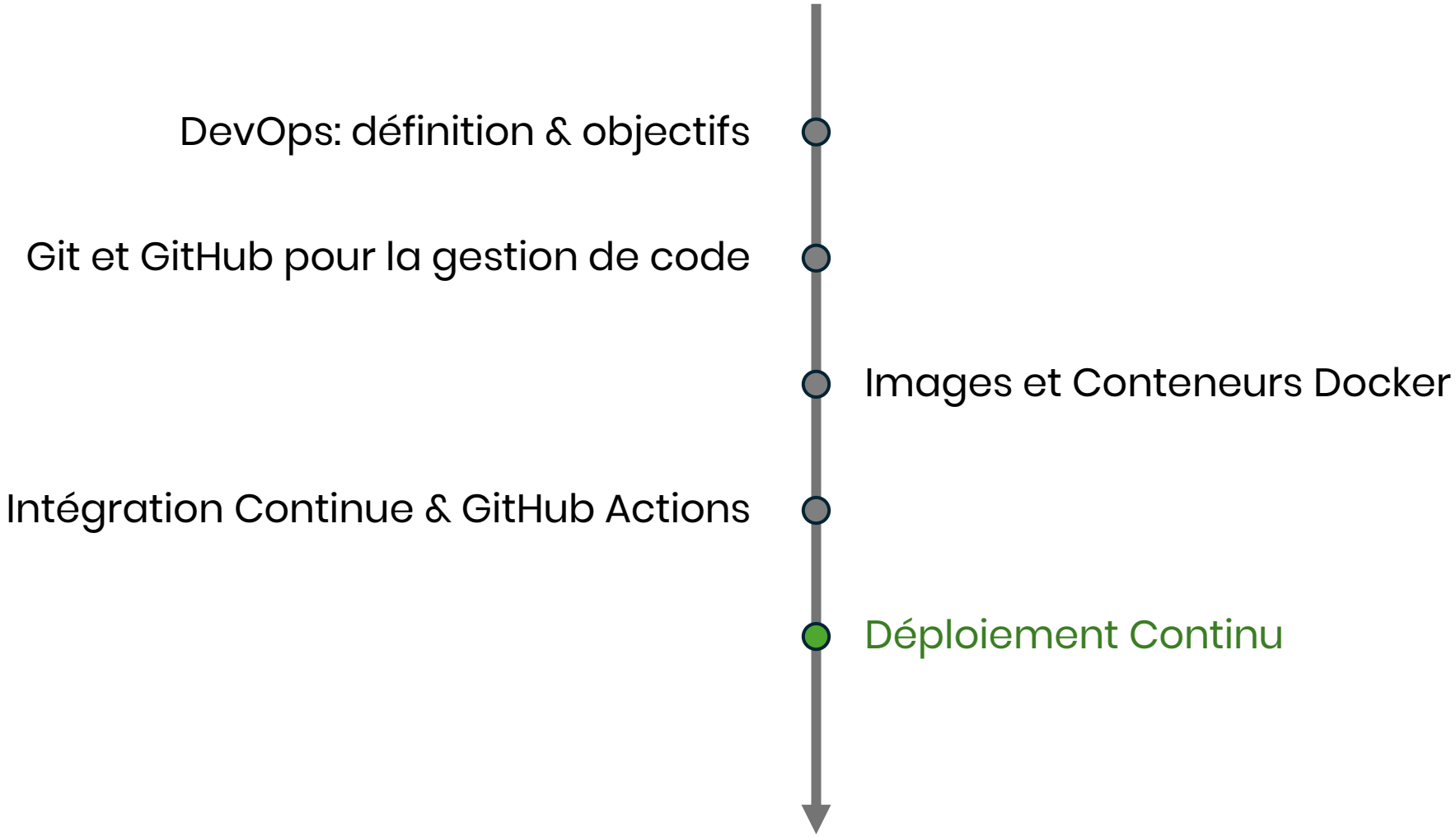
Projet Tutoré & DevOps

Hiba Najjar

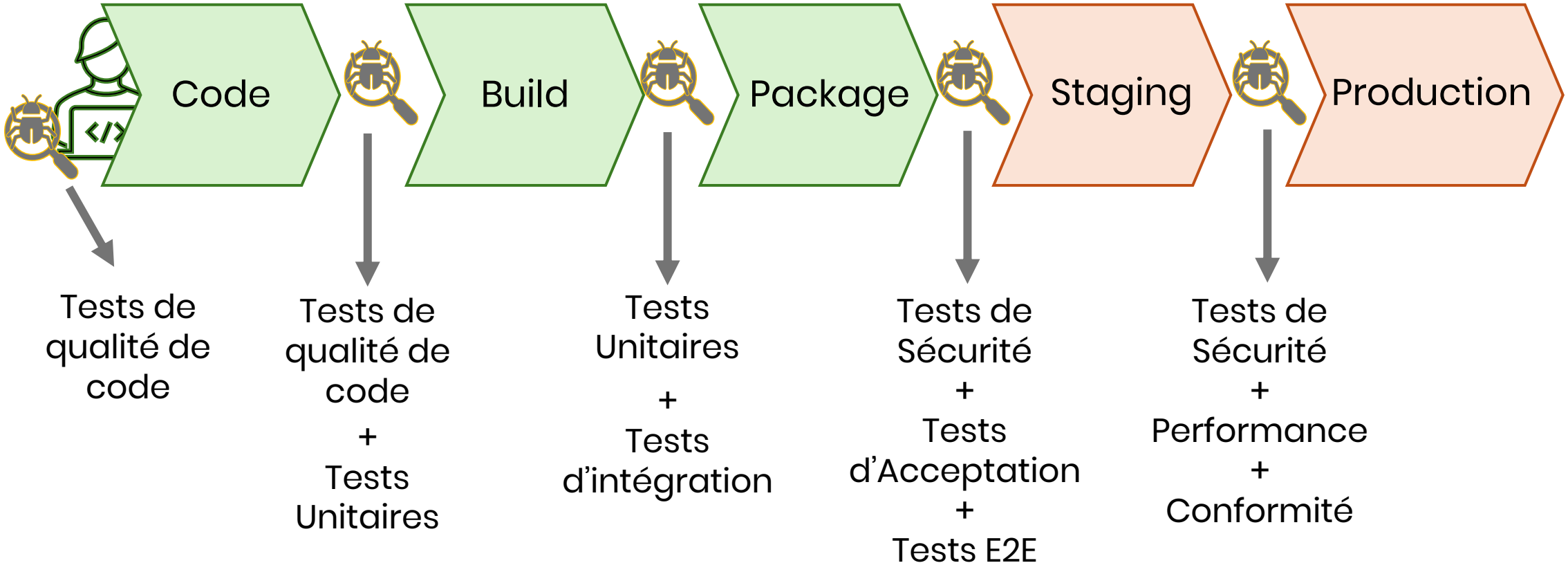
S7, GInf4

UPF Université Privée de Fès
الجامعة الخاصة لفاس
Private University of Fez

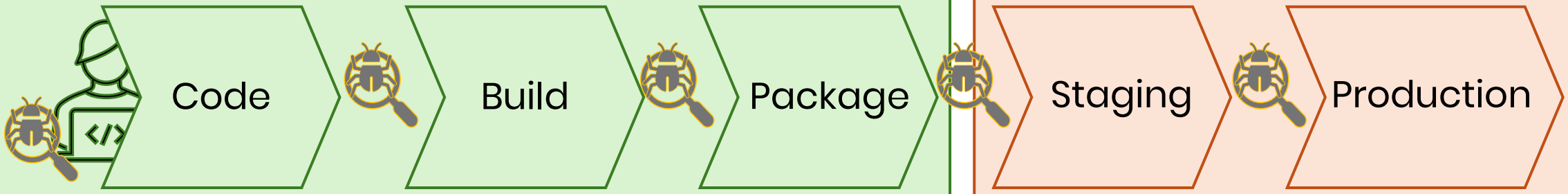
On a vu jusqu'à aujourd'hui



Intégration Continue > Déploiement Continu



Intégration Continue > Déploiement Continu



Intégration Continue

Déploiement Continu

Environnements Dev, Staging et Prod

- Écriture du Code
- Nouvelles fonctionnalités
- Expérimentations

