303 - Automation

DEVOPS

WIK-DEVOPS-302

Intervenant : Jeremy Trufier <jeremy@wikodit.fr>

ynov

wikodit

WIK-DEVOPS

Programme DEVOPS

301 – Introduction, Kaizen et CAMS	1 séance
302 – La culture DevOps	1 séance
303 – Automatisation / Déploiement continu	4 séance
304 – Mesure & Monitoring	2 séance
305 – Partage	2 séance

I. Introduction

La gestion des configurations

I. Introduction

La gestion des configurations

Chef / Puppet / Ansible / SaltStack

- Couplé avec l'Infrastructure as Code
- Gestions de parcs
- Tous ont une offre OpenSource
- Permet de provisionner des machines
- Basé sur un langage de programmation

La gestion des configurations

ServerSpec

- Tests unitaires pour l'infrastructure
- Basé sur Rspec
- Valider le bon fonctionnement des serveurs
- Peut-être couplé avec les outils de gestion de configurations

La gestion des configurations

Environnement Virtuels

- Environnement isolé et reproductible
- Virtual Environnement :
- Docker = Container
- Vagrant= Virtual Machine
- Gestions d'instances multiples
- Provisionnement sous forme de fichier de configuration

CI / CD

I. Introduction

La gestion des configurations

Continuous Integration (CI)

- Intégration du code de chacun sur une base commune
- Tests Automatisée avant intégration finale
- Créations de build
- Git, GitFlow & Pull Request
- Cible : Le développeur

Continuous Deployment (CD)

- Déploiement automatisé après CI
- Généralement environnement de pré-prod/staging
- Fournir au client un aperçu
- Cible : Le client/béta-testeur

La gestion des configurations

Continuous Delivery

- Validation QA (Quality Assurance)
- Release régulières
- Transparence pour l'utilisateur (ex: Chrome, store apps)
- Cible : L'utilisateur final

II. Docker

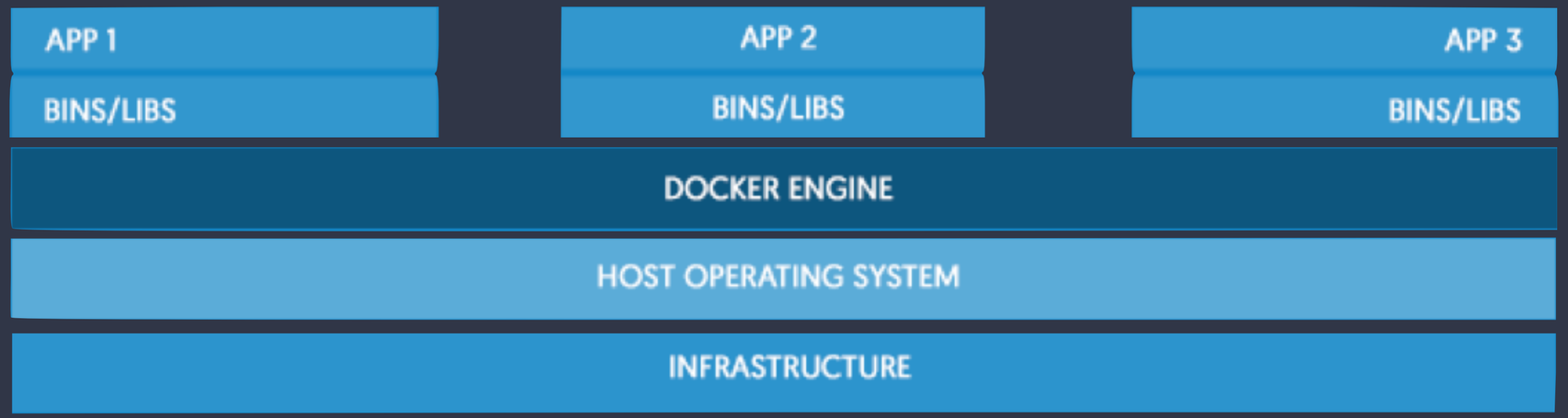
Docker Engine

II. Docker

Docker Engine

Vue d'ensemble

- Le coeur de la plateforme
- Containerisation



Docker Engine

Utilisation

```
> docker run --rm -ti -p 8080:80 dockercloud/hello-world
```

- Puis aller sur http://localhost:8080
- Télécharge l'image "hello-world" depuis le repos docker par défaut
- Créé un container
- Lance le container

Docker Engine

Image

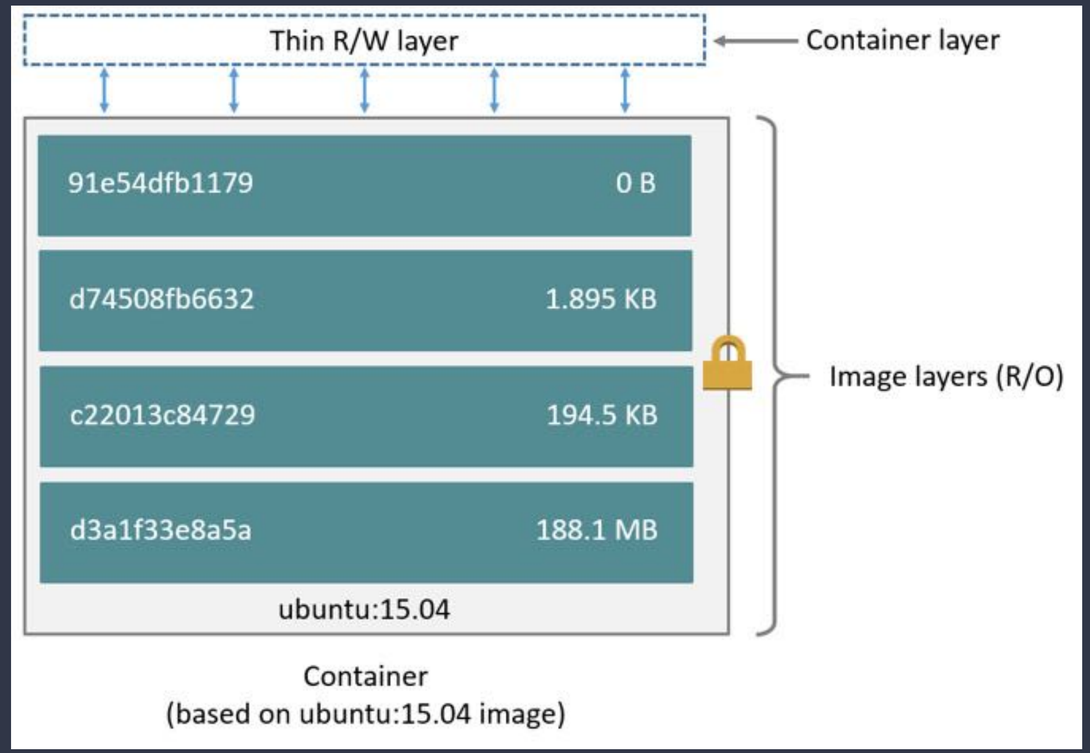
- Constitué de Layers (Read-Only snapshots)
- Les spécifications sont contenues dans un Dockerfile

Source: docs.docker.com

Docker Engine

Container

- Basé sur une image
- Lance et surveille une et une seule commande
- Possède un layer supplémentaire (=changements depuis l'image)
- Peut-être démarré, arrêté et suspendu
- Ne doit pas stocker de données
- Éphémère !!



