



Service informatique
OSU PYTHÉAS

Infrastructure Docker et intégration continue à l'OSU Pythéas

WWW.OSUPYTHEAS.FR

Alrick Dias

amU Aix
Marseille
Université



INRAE
la science pour la vie, l'humain, la terre


OSU PYTHÉAS

Contexte

10 ASR pour 1500 usagers

De plus en plus de projets de recherche demandent de l'hébergement d'appli web (environ 50 applis actuellement)

Dev par des stagiaires ou CDD > maintenance pas assuré

Plusieurs VM donc supervision compliquée

Besoin de réorganisation

Réduire le nombre de VM

Limiter les risques de sécurité liées à la maintenance

Autonomiser les développeurs sur le déploiement

Mieux cloisonner les rôles

- Les ASR gèrent les VM
- Les dev gèrent les applis dans leur ensemble

La réponse classique : Docker

Mais concrètement comment on fait ?

Qu'est ce que ça implique pour les Dev ?

Et d'un point de vue Admin Sys ?

Les indispensables

En plus d'un serveur docker configuré

Côté Dev : Contrainte technique principale

Contrainte :

- Une appli web = 3 services minimum (frontend, API, BDD)
- Un container par service
- Container isolé sauf si on expose un port
- Une BDD ne doit pas être atteignable depuis l'extérieur
- Mais les trois services doivent communiquer entre eux

Solution :

Utiliser les réseaux virtuels docker...

mais ça peut vite devenir complexe

Côté Dev : Les stacks Docker

Permet de cloisonner les containers dans un réseau virtuel

On ouvre qu'un container depuis l'extérieur (le front ou un proxy pour les applis plus complexe)

Création de sous réseaux dans la stack possible

Les containers n'utilisent pas une URL publique mais leur nom de service pour communiquer

Côté Dev : Les stacks Docker

fatercal-api	running	    	fatercal	registry.osupytheas.fr:443/adias/fatercal/api:main	2026-05-22 15:51:23	10.99.37.4	-
fatercal-cron	running	    	fatercal	registry.osupytheas.fr:443/adias/fatercal/cron:main	2026-05-22 15:51:24	10.99.37.5	-
fatercal-db	running	    	fatercal	postgres:18.3	2026-05-22 15:51:23	10.99.37.2	-
fatercal-web	running	    	fatercal	registry.osupytheas.fr:443/adias/fatercal/webapp:main	2026-05-22 15:51:23	10.99.37.3	 8200:80



Côté Dev : Docker compose

Permet de lancer tous les containers d'une application en une commande '*docker compose up*'

Crée une stack avec un réseau virtuel spécifique

Déclaratif donc accessible pour les dev

Permet une mise en production simplifié et rapide

Côté Dev : Docker compose

Un seul service ouvert (le front)

Variables d'environnements pour usage par les services

Réseau virtuel pour la stack

Versions des images fixées

Volumes pour persistance des données

```
1  services:
2    front:
3      image: registry.osupytheas.fr:443/monapp-front:latest
4      container_name: monapp-front
5      ports:
6        - "8080:80"
7      networks:
8        - monapp-network
9    api:
10     image: registry.osupytheas.fr:443/monapp-api:latest
11     container_name: monapp-api
12     environment:
13       DB_HOST: bdd
14       DB_USER: monapp
15       DB_PASSWORD: monapppwd
16       DB_NAME: monapp
17     networks:
18       - monapp-network
19    bdd:
20     image: mariadb:11
21     container_name: monapp-bdd
22     restart: always
23     environment:
24       MYSQL_ROOT_PASSWORD: root
25       MYSQL_DATABASE: monapp
26       MYSQL_USER: monapp
27       MYSQL_PASSWORD: monapppwd
28     volumes:
29       - monapp-db-data:/var/lib/mysql
30     networks:
31       - monapp-network
32  networks:
33    monapp-network:
34  volumes:
35    monapp-db-data:
```