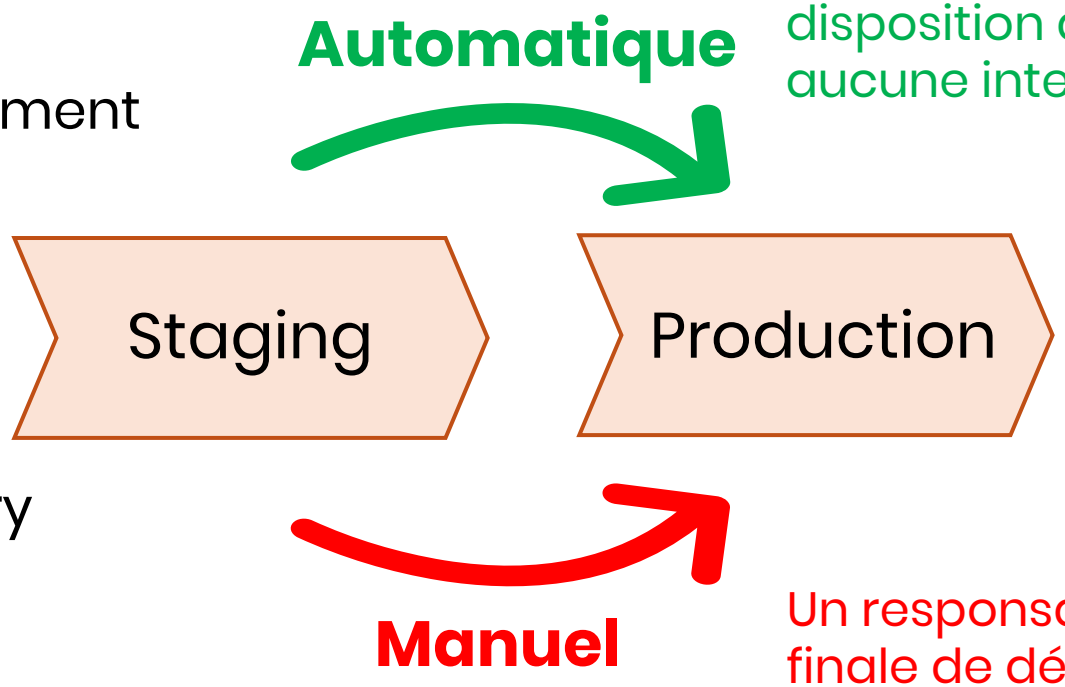
- ✚ Test rigoureux
- Assurance Qualité (QA)
- Tests de sécurité
- Imite l'environnement de production

- Étape finale
- Environnement disponible aux utilisateurs
- Surveillé de près
- Stable et sécurisé

Déploiement Continue vs. Livraison Continu

CD = Continuous **D**eployment
(Déploiement continu)

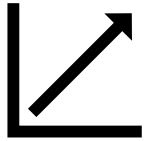
Chaque changement qui passe tous les tests automatisés est mis à la disposition des utilisateurs, sans aucune intervention humaine.



