Simplifier vos devs et vos tests avec docker

Paul Leclercq @polomarcus
Arnaud Bailly @dr_c0d3

PARISTESTCONF

PARISTESTCONF

Tester avec docker-compose

PARISTESTCONF

Data Engineer @ TABMO LABS

PARISTESTCONF

Développeur **backend** spécialisé dans les systèmes distribués

> “The only difference with a developer is you might spend a bit more time in your database/system config files or documentation, one of my running joke is that I'm not a data engineer but a **configuration engineer**”

Technologies/mots clés: Python, Scala, Java, SQL, Kafka, Spark, AirFlow, Entrepôt de données (BigQuery, RedShift, Snowflake...), Data Lake (AWS S3, GCP cloud storage...), NewSQL, mentalité **DevOps++**

En savoir plus: What I Like About Data Engineering

Mes problématiques de tous les jours

PARISTESTCONF

- Prendre le temps de **tester** ses versions d'app contre des systèmes variés
- Etre **autonome** sur les services externes (APIs, databases)
- Faire des **montées de versions** des systèmes externes
- **Automatiser** mes builds sur mon environnement d'intégration continue
- Gérer des apps auxquelles **je n'ai jamais touché**
- Ne pas polluer les données de tests
- Me sentir à l'aise pour faire "n'importe quoi" sur mes serveurs de tests
- **Faire utiliser des systèmes inconnus à d'autres** développeurs

PARISTESTCONF

Ce que je faisais avant docker-compose

PARISTESTCONF

```
> curl http://apache.crihan.fr/dist/kafka/2.2.0/kafka_2.12-2.2.0.tgz
> tar -xzf kafka_2.12-2.2.0.tgz
> cd kafka_2.12-2.2.0
> zookeeper-server-start.sh config/zookeeper.properties
> kafka-server-start.sh config/server.properties
> kafka-topics.sh --create --bootstrap-server localhost:9092
--replication-factor 1 --partitions 1 --topic test
> kafka-topics.sh --list --bootstrap-server localhost:9092
```

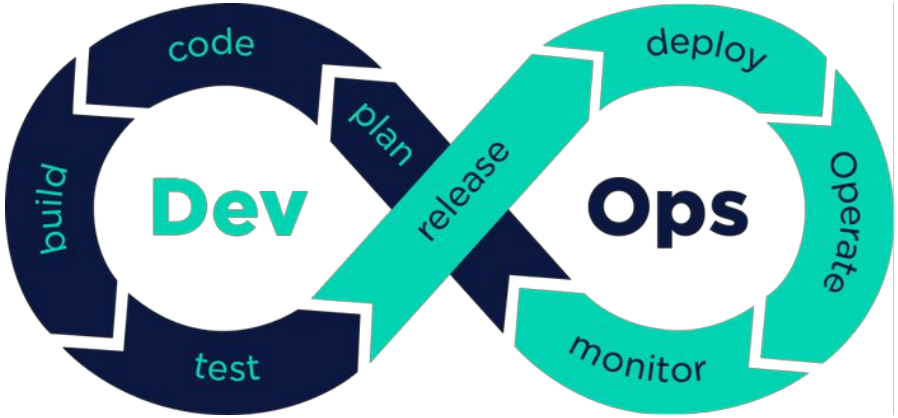
source kafka “quick start”

Docker, c’est quoi?

But: Avoir le même code applicatif en dev ou en production

Selon moi (dev) plus un outil pour développeurs, qu’un outil d’ops

- Responsabilise les devs sur la production
- Agilité DevOps

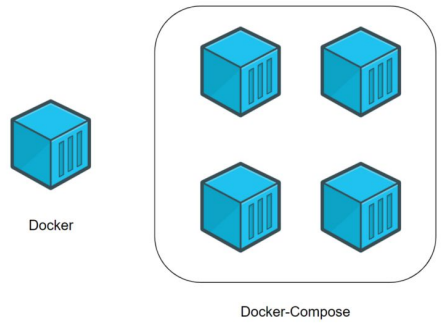


Docker compose, c’est quoi?