Côté Dev : Docker compose complexe

Proxy ouvert

Service d'auth

Secrets

Healthcheck

Dépendances

```
1  services:
2    front:
3      image: registry.osupytheas.fr:443/monapp-front:latest
4      container_name: monapp-front
5      networks:
6        - monapp-network
7      depends_on:
8        - api
9    api:
10     image: registry.osupytheas.fr:443/monapp-api:latest
11     container_name: monapp-api
12     environment:
13       DB_HOST: bdd
14       DB_USER: monapp
15       DB_PASSWORD: monapppwd
16       DB_NAME: monapp
17     depends_on:
18       - bdd
19     networks:
20       - monapp-network
21    bdd:
22     image: mariadb:11
23     container_name: monapp-bdd
24     restart: always
25     environment:
26       MYSQL_ROOT_PASSWORD: root
27       MYSQL_DATABASE: monapp
28       MYSQL_USER: monapp
29       MYSQL_PASSWORD: monapppwd
30     volumes:
31       - /mnt/monnas/monapp:/var/lib/mysql
32     networks:
33       - monapp-network
34
35 proxy:
36   image: registry.osupytheas.fr:443/monapp-proxy:latest
37   container_name: monapp-proxy
38   ports:
39     - "8080:80"
40   depends_on:
41     - front
42     - api
43   networks:
44     - monapp-network
45
46 keycloak:
47   image: registry.osupytheas.fr:443/monapp-keycloak:latest
48   command: start --optimized
49   environment:
50     KC_DB_URL_DATABASE: keycloak
51     KC_DB_USERNAME: keycloak
52     KC_DB_PASSWORD: "${KC_DB_PASSWORD}"
53     KC_DB_SCHEMA: public
54     KC_BOOTSTRAP_ADMIN_USERNAME: "${KEYCLOAK_ADMIN_USERNAME}"
55     KC_BOOTSTRAP_ADMIN_PASSWORD: "${KEYCLOAK_ADMIN_PASSWORD}"
56     JAVA_OPTS_APPEND: "-Xms512m -Xmx2048m"
57   depends_on:
58     postgres:
59       condition: service_healthy
60   healthcheck:
61     test: ["CMD-SHELL", "exec 3<>/dev/tcp/127.0.0.1/9000; echo -e"]
62     interval: 30s
63     timeout: 10s
64     retries: 3
65     start_period: 60s
66   networks:
67     - monapp-network
68
69 networks:
70   monapp-network:
71
72 volumes:
73   monapp-db-data:
```

Côté Admin Sys : Le registry

Dernière brique indispensable

Permet de stocker les images custom

Les rend accessibles au `'docker pull'`

Tourne dans un container Docker

Interface web et api pour lister les images dispos

Quasi indispensable

Apporte beaucoup de souplesse

Côté Admin Sys : Choix d'une GUI

La ligne de commande c'est bien mais :

- Oblige à donner accès SSH aux dev à la VM docker
- La gestion de 150 containers est complexe