CD = Continuous **D**elivery
(livraison continu)

Un responsable prend la décision finale de déployer ou non les changements déjà validés par les tests automatisés

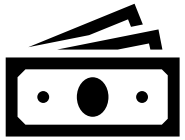
Avantages du Déploiement Continu



Meilleure qualité

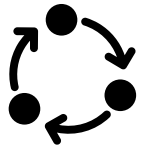
Accélération de la mise sur marché

Amélioration de l'expérience client



Réduction des coûts

Meilleure collaboration d'équipe



Boucle de rétroaction accélérée

Automatisation des déploiements
> packaging d'applications traditionnel

Fichier binaires / exécutables

Code source compilé en un fichier exécutable (.exe sur Windows, sans extension pour Linux)

Archives (JAR, WAR, ..)

Utilisé pour les applications basées sur des machines virtuelles ou des serveurs d'applications, contiennent le code, ressources et métadonnées

Paquets de scripts

Compresse le code, les assets, un env virtuel et les fichiers de dépendances

> nécessitent tous que l'environnement cible (le serveur de production) ait exactement les bonnes versions des dépendances (bibliothèques, *runtime*, OS) installées

Automatisation des déploiements
› Docker et registres d'images

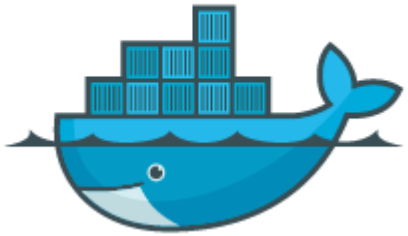
1

Construire l'image,

Dockerfile

2

la pousser vers un registre,



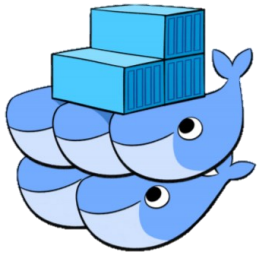
Docker HUB

GitHub Container Registry (GHCR)

GitLab Container Registry
Azure Container Registry (ACR)
Google Artifact Registry

3

la déployer par une plateforme d'orchestration de conteneurs



Docker Swarm

Kubernetes (K8s)

GitHub Container Registry (GHCR)

(documentation ici)

1. Pousser vers GHCR depuis votre CLI locale

A. Créer un PAT (Personal Access Token)

- Aller sur GitHub Settings › Developer settings › Personal access tokens › Tokens (classic)
- Générer un nouveau token avec permissions d'écriture des packages
- Copier le token

B. Se connecter à GHCR sur votre machine

- Depuis la CLI, définissait une variable d'env du token

```
export CR_PAT=<VOTRE_TOKEN>
```

- Authentification docker sur ghcr.io

```
echo $CR_PAT | docker login ghcr.io -u <VOTRE_NOM_UTILISATEUR> --password-stdin
```

C. Étiqueter et pousser l'image locale

- Modifier le nom de l'image:

```
docker tag NOM_IMAGE_LOCALE:TAG_LOCALE ghcr.io/NAMESPACE/NOM_IMAGE:VERSION
```

- Pousser vers ghcr

```
docker push ghcr.io/NAMESPACE/NOM_IMAGE:VERSION
```

GitHub Container Registry (GHCR)

(documentation ici)

1. Pousser vers GHCR depuis votre CLI locale

CMD

A. Créer un PAT (Personal Access Token)

- Aller sur GitHub Settings › Developer settings › Personal access tokens › Tokens (classic)
- Générer un nouveau token avec permissions d'écriture des packages
- Copier le token

B. Se connecter à GHCR sur votre machine

- Depuis la CLI, définissait une variable d'env du token

```
set CR_PAT=<VOTRE_TOKEN>
```

- Authentification docker sur ghcr.io

```
echo %CR_PAT% | docker login ghcr.io -u <VOTRE_NOM_UTILISATEUR> --password-stdin
```

C. Étiqueter et pousser l'image locale

- Modifier le nom de l'image:

```
docker tag NOM_IMAGE_LOCALE:TAG_LOCALE ghcr.io/ID-GITHUB/NOM_IMAGE:VERSION
```

- Pousser vers ghcr

```
docker push ghcr.io/ID-GITHUB/NOM_IMAGE:VERSION
```

GitHub Container Registry (GHCR)

(documentation ici)

2. Pousser vers GHCR depuis un workflow GitHub Actions

```yaml
- name: Log in to GHCR
  uses: docker/login-action@v3
  with:
    registry: ghcr.io
    username: ${{ github.actor }}
    password: ${{ secrets.GITHUB_TOKEN }}

- name: Build and Push Docker image
  uses: docker/build-push-action@v5
  with:
    push: true
    tags: ghcr.io/${{ github.actor }}/NOM_IMAGE:VERSION
```

GitHub Container Registry ou Docker HUB ?

Registre	Avantages	Inconvénients
GitHub Container Registry (GHCR)	**Intégration Optimale :** Utilise la même authentification que GitHub (pas de secrets supplémentaires pour les développeurs). **Workflows simplifiés** si vous utilisez GitHub Actions pour la CI/CD.	Moins "universel" que Docker Hub si vous changez d'écosystème plus tard.
Docker Hub	**Universalité :** Le registre le plus connu et le plus utilisé. Fonctionne avec absolument tous les outils. Authentification plus simple sur les machines de développement (le docker login est standard).	Nécessite la gestion de secrets (identifiants Docker Hub) séparés des identifiants GitHub.