“Compose is a tool for defining and running multi-container Docker applications.”

```
docker-compose -f stack.yml up
docker-compose -f stack.yml down
```

```
services:
 db:
   image: mariadb:lastest
   restart: always
   volumes:
    - src/db/mysql:/var/lib/mysql
    - ./sql_migration.sql:/docker-entrypoint.d/sql_migration.sql
   environment:
     - MYSQL_ROOT_PASSWORD
     - MYSQL_USER
     - MYSQL_PASSWORD
     - MYSQL_DATABASE
   expose:
     - "3306"
```

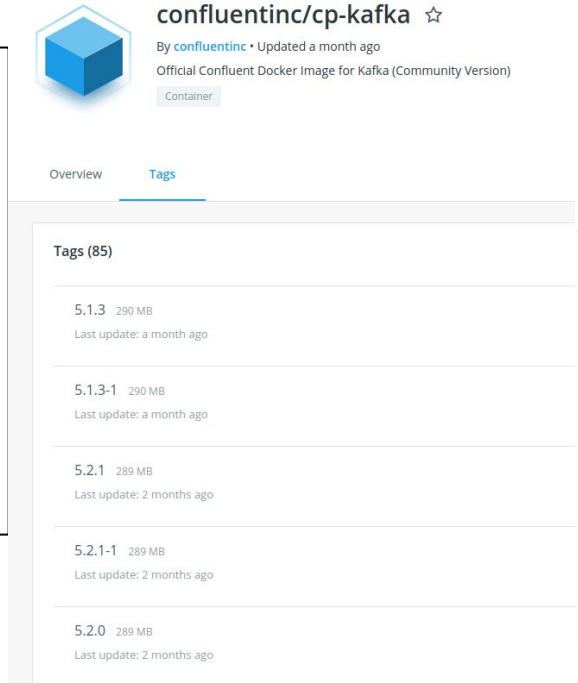


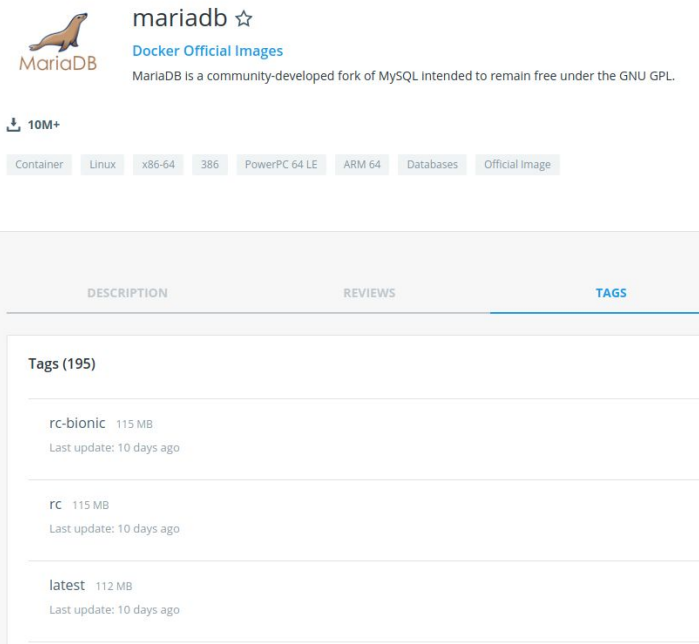
Docker compose, c’est quoi?

PARISTESTCONF

Gestion des versions https://hub.docker.com/

```
services:
 db:
  image: mariadb:10.4
 kafka:
  image:
confluentinc/cp-kafka:5.2.1
 app:
  build: .
```





Docker compose, c’est quoi?

PARISTESTCONF

Créer des mini serveurs locaux AKA ***containers*** auxquelles on peut se connecter

```
$ docker ps # get all my containers
$ docker exec -ti my_container_name bash
> ls

# consulter les logs du serveur
$ docker logs my_container_name
```

Docker compose, pourquoi? onboarding - projets

PARISTESTCONF

Prerequisites

Burrow is written in Go, so before you get started, you should install and set up Go.