Nous avons choisis Portainer : open source et gratuit en version CE



Côté Admin Sys : Portainer

Permet aux devs de gérer leurs stack facilement

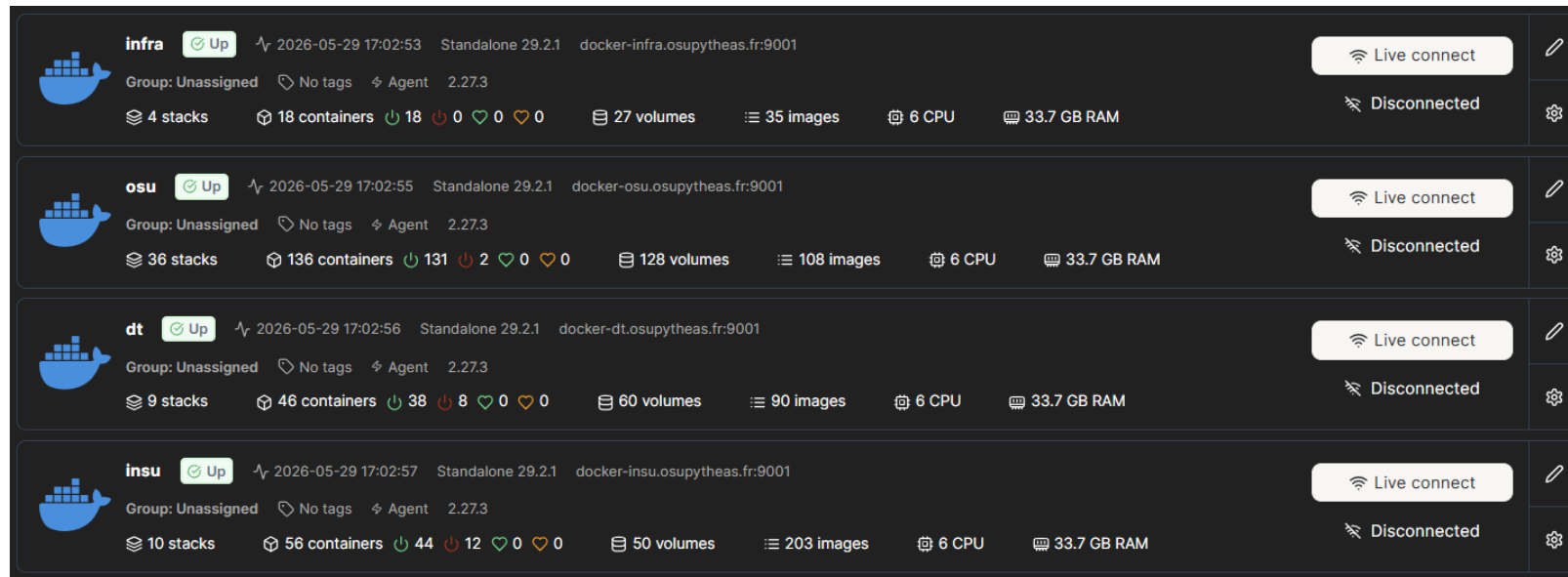
Authentification LDAP

Gestion des niveaux utilisateurs et des équipes

Côté Admin Sys : Portainer

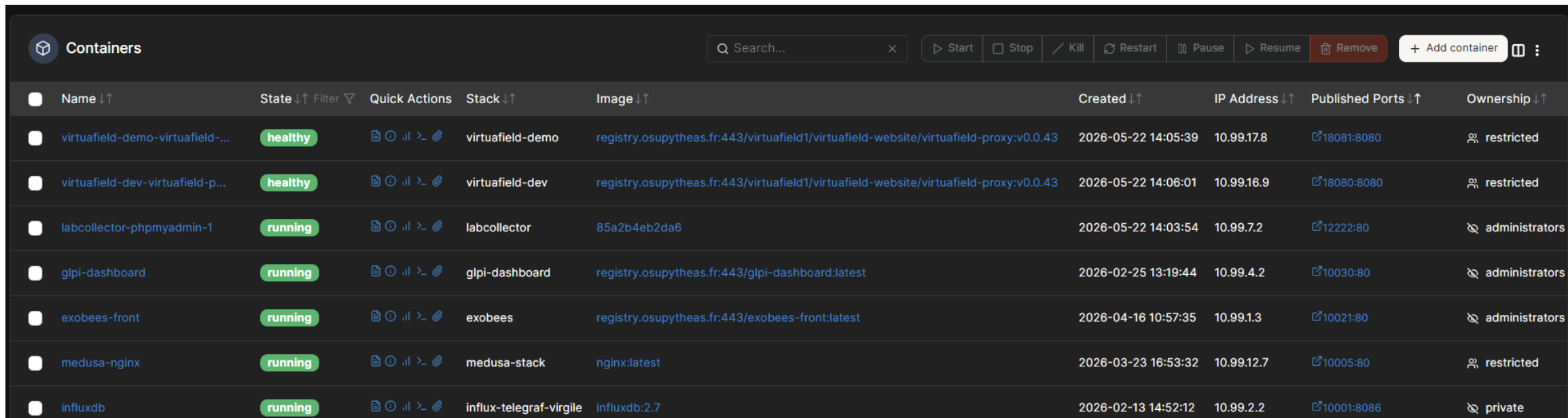
Permet de créer des « environnements »

- Agents installés sur VM docker pour centraliser la gestion
- Cloisonnement des droits suivant les environnements



Côté Admin Sys : Portainer

Permet d'avoir une vue d'ensemble des containers



The screenshot shows the Portainer 'Containers' page. At the top, there's a search bar and a row of action buttons: Start, Stop, Kill, Restart, Pause, Resume, Remove, and an 'Add container' button. Below this is a table with columns for Name, State, Quick Actions, Stack, Image, Created, IP Address, Published Ports, and Ownership. The table lists eight containers with their respective states (healthy or running) and ownership (restricted or private).