Automatisation des déploiements
› Docker et registres d'images

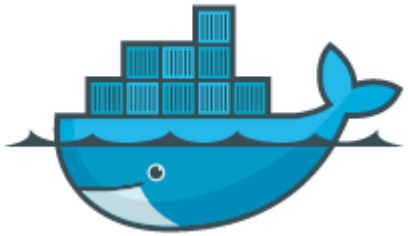
1

Construire l'image

Dockerfile

2

La pousser vers un registre



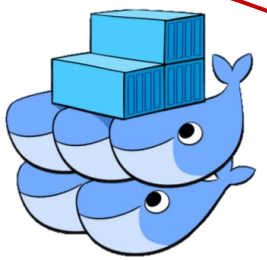
Docker HUB

GitHub Container Registry (GHCR)

GitLab Container Registry
Azure Container Registry (ACR)
Google Artifact Registry

3

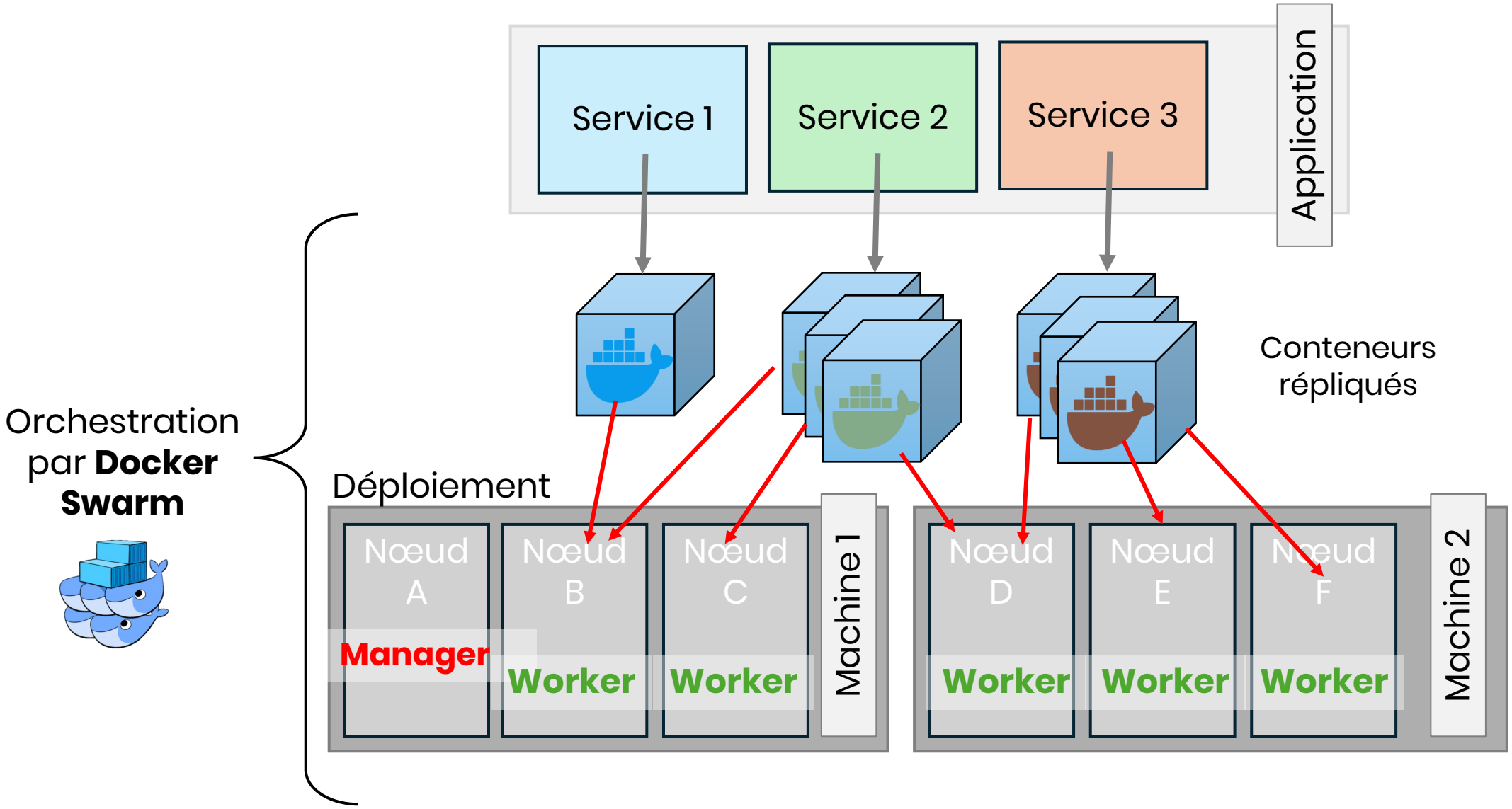
La déployer par une plateforme d'orchestration de conteneurs