Source: docs.docker.com

Utilisation des containers

Lancer l'interpreter SH sur un nouveau container Linux Alpine

```
> docker container run -it --rm alpine sh
# ls -l
```

Création, lancement et utilisation d'un container Redis

```
> docker container create --name devops-redis -p 36379:6379 redis:alpine
> docker container start devops-redis
> docker container ls
> docker container exec -ti devops-redis redis-cli
> docker container stop devops-redis
```

Obtenir de l'aide

```
> docker container --help
> docker container COMMAND --help
```

Docker Engine

Volumes

- Stockage de données
- Accessible depuis un ou plusieurs containers
- Jamais supprimés automatiquement
- Volume Plugins : iSCSI, NFS, ...
- Snapshot, backup, etc... sont à faire sur les volumes

Docker Engine

Volumes

Monter un dossier de l'host en tant que volume

```
> docker container run --rm -it -v /Users/jeremyt:/test alpine sh
# ls /test
```

Créer/utiliser un volume nommé (my-test-volume)

```
> docker container run --rm -it -v my-test-volume:/test alpine sh
# ls /test
# exit
# docker volume ls
```

Création d'une image

- Créer un dossier qui sera la base de notre image
- Créer un .dockerignore
- Créer un Dockerfile

Docker Engine

Le Dockerfile

Dans un nouveau dossier, ajoutez un fichier nommé `monfichier.txt` avec ce que vous voulez dedans.
Puis créez le Dockerfile suivant

Image de base à utiliser

Lancer une commande : installer nano, vim et créer /app

Remplir /app avec les données du répertoire actuel : .

```
FROM alpine:latest

RUN apk update && \
  apk add \
    nano \
    vim && \
  mkdir /app

COPY . /app
CMD cat /app/monfichier.txt
```

Génère une image nommée mytest et taguée latest

Lance un nouveau container en utilisant cette image et execute SH

```
> docker build -t mytest:latest .
> docker run --rm -it mytest sh
# cd /app
# ls
# vim monfichier.txt
```

Docker Engine

TD: Dockerize un projet NodeJS

https://github.com/Tronix117/devops-303

```
git clone https://github.com/Tronix117/devops-303.git
```

- ami-881a3fee

- Créez le Dockerfile qui va bien pour ce projet

FROM	Image de base (premier layer) - Astuce : `node:alpine`
RUN	Lancer une commande
WORKDIR /toto	Définir le répertoire de travail
USER xxxx	Change l'utilisateur d'execution
EXPOSE 5000	Ouvre un port sur l'extèrieur
CMD ["echo", "quelque-chose"]	La commande que les containers devront lancer et surveiller
COPY ou ADD .	Copier des fichiers dans l'image

Lister les container en cours `docker container ps`

Supprimer un container `docker container remove a356f21`

Créer un container avec un bind sur un port `docker container create --name toto -p 8080:8888 image-name`

Lancer un container pour une seule utilisation `docker container run --rm -d -p 8080:8888 image-name`

Arrêter/lancer un container `docker container start/stop a356f21`

Actions sur les images `docker image list/rm a356f21`

- ami-881a3fee

- Créez le Dockerfile pour `git clone https://github.com/Tronix117/devops-303.git`

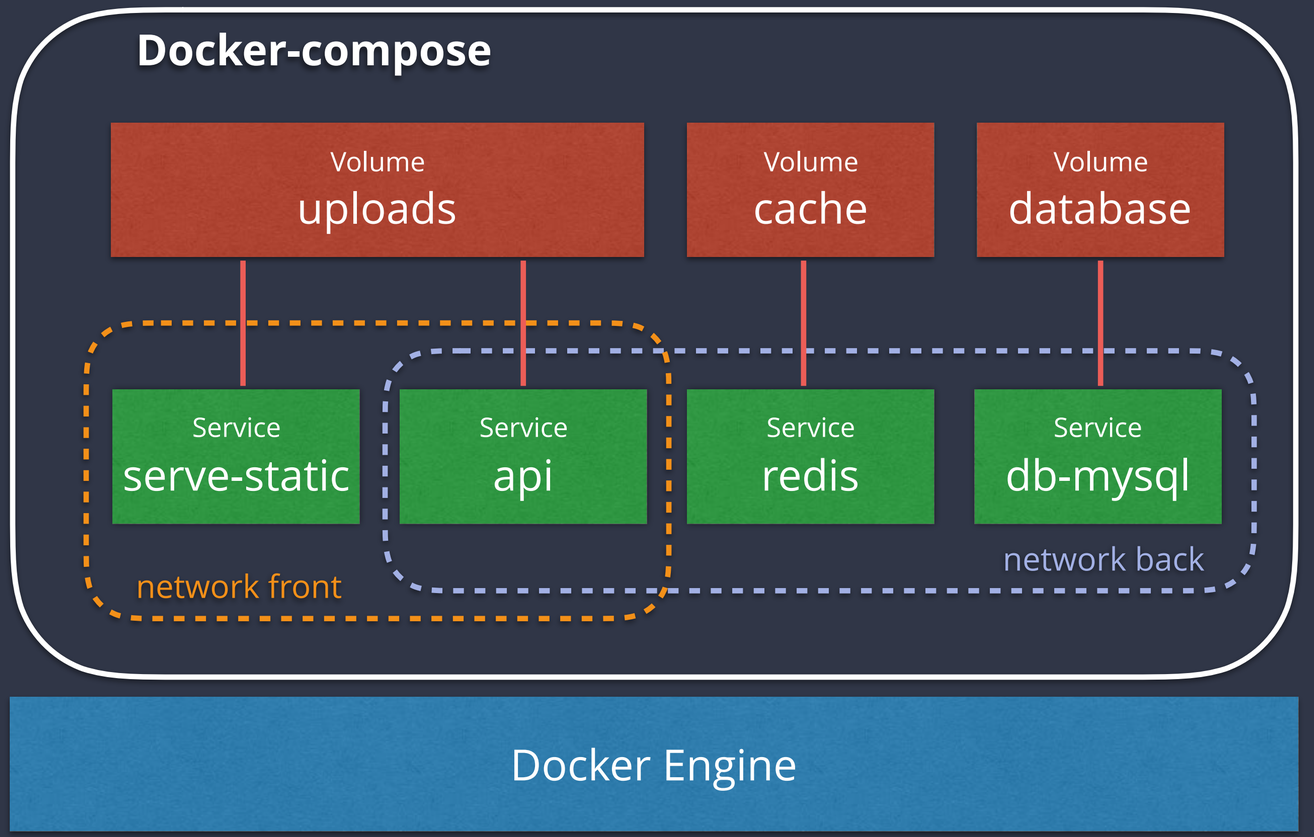
FROM image	Image de base (premier layer) - Astuce : `node:alpine`
RUN mkdir /toto	Lancer une commande
WORKDIR /toto	Définir le répertoire de travail
USER xxxx	Change l'utilisateur d'execution
EXPOSE 5000	Ouvre un port sur l'extèrieur
CMD echo quelque-chose	La commande que les containers devront lancer et surveiller
COPY ./ /toto	Copier des fichiers dans l'image