Name ↓↑	State ↓↑ Filter ▾	Quick Actions	Stack ↓↑	Image ↓↑	Created ↓↑	IP Address ↓↑	Published Ports ↓↑	Ownership ↓↑
virtuafield-demo-virtuafield-...	healthy		virtuafield-demo	registry.osupytheas.fr:443/virtuafield1/virtuafield-website/virtuafield-proxy:v0.0.43	2026-05-22 14:05:39	10.99.17.8	18081:8080	restricted
virtuafield-dev-virtuafield-p...	healthy		virtuafield-dev	registry.osupytheas.fr:443/virtuafield1/virtuafield-website/virtuafield-proxy:v0.0.43	2026-05-22 14:06:01	10.99.16.9	18080:8080	restricted
labcollector-phpmyadmin-1	running		labcollector	85a2b4eb2da6	2026-05-22 14:03:54	10.99.7.2	12222:80	administrators
glpi-dashboard	running		glpi-dashboard	registry.osupytheas.fr:443/glpi-dashboard:latest	2026-02-25 13:19:44	10.99.4.2	10030:80	administrators
exobeas-front	running		exobeas	registry.osupytheas.fr:443/exobeas-front:latest	2026-04-16 10:57:35	10.99.1.3	10021:80	administrators
medusa-nginx	running		medusa-stack	nginx:latest	2026-03-23 16:53:32	10.99.12.7	10005:80	restricted
influxdb	running		influx-telegraf-virgile	influxdb:2.7	2026-02-13 14:52:12	10.99.2.2	10001:8086	private



Côté Dev : Portainer

Permet de relancer, stopper et migrer sa stack facilement

Define or paste the content of your docker compose file here

```
1 services:
2
3 watchdog:
4   container_name: watchdog
5   image: registry.osupytheas.fr:443/watchdog
6   restart: always
7   environment:
8     - TZ=Europe/Paris
9   volumes:
10    - /etc/timezone:/etc/timezone:ro
11   labels:
12     - "com.centurylinklabs.watchtower.enable=true"
13     - "com.centurylinklabs.watchtower.scope=global"
14
15 prometheus:
16   container_name: prometheus
17   image: prom/prometheus
18   volumes:
19     - /mnt/srvstk2d/docker/data/watchdog/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
20   ports:
21     - "9521:9090"
22
23
24 grafana:
25   container_name: grafana
```

Environment variables

Webhooks

Create a Stack webhook  ☐  Business Feature

Actions

Containers

Search...  Start  Stop  Kill  Restart  Pause  Resume  Remove 

	Name ↓↑	State ↓↑ Filter ▾	Quick Actions	Stack ↓↑	Image ↓↑	Created ↓↑	IP Address ↓↑	Published Ports ↓↑	Ownership ↓↑
☐	grafana	running	   	watchdog	grafana/grafana	2026-03-30 17:02:32	10.99.27.4	 9520:3000	 administrators
☐	prometheus	running	   	watchdog	prom/prometheus	2026-03-30 17:02:32	10.99.27.3	 9521:9090	 administrators
☐	watchdog	running	   	watchdog	registry.osupytheas.fr:443/watchdog	2026-03-30 17:02:32	10.99.27.2	-	 administrators