Docker Swarm

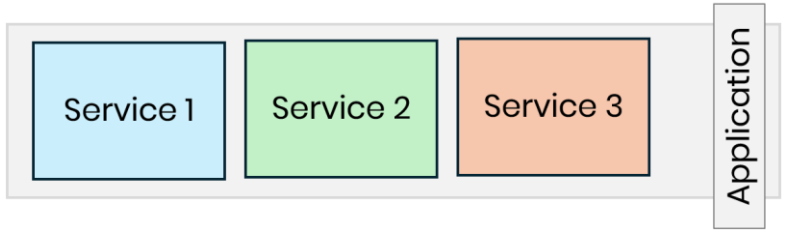
Kubernetes

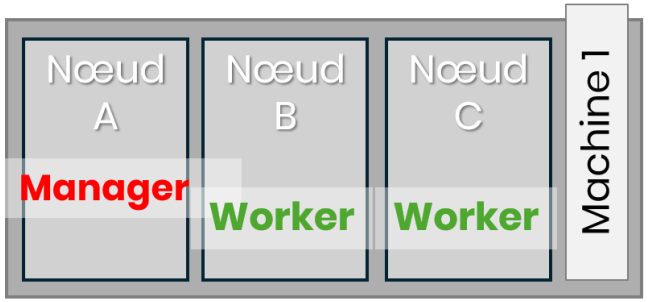
Déploiement avec Docker Swarm



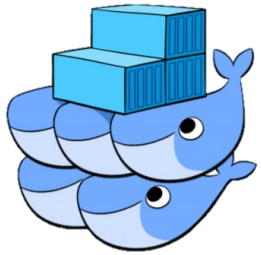
Déploiement avec Docker Swarm

1. L'**Application** est composée d'un ensemble de services interdépendants (ex: Front-End, API, Paiement)
2. Les **services** définissent l'image Docker à utiliser, le nombre de réplicas souhaités, et les configurations nécessaires
3. Les **Conteneurs** sont les instances légères et isolées qui exécutent concrètement le code d'un service
4. Les **Réplicas** sont des copies multiples d'un même conteneur, lancées par l'orchestrateur pour garantir la haute disponibilité et la répartition de la charge de travail sur le cluster.
5. Un **nœud Manager** est la machine virtuelle ou physique qui sert de cerveau au cluster >> Maintien et planification.
6. Un **nœud Worker** est la machine virtuelle ou physique qui exécute la charge de travail >> il reçoit les ordres du Manager pour lancer et maintenir les conteneurs (réplicas).
7. Les **Machines Physiques / Cloud** : Fournissent le matériel de base (CPU, RAM, Disque) sur lequel les VMs sont exécutées





Déploiement avec Docker Swarm



Fonctions de Docker Swarm:

- Regrouper et gérer des nœuds Docker (cluster logique)
- Déployer des services à l'échelle (répliqués et distribués sur plusieurs nœuds du cluster > mise à l'échelle facile = scaling)
- Assurer la haute disponibilité et la tolérance aux pannes (redémarrarage automatique du conteneur défaillant sur un autre nœud disponible)
- Équilibrer la charge (Le trafic entrant vers un service est automatiquement distribué entre les conteneurs d'un service chargé)
- Simplifier le réseau (communication sécurisée et transparente entre les conteneurs)