Docker Compose

II. Docker

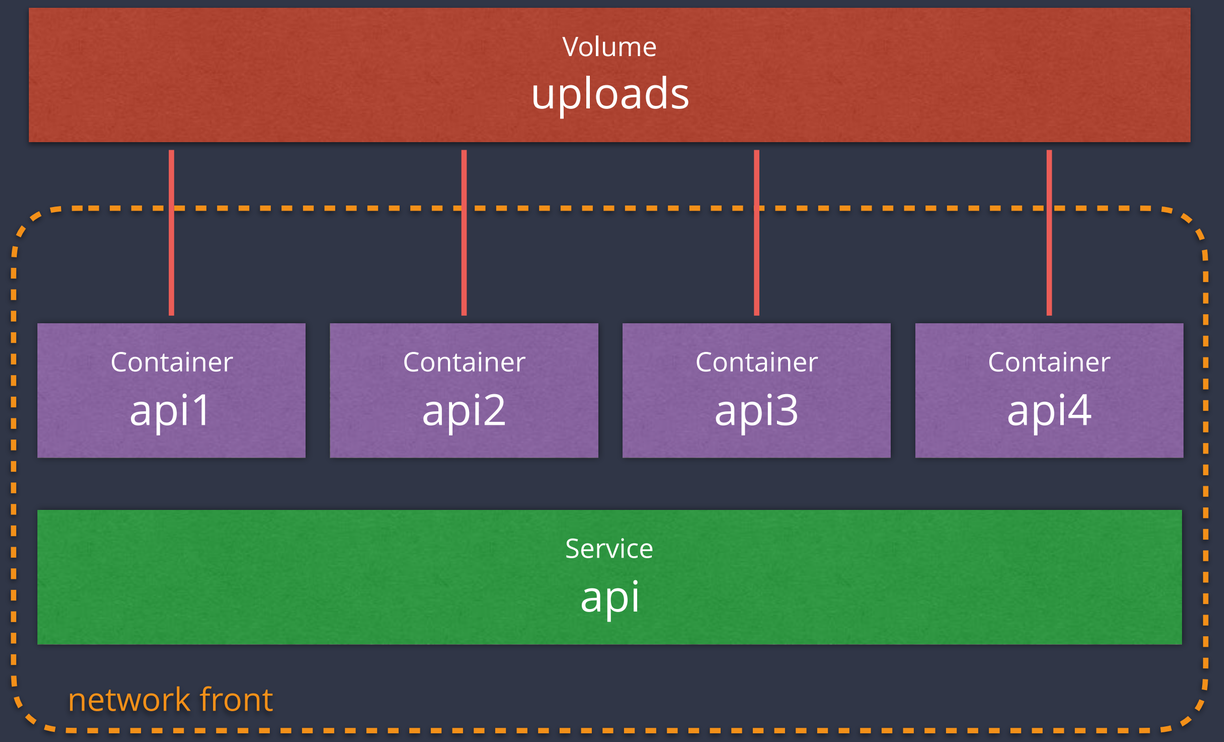
Vue d'ensemble

- Définit un ensemble de services Docker et leur interconnections
- Interface de scripting haut-niveau
- Fichier de configuration en YML
- Indispensable pour le dev, staging, testing



Services Docker

- Un container est un environnement virtuel
- Un service est une fonction a assurer
- Un service gère des containers répliqués
- Ici, le Service est API, on peut le scaler X fois, cela créera X containers exposant le même service



Version du fichier docker-compose (3 est la plus récente)

Définition des containers

Nom du container (aussi son hostname)

Image utilisée

Le nom du réseau dans lequel se trouvera ce container

On défini le stockage de la base de donnée sur l'hôte

Ce container doit être construit depuis le Dockerfile
Pour surcharger la commande indiquée dans le Dockerfile

On défini que le répertoire actuel sera monté sous /app
Et que /app/node_modules est un volume du nom de dependencies

On bind le port 3003 de l'hôte sur le port 3000 du container

Ce container doit pouvoir accéder à notre premier container

Des variables d'environnements

Les réseaux dans lesquels se trouvera ce container

Définition des réseaux

Définition des volumes
(aucune options)

Docker Compose

Exemple : docker-compose.yml

```yaml
version: '3'
services:
  db-mongo:
    image: mvertes/alpine-mongo:latest
    networks:
      - back-tier
    volumes:
      - db-data:/data/db
  backend:
    build: .
    command: nodemon . --exitcrash
    volumes:
      - .:/app
      - dependencies:/app/node_modules
    ports:
      - "3003:3000"
    links:
      - db-mongo
    environment:
      - PORT=3000
      - NODE_ENV=development
    networks:
      - front-tier
      - back-tier
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
volumes:
  dependencies:
  db-data:
```

Version du fichier docker-compose (3 est la plus récente)

Définition des containers

Nom du container (aussi son hostname)

Image utilisée

Le nom du réseau dans lequel se trouvera ce container

On défini le stockage de la base de donnée sur l'hôte

Ce container doit être construit depuis le Dockerfile
Pour surcharger la commande indiquée dans le Dockerfile

On défini que le répertoire actuel sera monté sous /app
Et que /app/node_modules est un volume du nom de dependencies

On bind le port 3003 de l'hôte sur le port 3000 du container

Ce container doit pouvoir accéder à notre premier container

Des variables d'environnements

Les réseaux dans lesquels se trouvera ce container

Définition des réseaux

Définition des volumes
(aucune options)

```
version: '3'
services:
  db-mongo:
    image: mvertes/alpine-mongo:latest
    networks:
      - back-tier
    volumes:
      - db-data:/data/db
  backend:
    build: .
    command: nodemon . --exitcrash
    volumes:
      - .:/app
      - dependencies:/app/node_modules
    ports:
      - "3003:3000"
    links:
      - db-mongo
    environment:
      - PORT=3000
      - NODE_ENV=development
    networks:
      - front-tier
      - back-tier
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
volumes:
  dependencies:
  db-data:
```

Démarrer/Arrêter les services (en arrière plan)

```
> docker-compose up -d
> docker-compose down
```

Docker Compose

Utilisation

Lancer/construire les services

```
> docker compose up
```

Démarrer/Arrêter les services (en arrière plan)

```
> docker compose up -d
> docker compose down
```

Docker Compose

Get crazy !

Ajouter un service Load-Balancer (haproxy se débrouille)

```
load-balancer:
  image: dockercloud/haproxy:latest
  ports:
    - 8080:80
  depends_on:
    - backend
  links:
    - backend
  networks:
    - front-tier
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

Augmenter le nombre de container de backend et tester !

```
> docker-compose scale backend=4
> docker container ps
> curl http://127.0.0.1:8080
> curl http://127.0.0.1:8080
> curl http://127.0.0.1:8080
```

Docker Compose

TD : Docker-Compose

(Le fichier ci-contre ignore la partie mongo)

Regarder la doc de dockercloud/haproxy pour voir les bonnes variables d'environnement à mettre sur les services

Par défaut HAProxy utilise les `links` pour savoir quoi LoadBalancer et le port exposé dans le Dockerfile pour savoir sur quel port le load-balancer doit s'effectuer

```
version: '3'
services:
  backend:
    build: .
    volumes:
      - .:/app
      - dependencies:/app/node_modules
    networks:
      - front-tier
  load-balancer:
    image: dockercloud/haproxy:latest
    ports:
      - 8080:80
    depends_on:
      - backend
    links:
      - backend
    networks:
      - front-tier
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
networks:
  front-tier:
    driver: bridge
volumes:
 dependencies:
```

TP Dockerize

II. Docker

TP: Dockerize & docker-compose

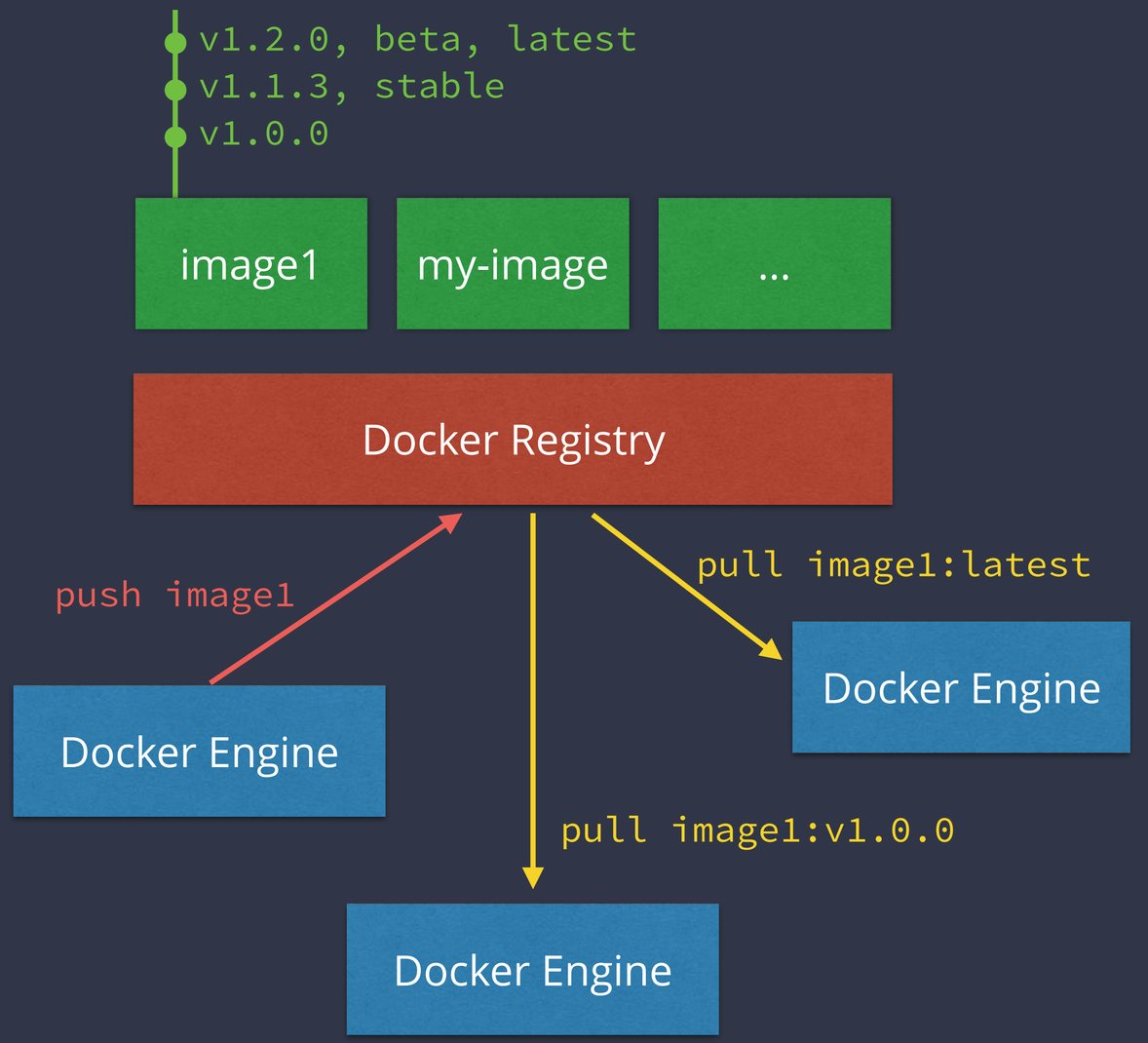
- Démarrez plusieurs containers dockercloud/hello-world au sein d'un docker compose
- Ces hello-world doivent être load balancés par dockercloud/haproxy
- Mettez en place Docker
- Vous devez rédiger au moins :
  - docker-compose.yml

Docker Registry

II. Docker

Vue d'ensemble

- Dépot d'image versionnées
- Notifications permettent de le lier avec des systèmes de CI/CD



Docker Registry

Installation et Utilisation

Rien de plus simple :