Items per page 10 ▾

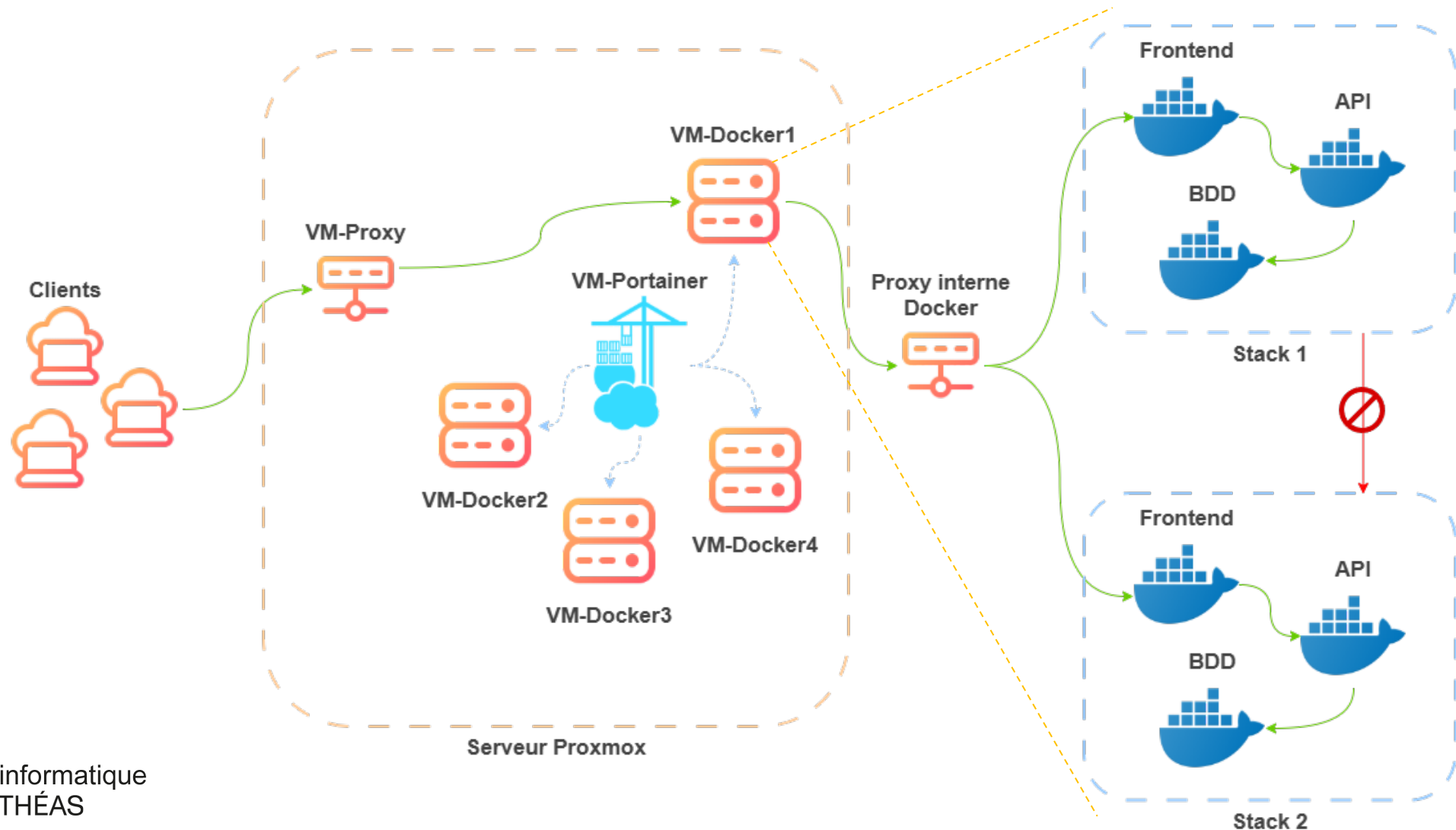


Et l'infra dans tout ça ?



Côté Admin Sys : L'infra

Plusieurs dizaines de VM > 5 VM dédiés web (hors proxy)



Côté Admin Sys : L'infra

Avantage des VM = possibilité de faire des snapshot

- Si un container est lancé en :latest
- Au reboot il y a un risque de switch de version
- Incompatibilité entre la version du service et du code
- Impossible de connaître l'ancienne version du service
- On peut faire un rollback des snapshot
- Puis le dev fixe la version de l'image

Pour aller plus loin

Intégration Continue et outils DevOps



Côté Dev : Pipeline CI/CD

Au cœur des méthodes DevOps

Globalement permet d'automatiser la mise en production

Permet entre autre :

- Faire des tests unitaires ou autre
- Tester la qualité du code (SonarQube,...)
- Builder les containers
- Push les containers sur un registry

Côté Dev : Pipeline CI/CD

Gitlab intègre la
fonctionnalité nativement

Fichier en YAML (déclaratif)