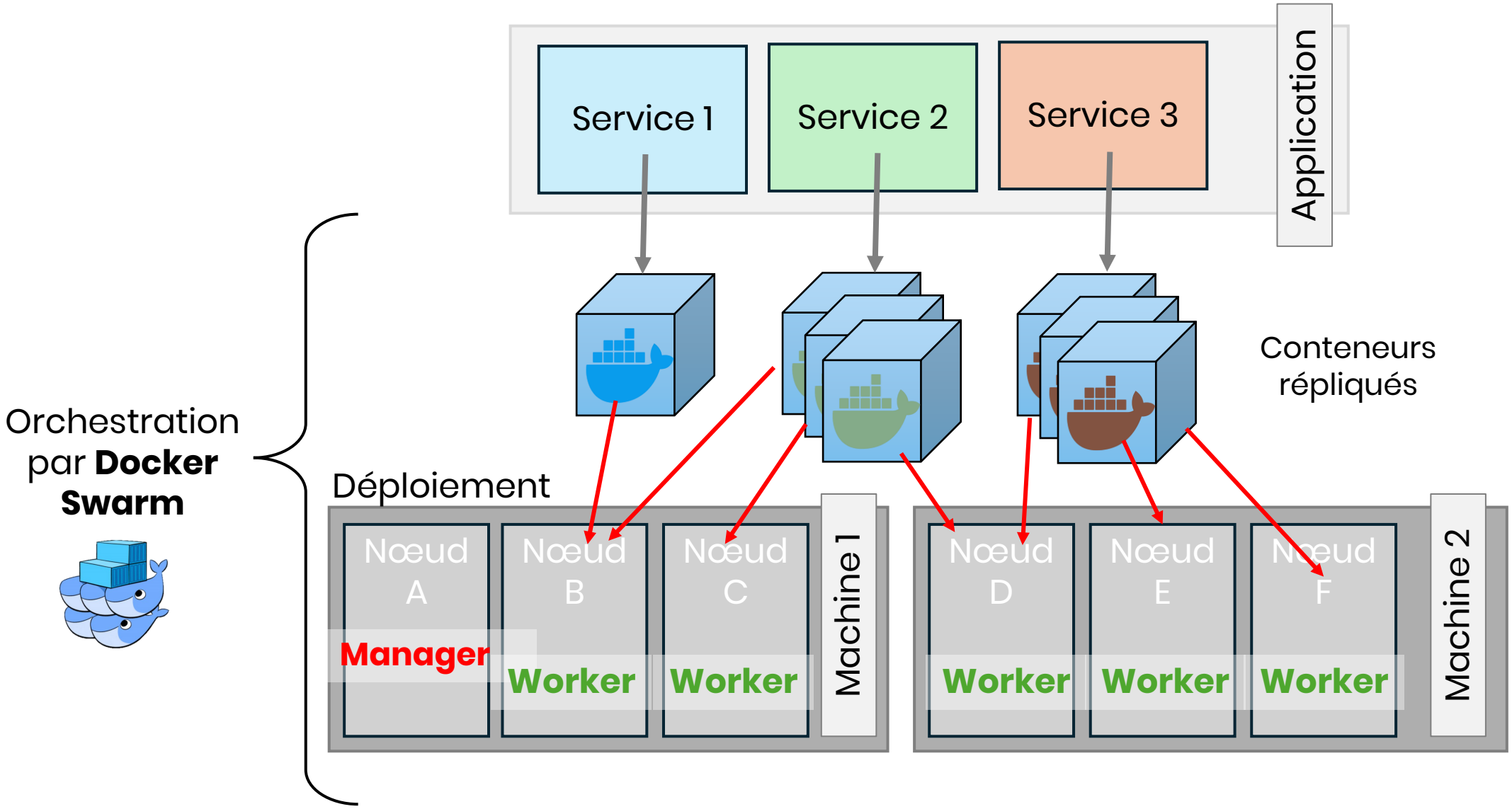
Déploiement avec Docker Swarm

Pourquoi des Nœuds / VM et pas des machines physiques ?

- Une seule machine physique puissante peut héberger des dizaines de machines virtuelles de différentes tailles.
- La mise à l'échelle est quasi instantanée. Pour ajouter un nouveau nœud à votre cluster Swarm, il suffit de créer une nouvelle VM en quelques minutes / secondes.
- Les VMs permettent d'allouer précisément les ressources nécessaires et de les ajuster dynamiquement.
- Les VMs offrent une isolation plus stricte que les conteneurs.
- Une machine virtuelle peut être déplacée d'un serveur physique à un autre, ou même d'un fournisseur de cloud à un autre, sans avoir à reconfigurer le système d'exploitation ou le matériel

>> utiliser la virtualisation (VM) pour gérer et isoler le matériel, et la conteneurisation (Docker Swarm) pour gérer et isoler les applications.

Déploiement avec Docker Swarm



Application

Lancer Docker Swarm sur sa machine locale

1. Adapter docker-compose.yml

docker-compose.yml

```
……
services:
  db:
    ……
    deploy:
      mode: replicated
      replicas: 2
      restart_policy:
        condition: any
```

Application

Lancer Docker Swarm sur sa machine locale

1. Adapter docker-compose.yml
2. Initialiser Docker Swarm, deployer le stack, verifier l'état des services

cmd

```
# initialisation, définir les nœuds managers / workers
docker swarm init
docker node ls
docker node update --availability drain docker-desktop
docker node update --availability active docker-desktop

# déploiement
docker stack deploy -c docker-compose.yml articlesapp_swarm

# vérification des services
docker service ls
docker stack ps articlesapp_swarm
```

Application

Lancer Docker Swarm sur sa machine locale

1. Adapter docker-compose.yml
2. Initialiser Docker Swarm, deployer le stack, verifier l'état des services
3. Accéder à l'application (localhost) et verifier les replicas qui servent la page
4. Tester la disponibilité en supprimant un conteneur

cmd

```
# obtenir l'ID d'un conteneur puis l'arrêter
docker ps
docker container stop <ID_d_un_conteneur>

# vérification des réplicas disponibles
docker stack ps articlesapp_swarm
```

Application

Lancer Docker Swarm sur sa machine locale

1. Adapter docker-compose.yml
2. Initialiser Docker Swarm, deployer le stack, verifier l'état des services
3. Accéder à l'application (localhost) et verifier les replicas qui servent la page
4. Tester la disponibilité en supprimant un conteneur
5. Mise à échelle en augmentant le nombre de réplicas

cmd

```
# augmenter le nombre de réplicas
docker service scale articlesapp_swarm_app=5

# vérification des réplicas disponibles
docker stack ps articlesapp_swarm
```