```
> docker run -d -p 5000:5000 -v le-volume-registry:/var/lib/registry --name registry registry:latest
```

Et on peut s'amuser :

```
> docker image ls
> docker tag mytest localhost:5000/mytest:latest
> docker push localhost:5000/mytest:latest
> docker image rm localhost:5000/mytest:latest
> docker image rm mytest
> docker image ls
> docker pull localhost:5000/mytest:latest
> docker image ls
```

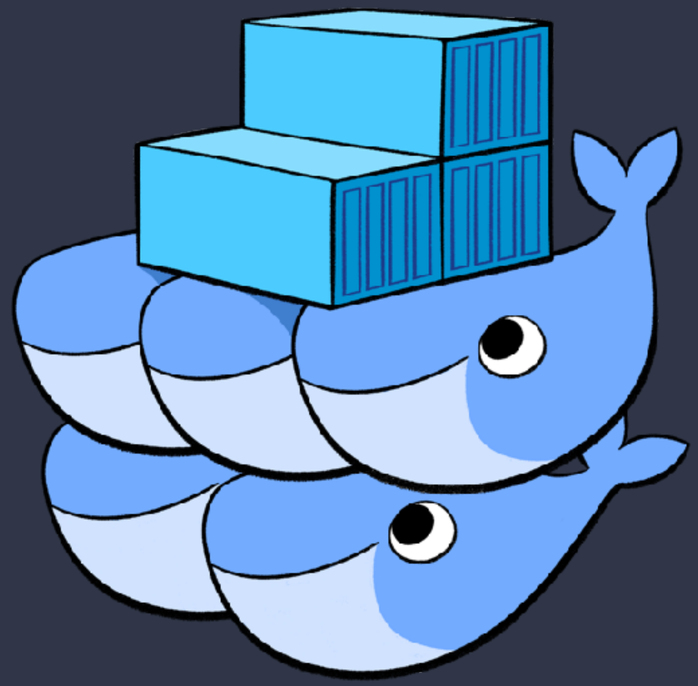
Docker Swarm

II. Docker

Docker Swarm

Vue d'ensemble

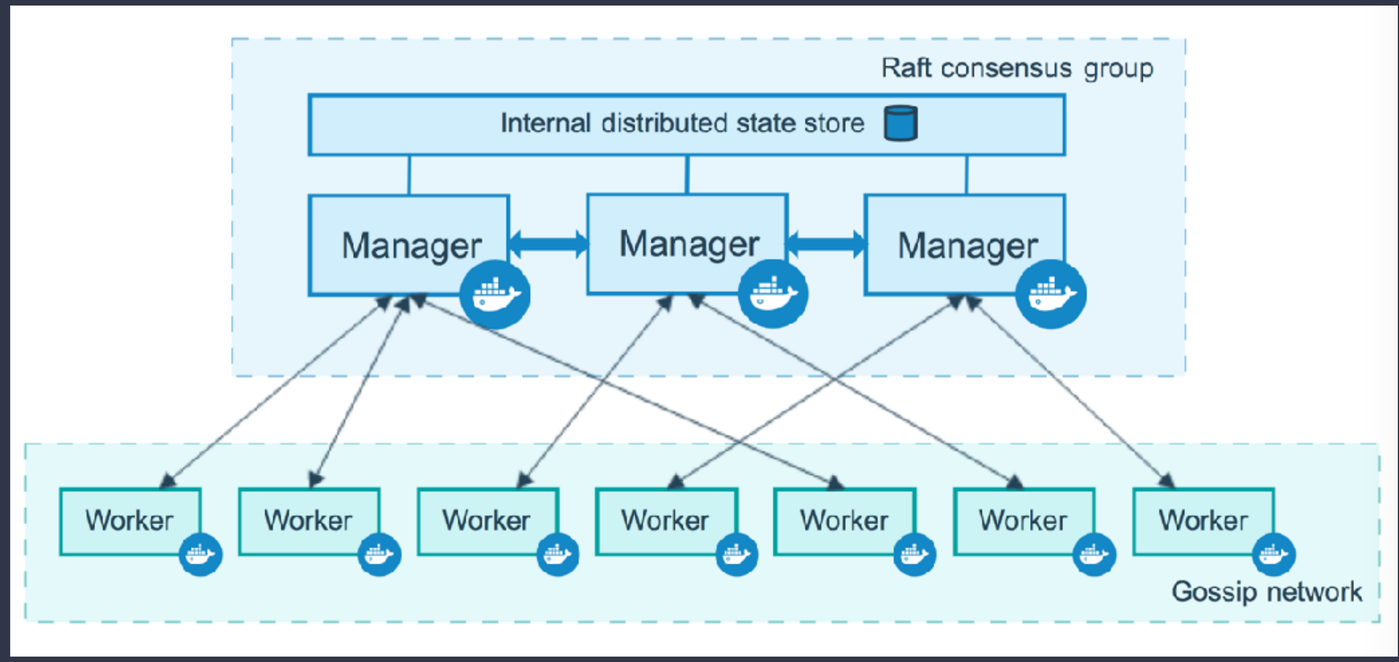
- Gestion de cluster
- Décentralisé
- Scaling
- LoadBalancing



Docker Swarm

En détail

- Cluster de Docker Engine sur différentes machines
- Docker Swarm décide où lancer les containers
- Division de la charge
- Deux types de noeuds : Manager et Workers



Docker Swarm

Manager Node

- Démarre et arrêtes les services
- Maintient le cluster de worker
- Fait office de Load Balancer
- Assure une haute disponibilité à raison d'une perte de manager possible à hauteur de (N-1)/2 (N étant le nombre de manager)

Docker Swarm

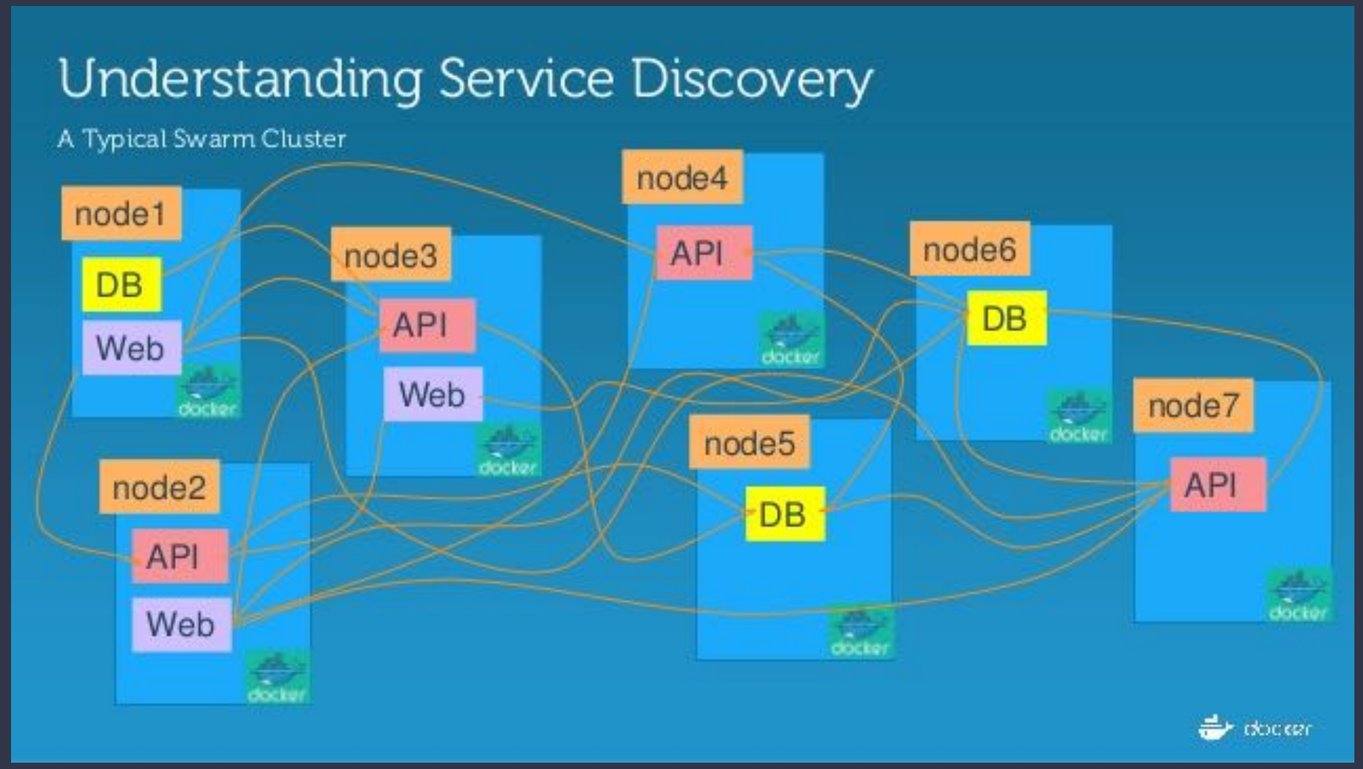
Worker node

- Aucun pouvoir décisionnel
- Un manager contient aussi un worker

Docker Swarm

Service Discovery

- Détection des états des différents noeuds du Swarm
- Embedded DNS
- Haute disponibilité



Docker Swarm

Implémentation

- 1 manager
- 3 worker (dont 1 est aussi manager)

Les labels

- Il est possible de flagguer les Node avec des labels
- Et donc de spécifier à un service quelles Nodes sont compatibles
- `constraint:storage==ssd`
- `constraint:node==aws-worker-1`
- `affinity:image==redis`
- ou encore des custom labels : `constraint:engine.labels.toto==lol`

Docker Swarm

Création VM AWS

- Créez 1 VM Amazon basé sur une image Ubuntu ou si disponible sur "ami-881a3fee" (si ça fonctionne, pas besoin du slide suivant)
- Le groupe de sécurité commun doit avoir les ports suivants ouverts :

Type ⓘ	Protocole ⓘ	Plage de ports ⓘ	Source ⓘ
Règle TCP personnalisée	TCP	2377	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0
Règle TCP personnalisée	TCP	4789	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Règle UDP personnalisée	UDP	7946	0.0.0.0/0
Règle TCP personnalisée	TCP	7946	0.0.0.0/0
Règle UDP personnalisée	UDP	4789	0.0.0.0/0

Implémentation

Initialisez le Swarm sur votre Manager (une des trois machines)