La base : on test, on build et
on push

```
1  stages:
2    - test
3    - build
4    - deploy
5
6  test:
7    stage: test
8    before_script:
9      - npm ci
10   script:
11     - npx ng test --watch=false
12   only:
13     - main
14
15  build:
16    stage: build
17    script:
18      - docker build --no-cache=true -t monapp-api .
19    only:
20      - main
21
22  deploy:
23    stage: deploy
24    script:
25      - docker tag monapp-api registry.osupytheas.fr:443/monapp-api:latest
26      - docker push --all-tags registry.osupytheas.fr:443/monapp-api
27    only:
28      - main
```

Côté Admin Sys : Reboot des containers

Pour mettre à jour les containers il faut les redémarrer

Pour automatiser on utilise Watchtower (open-source)

Container qui vérifie à intervalle régulier si les images du registry ont été modifiées

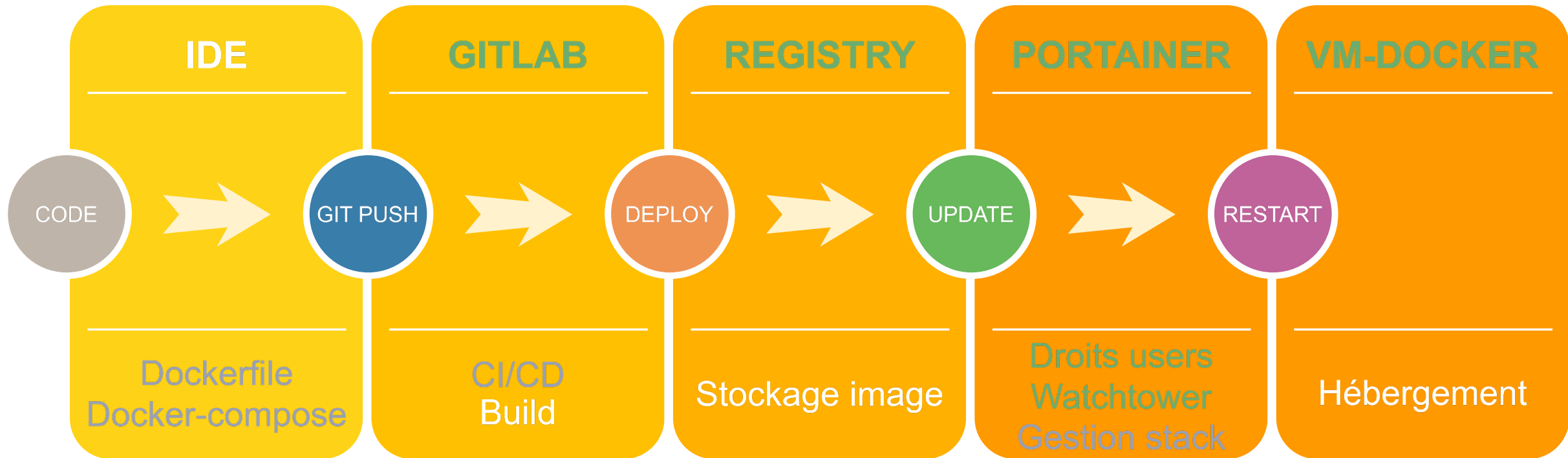
Deux lignes à rajouter dans le compose pour monitorer les containers

Retour d'expérience

Le bilan pour l'OSU



En résumé



Développeur Admin Sys

Globalement très positif ...

Tous les Dev de l'OSU ont fait la transition

Beaucoup plus d'autonomie et de liberté

Moins d'intervention côté ASR, principalement sur le DNS et le proxy

Infra rationalisée et plus sécurisée

Mais ...

Nouvelles compétences à acquérir pour les Dev

La courbe d'apprentissage est raide au début, surtout sans connaissance d'ASR

Pas à la porté de tout le monde (stagiaires sans connaissances docker par exemple)

Réflexion en cours : Comment on gère les BDD ?

BDD dans un volume NFS monté dans le docker
> corruptions de base régulières

BDD dans un volume docker sur la VM
> risque d'une VM trop grosse avec beaucoup de projets

BDD sur une/des VM dédiées hors docker
> on perd le cloisonnement de la stack