Stratégies de déploiement

Rolling

Blue/Green

Canary

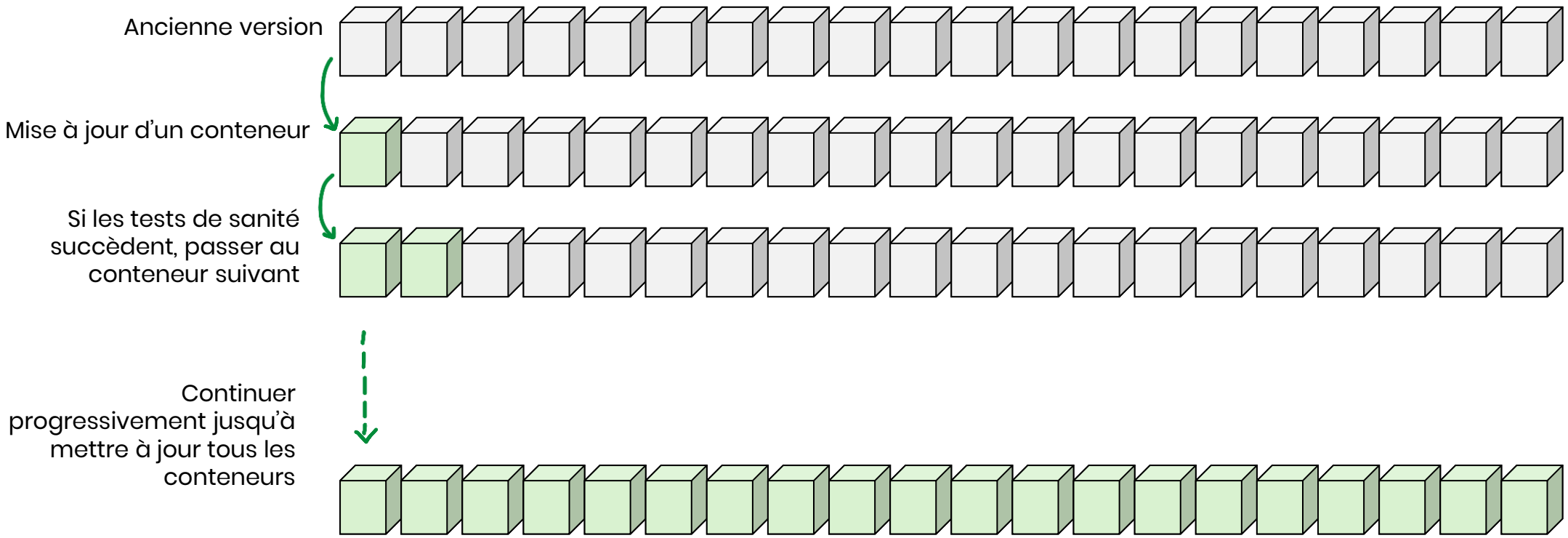
Stratégies de déploiement

Rolling | Blue/Green | Canary

- Utilisée par défaut par Docker Swarm.
- Remplace progressivement l'ancienne version par la nouvelle, conteneur par conteneur.
- Vérifie à chaque fois que la nouvelle version est saine (health checks), puis passe au conteneur suivant.
- Pendant le déploiement, certains utilisateurs peuvent être servis par l'ancienne version et d'autres par la nouvelle.
- ✗ Si la nouvelle version présente un bogue, l'annulation (rollback) doit également se faire progressivement, ce qui peut prendre du temps.
- ✗ Coexistence : L'ancienne et la nouvelle version coexistent pendant un certain temps, ce qui peut causer des problèmes de compatibilité (par exemple si les versions parlent à la même base de données)

Stratégies de déploiement

Rolling | **B**lue/Green | **C**anary



Stratégies de déploiement

Rolling | **B**lue/Green | **C**anary

docker-compose.yml

```
.....
  app:
    …
    deploy:
      mode: replicated
      replicas: 3
      restart_policy:
        condition: any
      update_config:
        parallelism: 1
        order: stop-first # ou 'start-first'
        monitor: 15s  # Temps pour considérer qu'un conteneur est "sain«
        delay: 10s    # Temps avant de passer au groupe suivant, après monitor
```

Stratégies de déploiement

Rolling

- Utilisée par défaut par Docker Swarm.
- Remplace progressivement l'ancienne version par la nouvelle, conteneur par conteneur.
- Vérifie à chaque fois que la nouvelle version est saine (health checks), puis passe au conteneur suivant.
- Pendant le déploiement, certains utilisateurs peuvent être servis par l'ancienne version et d'autres par la nouvelle.
- ✗ Si la nouvelle version présente un bogue, l'annulation (rollback) doit également se faire progressivement, ce qui peut prendre du temps.
- ✗ Coexistence : L'ancienne et la nouvelle version coexistent pendant un certain temps, ce qui peut causer des problèmes de compatibilité (par exemple si les versions parlent à la même base de données)

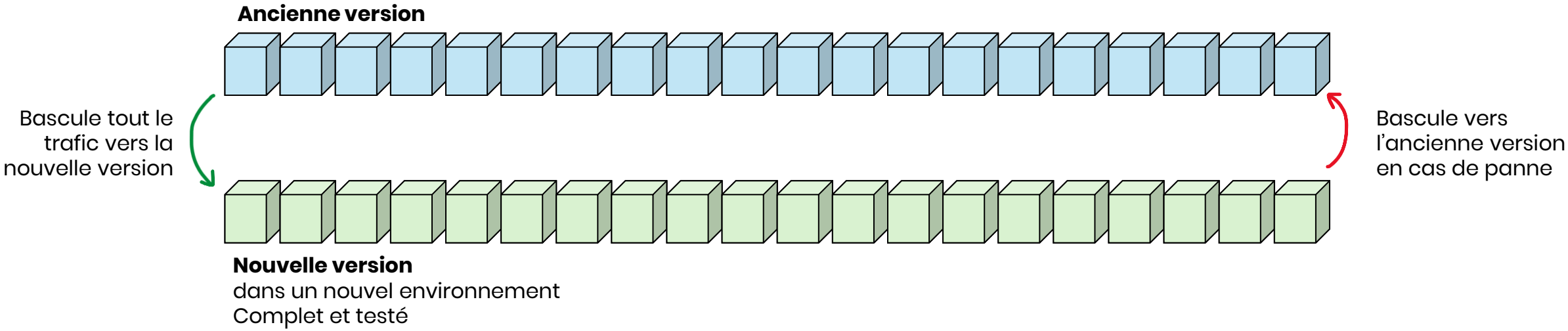
Blue/Green

- Maintient deux environnements de production complets et identiques: le "bleu" pour l'ancienne version, et le "vert" pour la nouvelle.
- Procédure : La nouvelle version est déployée sur l'environnement Vert et bien testée. Lorsque tout est validé, le Load Balancer est instantanément basculé pour diriger tout le trafic vers l'environnement Vert
- L'environnement Bleu est conservé en veille comme solution de rollback immédiate.
- L'utilisateur voit soit V1, soit V2, mais jamais les deux en même temps.
- ✗ Nécessite le double des ressources (deux environnements complets) pendant la phase de déploiement >> coûteux.