```
> docker swarm init
> docker swarm join-token worker
```

La dernière commande çi-dessus vous indique comment ajouter une machine dans le swarm.
Joigner le Swarm depuis vos deux autres machines en executant cette commande sur les deux autres machines

Vérifier votre Swarm depuis votre Manager

```
> docker node ls
```

Docker Swarm
Implémentation

Lancez un service Hello World sur vos workers

Lancez cette commande sur le manager

```
> docker service create -p 80:80 --name app --constraint "node.role != manager" --replicas 2 dockercloud/hello-world
```

Testez

```
> curl http://ip-du-manager/
```

Docker Swarm
Implémentation

Créez un network sur le swarm

```
> docker network create -d overlay proxy
```

Lancez haproxy sur le manager uniquement

```
> docker service create --name haproxy --network proxy --mount target=/var/run/docker.sock,source=/var/run/docker.sock,type=bind -p 80:80 --constraint "node.role == manager" dockercloud/haproxy
```

Lancez un service Hello World sur vos workers

```
> docker service create --name app --network proxy --constraint "node.role != manager" dockercloud/hello-world
```

Scalez votre service

```
> docker service scale app=2
```

Testez

```
> curl http://ip-du-manager/
```

Docker Swarm
STACK

- Utiliser des fichiers docker-compose sur le swarm

```
> docker stack deploy --compose-file ma-stack.yml ma-stack
> docker stack ls
> docker stack ps ma-stack
> docker stack rm ma-stack
```

Définition des containers
Nom du container (aussi son hostname)
Image utilisée

Le nom du réseau dans lequel se trouvera ce container

On défini le stockage de la base de donnée sur l'hôte

La clé deploy défini les spécifications de notre stack
Lancée en 2 exemplaire
Les spécs du déploiement
Déploiement en parallèle
Délais de 10s entre le déploiement de chaque réplicas

Limitation de ressources
MAX 10% du cpu
MAX 64Mb de ram

MIN 1% de CPU
MIN 16M de Ram

Quand est-ce que les containers doivent se relancer

```yaml
version: '3.3'
services:
  db-mongo:
    image: mvertes/alpine-mongo:latest
    networks:
      - back-tier
    volumes:
      - db-data:/data/db
    deploy:
      replicas: 2
      update_config:
        parallelism: 1
        delay: 10s
      resources:
        limits:
          cpus: "0.1"
          memory: 64M
        reservations:
          cpus: '0.01'
          memory: 16M
      restart_policy:
        condition: on-failure
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
volumes:
  db-data:
```

Docker Stack

Exemple 1 : ma-stack.yml

Docker Stack

Exemple 2 LOAD BALANCER

En mode GLOBAL, le service ne lancera qu'un seul container par Node

Le service ne se lancera que sur les NODE qui possèdent ces contraintes
Il est possible de donner des LABELS custom à nos NODES

On bind le port en mode INGRESS, cela veut dire que DOCKER ne cherchera pas à LOAD BALANCER ce service

```
> docker network create -d overlay proxy
```

On le lance au sein du network externe PROXY, défini par la commande ci-dessus

```yaml
version: "3.3"
services:
  haproxy:
    image: dockercloud/haproxy
    deploy:
      mode: global
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
      update_config:
        parallelism: 2
        delay: 10s
      placement:
        constraints:
          - "node.labels.loadbalanced == true"
          - "node.role == manager"
    ports:
      - target: 443
        published: 443
        protocol: tcp
        mode: ingress
    volumes:
      - '/var/run/docker.sock:/var/run/docker.sock'
    networks:
      - proxy
    environment:
      - RELOAD_TIMEOUT=-1
      - HEALTH_CHECK=check inter 2000 rise 2 fall 3
      - EXTRA_GLOBAL_SETTINGS=spread-checks 5
networks:
  proxy:
    external: true
```

ICI on utilise le network PROXY géré par le load balancer, et les variables d'environnement seront utilisées par HAPROXY

Docker Stack

Hello-World load balanced

