	C	en_US.utf8	=Tc/ESGI + ESGI=CTc/ESGI + odoo=CTc/ESGI
postgres	ESGI	UTF8	en_US.utf8	en_US.utf8	=c/ESGI + ESGI=CTc/ESGI
template0	ESGI	UTF8	en_US.utf8	en_US.utf8	=c/ESGI + ESGI=CTc/ESGI
template1	ESGI	UTF8	en_US.utf8	en_US.utf8	=c/ESGI + ESGI=CTc/ESGI
test_database	ESGI	UTF8	C	en_US.utf8	
victor_database	ESGI	UTF8	C	en_US.utf8	

(8 rows)

Déploiement de pgAdmin

J'ai ensuite testé pgAdmin. L'interface se lance bien et permet de visualiser les bases PostgreSQL.





Conclusion

Le projet est totalement fonctionnel :

- Webapp Flask conteneurisée et déployée avec des variables dynamiques
- Odoo 13.0 opérationnel avec persistance de données
- pgAdmin configuré pour se connecter à la base Odoo automatiquement
- Tout le déploiement est fait dans le namespace icgroup avec les bons labels

Toutes les étapes demandées dans le sujet ont été respectées. Le cluster est propre et chaque service est accédé via interface graphique comme prévu.