Canary

Stratégies de déploiement

Rolling | **B**lue/Green | **C**anary



Stratégies de déploiement

Rolling

- Utilisée par défaut par Docker Swarm.
- Remplace progressivement l'ancienne version par la nouvelle, conteneur par conteneur.
- Vérifie à chaque fois que la nouvelle version est saine (health checks), puis passe au conteneur suivant.
- Pendant le déploiement, certains utilisateurs peuvent être servis par l'ancienne version et d'autres par la nouvelle.
- ✗ Si la nouvelle version présente un bogue, l'annulation (rollback) doit également se faire progressivement, ce qui peut prendre du temps.
- ✗ Coexistence : L'ancienne et la nouvelle version coexistent pendant un certain temps, ce qui peut causer des problèmes de compatibilité (par exemple si les versions parlent à la même base de données)

Blue/Green

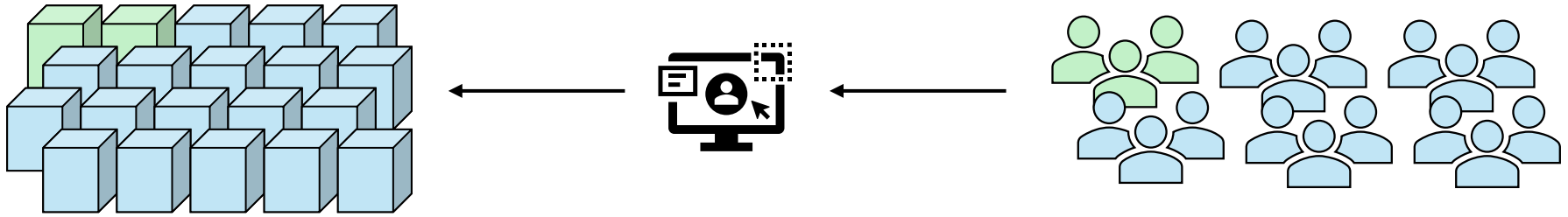
- Maintient deux environnements de production complets et identiques: le "bleu" pour l'ancienne version, et le "vert" pour la nouvelle.
- Procédure : La nouvelle version est déployée sur l'environnement Vert et bien testée. Lorsque tout est validé, le Load Balancer est instantanément basculé pour diriger tout le trafic vers l'environnement Vert
- L'environnement Bleu est conservé en veille comme solution de rollback immédiate.
- L'utilisateur voit soit V1, soit V2, mais jamais les deux en même temps.
- ✗ Nécessite le double des ressources (deux environnements complets) pendant la phase de déploiement >> coûteux.

Canary

- Approche progressive et à faible risque.
- Procédure : La nouvelle version est déployée sur un petit sous-ensemble de réplicas ou n'est exposée qu'à un petit pourcentage d'utilisateurs. L'équipe surveille attentivement les indicateurs clés de performance (KPI) sous la nouvelle version. Si elle est stable, le pourcentage de trafic bascule progressivement jusqu'à 100%.
- En cas de problème, seul un petit nombre d'utilisateurs est affecté, et l'on procède immédiatement à l'annulation (rollback).
- Permet de tester les performances et la stabilité de la nouvelle version sous une charge réelle et progressive.
- ✗ Nécessite un système d'équilibrage de charge sophistiqué.

Stratégies de déploiement

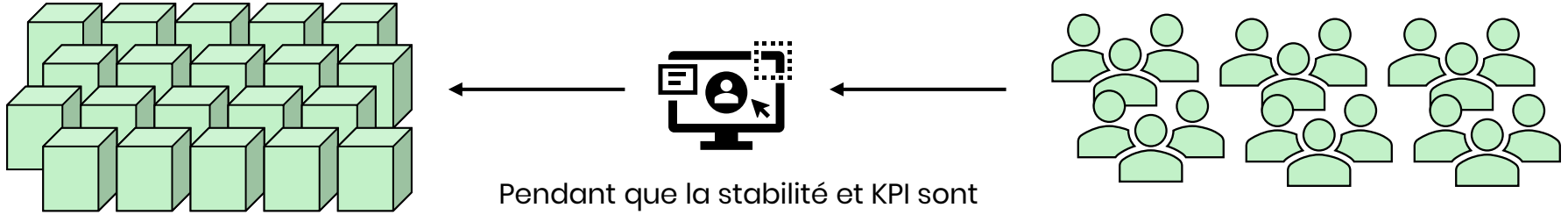
Rolling | **B**lue/Green | **C**anary



nouvelle version déployée sur un petit sous-ensemble de réplicas

ou

nouvelle version exposée à un petit pourcentage d'utilisateurs



Pendant que la stabilité et KPI sont surveillé, le trafic bascule **progressivement** jusqu'à 100%.

Contrôle 1

Devoirs manquants → 0

Période de rattrapage contrôle 1: jusqu'à la séance prochaine, avec **pénalités**:

- Retard sur le devoir
- Retard sur le sujet et groupes du projet

Sujet projet manquant:

najjar@upf.ac.ma

Groupe projet manquant:

https://tinyurl.com/GI4APROJECTLL

https://tinyurl.com/GI4BPROJECTLL

Projets

Cahier de charges

1. Ouvrir ce doc (lecture seulement):
   https://tinyurl.com/GIPROJECTTEMPLATE
2. Un membre le copie sur son google drive
   1. Le renommer <ProjetTutoré_CDC_grpX>
   2. Ajouter le reste de l'équipe (share > add)
   3. M'ajouter aussi (hibanajjar998@gmail.com) en mode « commentateur »
3. Remplir les champs déjà définit la séance dernière (problématique et solution)
4. Remplir les nouvelles sections

Semaine prochaine

- Notes et Correction contrôle 1
- Il se peut que vous ayez le contrôle 2