```yaml
version: "3.3"
services:
  lol:
    image: dockercloud/hello-world
    deploy:
      replicas: 4
      update_config:
        parallelism: 1
        delay: 4s
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
        reservations:
          cpus: '0.001'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
      placement:
        preferences:
          - spread: node.labels.location
    environment:
      - SERVICE_PORTS=80
      - VIRTUAL_HOST=http://mondomain.tk
    networks:
      - proxy
networks:
  proxy:
    external: true
```

III. Ansible

Au préalable

III. Ansible

Au préalable

Vue d'ensemble

- Gestion de configuration
- Langage d'automatisation : Ansible
  Playbook => Simple
- Automation Engine
- Agentless

Au préalable

Préparation

- Assurez-vous d'avoir trois machines distincte
- ex: 3 ubuntu sur AWS ou 2 ubuntu AWS + votre machine ou 3 VM ubuntu
- Une machine de contrôle
- Deux machines à administrer

Au préalable

Installation

- Seulement nécessaire sur la machine de contrôle

installation par Python (nécessite PIP)

```
sudo pip install ansible
```

Pou

```
sud
sud
sud
sud
```

- Vérifiez que python est installé sur toutes vos machines

Inventory

III. Ansible

Inventory

Hosts & Groups

- Ansible administre un parc de machine
- Listé dans un fichier `inventory.ini`
- Possibilité de définir des groupes

Exemple :

```
other.mydomain.com

[front]
www.mydomain.com
www.stage.mydomain.com

[back]
api.mydomain.com
api.stage.mydomain.com

[stage]
api.stage.mydomain.com
stage.mydomain.com

[cdn]
cdn[01:10].mydomain.com
```

Configurer ce fichier avec les IPs de vos deux machines AWS. Séparez les dans deux groupes distincts et regroupez les dans un troisième groupe.

Inventory

Hosts & Var

- Possible de définir des variables pour les groupes ou les hosts individuels
- Ré-utilisation de ces variables lors des tâches à utiliser

Exemple :

```
[back]
api.mydomain.com
api.stage.mydomain.com auth=true

[stage]
api.stage.mydomain.com
stage.mydomain.com

[stage:vars]
environment=staging
```

Utilisation

III. Ansible

Utilisation

Préparation

1. Générez depuis votre machine de contrôle un nouveau couple clé privée/public ssh-keygen -t rsa

2. Dans ~/.ssh vous avec id_rsa (clé privée) et id_rsa.pub (clé public), affichez la clé publique : cat ~/.ssh/id_rsa.pub

3. Sur les autres machines, faites un nano ~/.ssh/authorized_keys et ajoutez la clé PUBLIQUE de la machine de contrôle sur une nouvelle ligne. Votre machine de contrôle peut désormais se connecter librement en ssh sur vos autres machines.

4. Sur vos autres machines, python est nécessaire, installez-le s'il n'est pas déjà là :
sudo apt -y update && sudo apt install -y python-minimal

5. Depuis la machine de contrôle, essayez de vous connecter à vos autres machines depuis ansible :
ansible all -i inventory.ini -u ubuntu -a "echo hello"
ou
ansible all -i inventory.ini --private-key=your-key.pem -u ubuntu -a "echo hello"

Résultat :

```
34.248.76.204 | SUCCESS | rc=0 >>
hello

34.250.152.200 | SUCCESS | rc=0 >>
hello
```

La commande ansible

```
ansible all --private-key=./devops.pem -u ubuntu -a "echo hello"
```

groupes / host-pattern

Options

```
-a "command args"          Execute une commande
-m module                  Utilise un module Ansible
--sudo                     Execute en SUDO
--private-key=file         Utilise une clé privée pour l'authentification
-u user                    L'utilisateur utilisé pour l'authentification
-f 5                       Nombre à exécuter en parallèle (5 par défaut)
```

Utilisation

Les modules

- Les modules regroupent des actions et comportements
- Le module par défaut est "command"
- Possible de créer ces propres modules
- Ou d'en utiliser un de la liste : http://docs.ansible.com/ansible/service_module.html

Faire un ping sur les machines (module ping)
ansible all -u ubuntu -m ping

Redémarrer Apache (module service)
ansible all-u ubuntu -m service -a "name=apache state=restarted"

Playbook

III. Ansible

Introduction

- Un playbook est un fichier contenant la définition de ce qui doit être fait
- Un playbook peut ensuite être exécuté sur un ensemble de machine grâce à Ansible
- Les playbooks sont simples à lire et à écrire au format YML

Exemple (play-mongo.yml)

```yaml
---

- name: Install and start mongodb
  hosts: db
  remote_user: ubuntu
  become: true
  tasks:

    - name: install MongoDb
      apt:
        name: mongodb
        update_cache: yes

    - name: start MongoDb
      service:
        name: mongodb
        state: restarted
```

```
ansible-playbook -i inventory.ini play-mongo.yml
```

apt et service sont des modules

Playbook

Les tâches

- Chaque tâche est exécutée de façon séquentielle
- En cas d'erreur, l'execution s'arrête pour le host en défaut
- Une tâche exécute un module avec des arguments

Exemple (extrait du slide précédent) :

```
- name: install MongoDb
  apt:
    name: mongodb
    update_cache: yes

- name: start MongoDb
  service:
    name: mongodb
    state: restarted
```

Playbook

Les handlers

- Les handlers sont exécutés après que les tâches ont toute été réalisées
- Chaque handler ne sera exécuté qu'une seule fois
- Pour qu'un handler s'exécute, une tâche doit l'avoir notifié
- Permet d'exécuter une seule fois une action demandée par plusieurs tâches

Exemple (extrait)

```
tasks:
  - name: install MongoDb
    apt:
      name: mongodb
      update_cache: yes

  - name: configure MongoDb
    template:
      src: ./mongod.conf
      dest: /etc/mongod.conf
    notify: restart mongodb

handlers:
  - name: restart mongodb
    service:
      name: mongodb
      state: restarted
```

Playbook

Les rôles

- Les rôles permettent de subdiviser le Playbook en une multitude de fichiers

```
site.yml
roles/
   mongo/
     files/
     templates/
       mongod.conf.j2
     tasks/
       main.yml
     handlers/
       main.yml
     vars/
     defaults/
     meta/
```

site.yml

```yaml
---

- name: install and start mongodb
  hosts: dbservers
  remote_user: root
  become: true

  roles:
    - mongo
```

roles/mongo/tasks/main.yml

```yaml
---

- name: install MongoDb
    apt:
      name: mongodb
      update_cache: yes

- name: configure MongoDb
  template:
    src: mongod.conf.j2
    dest: /etc/mongod.conf
    notify: restart mongodb
```

Ansible Galaxy

- Gestionnaire de rôle pour Ansible
- Communauté OpenSource
- Rôles configurables

site.yml

```
ansible-galaxy install bennojoy.mysql
```

play.yml

```
---
- name: "Provision MySQL Master server"
  hosts: mysql
  remote_user: debian
  become: true
  roles:
    - role: bennojoy.mysql
      mysql_db_id: 1
      mysql_db:
        - name: project-dev
          replicate: no
      mysql_users:
        - name: my-user
          pass: the-pass-of-my-user
          priv: "project-dev.*:ALL"
      mysql_root_db_pass: Th3R00TP@ssOfMy$ql
      mysql_bind_address: "0.0.0.0"
      mysql_repl_user: []
```

Ansible

with_items

- Permet de faire des Itérations

sans loop

```
---
- name: add user testuser1
  user:
    name: "testuser1"
    state: present
    groups: "wheel"
- name: add user testuser2
  user:
    name: "testuser2"
    state: present
    groups: "wheel"
```

avec loop

```
---
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  with_items:
     - testuser1
     - testuser2
```

avec loop et variables

```
---
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  with_items: '{{ users }}'
```

Ansible

conditions

```
---
tasks:
  - shell: echo "only on Red Hat 6,
derivatives, and later"
    when:
      - ansible_os_family == "RedHat"
      - ansible_lsb.major_release|int >= 6

  - shell: echo "only on Debian or Ubuntu"
    when: ansible_distribution == "Debian"
or ansible_distribution == "Ubuntu"

  - shell: echo "only on Debian or Ubuntu"
    when: ansible_os_family == "Debian"
```

TD

Ansible

Ansible

TD

- Créez un playbook avec roles qui installe vos machines avec :
  - un serveur FTP
  - MySQL
  - Apache avec PHP d'activé
- Créez un playbook qui permet de déployer un Wordpress sur vos machines

=> Essayez d'utiliser au maximum les variables pour ne pas faire tout à fait la même chose selon la machine, rechercher `host_vars` et `group_vars`

IV. CI / CD

Git, Gitflow et PR

III. CI / CD

Git

- Gestionnaire de version décentralisé
- Simple et performant
- Décentralisé contrairement à CVS ou SVN
- De nombreux GUI
- De nombreux services d'hébergements

Git notions de bases

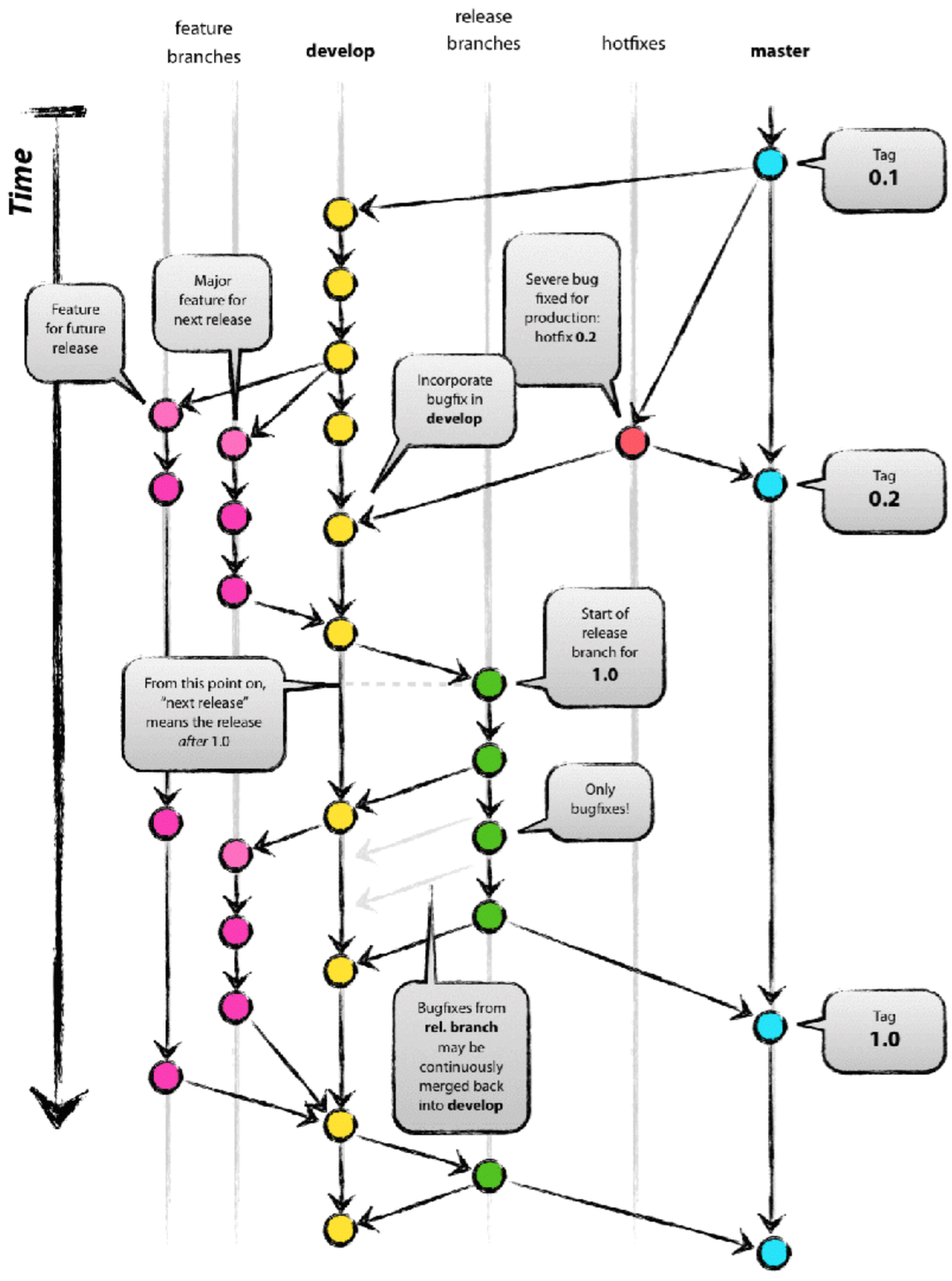
Initialise git dans le dossier en cours	`git init`
Affiche l'état des fichiers dans le git	`git status`
Clone un repo distant	`git clone http://xxx/yyy.git`
Lie à une remote nommée `origin`	`git remote add origin http://xxx/yyy.git`
Ajoute tous les fichiers nouveaux/modifiés dans Git	`git add .`
Créé un nouveau commit identifié par un hash	`git commit -am "Message de commit"`
Modifie le dernier commit	`git commit --amend`
Créé une nouvelle branche à partir de la branche actuelle	`git checkout -b my-new-branch`
Envoi la branche "master" sur "origin"	`git push origin master`
Récupère la branche "master" depuis "origin"	`git pull origin master`
Créé un tag vers le commit actuel	`git tag v1.0.0`
* Bascule sur un tag *	`git checkout v1.0.0`
Bascule sur une branche	`git checkout my-new-branch`
* Bascule sur un commit *	`git checkout [commit hash]`
Supprime toute modification non commitée	`git reset --hard HEAD`
Revient à l'avant dernier commit	`git reset --hard HEAD~1`
Sauvegarde temporairement les modifications en cours	`git stash`
Ré-applique les modification temporairement sauvegardée	`git stash apply`
Permet d'éditer, d'amender ou de supprimer des commits depuis un certain commit	`git rebase [commit hash] -i`

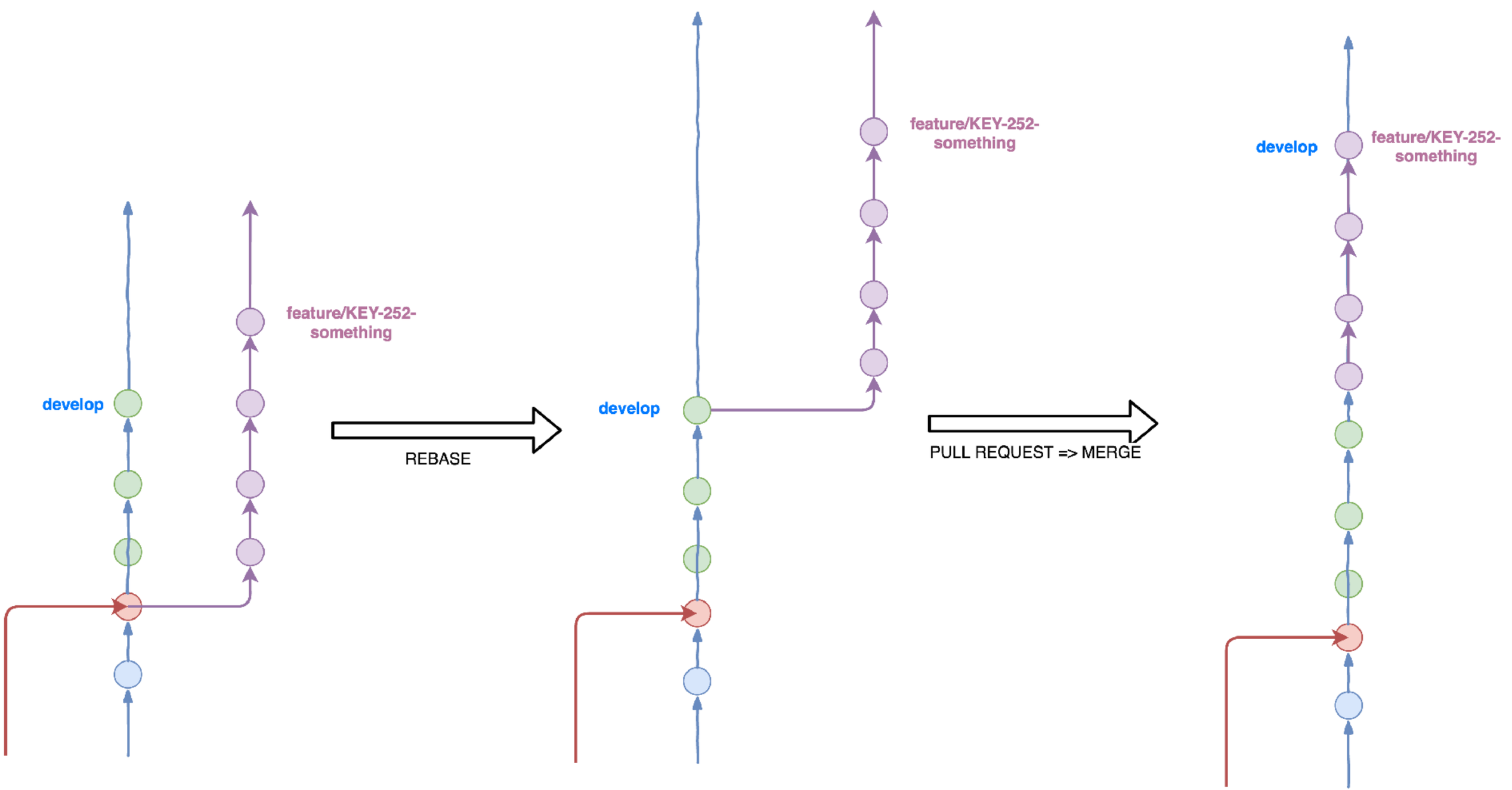
* Lors du checkout d'un Tag ou d'un Commit, un `git checkout -b xxx` sera nécessaire afin d'enregistrer toute modification *

Git, Gitflow et PR

Gitflow

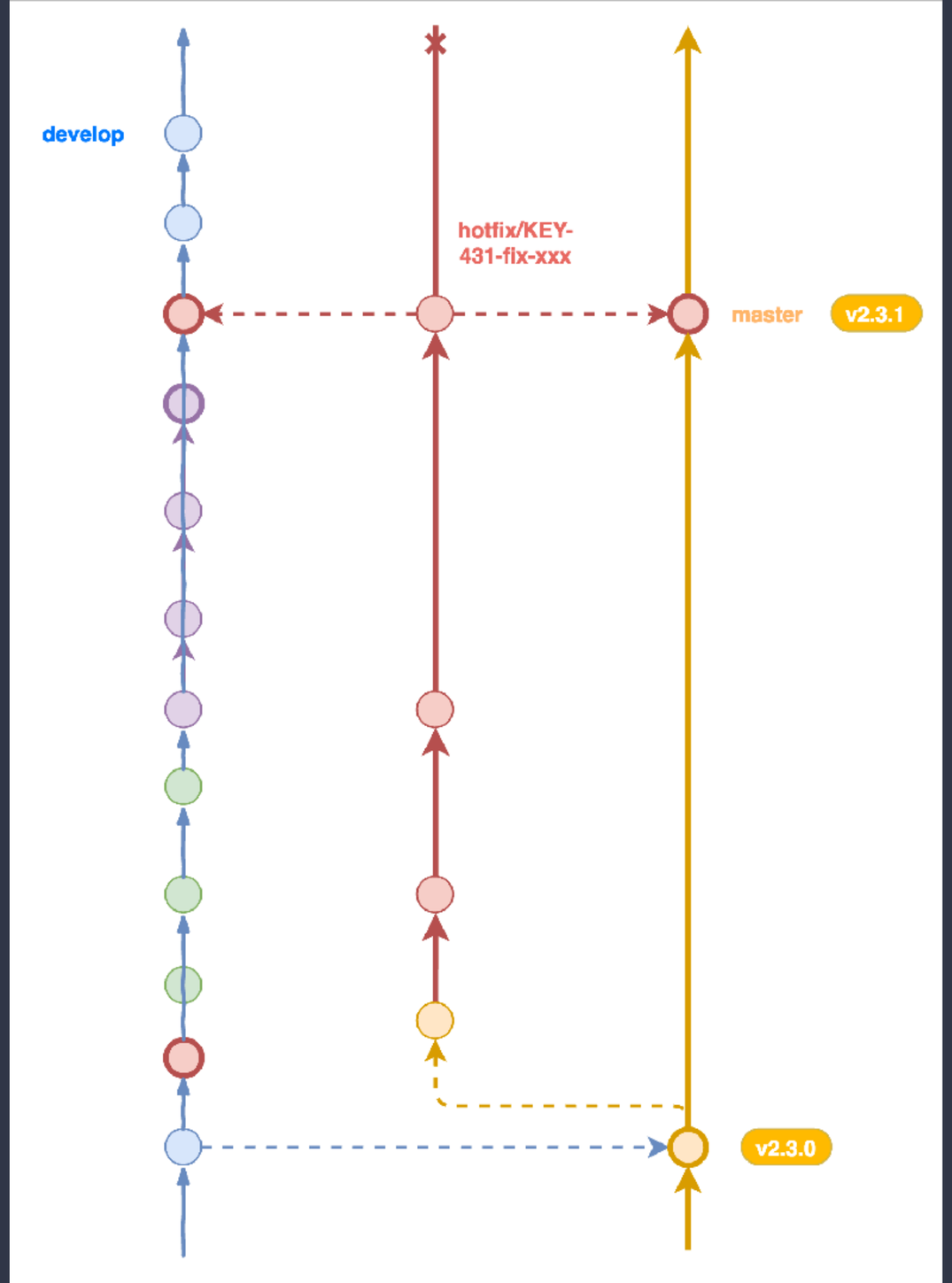
- Une CLI existe : git flow
- Modèle performant totalement adapté au CI/CD
- master => Branche de production
- develop => Branche de développement
- feature/xxx => Développement de la feature xxx
- release/v1.0 => Préparation d'une release
- hotfix/myfix => Préparation d'un patch




develop
feature/KEY-252-something
REBASE
develop
feature/KEY-252-something
PULL REQUEST => MERGE
develop
feature/KEY-252-something

Git, Gitflow et PR

Gitflow



Les versions

- Chaque entité à son propre système de versionning
- Souvent : major, minor, patch (et éventuellement build)
- v3.4.2 (Version majeure 3, mineure 4, et 2 patchs appliqués)
- Version majeur : incrémentée lors de grands changements (peut indiquer aussi une non-rétrocompatibilité)
- Version mineure : incrémentée à chaque release (ajout de fonctionnalité, plusieurs changements)
- Version patch : incrémentée lors de l'application de hotfix

Les pull request

- L'ouverture d'un ticket de demande de merge
- Permet de discuter des changements au niveau du code, de rajouter des choses qui manquent
- Finalement une fois que la pull request a été acceptée, elle est acceptée et le code est mergé sur la branche principale
- Une pull request peut venir d'une branche internet comme d'une branche d'un Fork
- Github, Bitbucket et Gitflow ont tous trois des mécanismes de pull request

Bitbucket pipeline

III. CI / CD

Bitbucket pipeline

Le service

- Gratuit est entièrement intégré (limité à 1 build simultanée dans sa version gratuite)
- Suffisamment configurable
- Permet d'effectuer du CI avec un retour instantané au développeur (notifications)
- Permet d'executer des batteries de tests
- Permet d'executer du CD en appelant un script Ansible par exemple

Bitbucket pipeline

Écrire une pipeline

bitbucket-pipelines.yml

Image Docker à utiliser

Pipeline par défaut (pour tout commit)

Les scripts à executer, en cas d'erreur, l'execution s'arrête et une notification est envoyée

Il est aussi possible de définir des pipelines plus spécifique, par exemple pour les branches release/xxxx

On a aussi accès à des variables d'environnements comme BITBUCKET_COMMIT ou BITBUCKET_BRANCH

```yaml
image: spittet/node-mongodb:4.6.0

pipelines:
  default:
    - step:
        script:
          - npm install
          - service mongod start
          - npm test

  branches:
    release/*:
      - step:
        script:
          - npm install
          - service mongod start
          - npm test
          - VERSION=$(echo ${BITBUCKET_BRANCH} | sed s,release\/,,)
          - npm run build $VERSION-${BITBUCKET_COMMIT}
          - npm run deploy-staging $VERSION-${BITBUCKET_COMMIT}
```

Félicitations !!

Cours WIK-DEVOPS-303 burned :)