If you have not yet installed the Go Dependency Management Tool, please go over ther
instructions. dep is used to automatically pull in the dependencies for Burrow so you do

Build and Install

```
$ go get github.com/linkedin/Burrow
$ cd $GOPATH/src/github.com/linkedin/Burrow
$ dep ensure
$ go install
```

→ 1 heure plus tard toujours entrain de galérer avec ma config (linkedin/Burrow)

Development Quick Start

```
# Start server with 10 observations
make install-with-data

# verify you can get all observations http://localhost/get_issues.php
```

3 minutes: up and running

vigilo

Docker compose, pourquoi? en réalité

PARISTESTCONF

“Merci Paul, mais j’allais pas m’amuser à tester Kafka en local, je dois livrer, j’ai envoyé directement en prod”

“Si si j’ai testé mais seulement en version Y”

- **Erreurs**: “Comment on fait pour supprimer ces données?”
- **Frustration** : “Cette techno X c’est de la merde!”

Faire utiliser des systèmes inconnus à d’autres développeurs

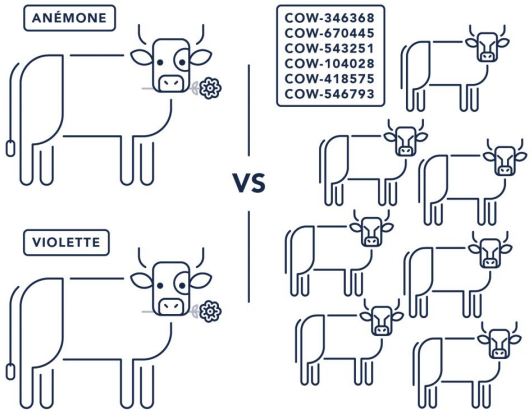
Docker compose, pourquoi? Intégration Continue

PARISTESTCONF

“Cattle VS Pet”

Avant: **MON** serveur de BD “violette.dev” dont tout le monde prend soin toute l’année comme un animal de compagnie

Aujourd’hui: un container de ma DB créé par
un build de la CI / ou en dev local comme du bétail



« pet vs. cattle », de l’artisan du serveur à l’artisan codeur Octo

Docker compose, pourquoi? Intégration Continue

PARISTESTCONF

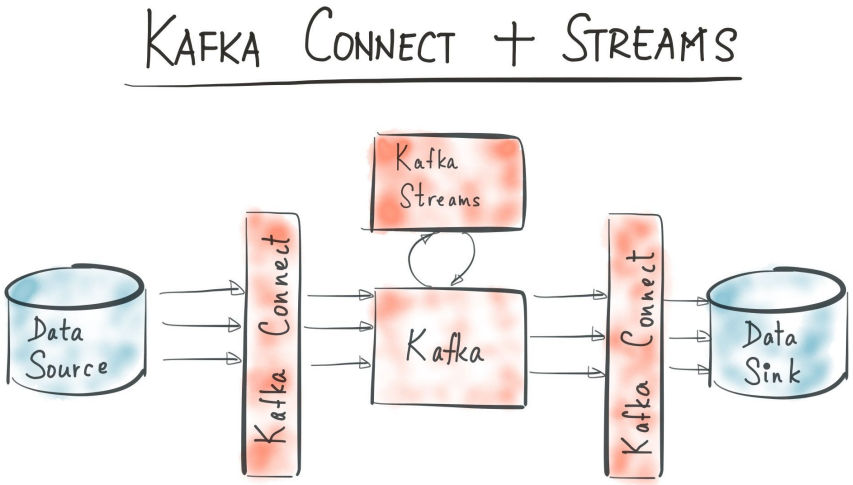
```
language: scala
services:
 - docker
before_install:
 - docker-compose -f src/test/resources/docker-compose.yml up -d
 - ./init-stack-kafka-with-data.sh # populate kafka
script:
 - sbt run
 - ./test-message-kafka.sh # end to end tests
```

on gagne en sérénité en quelques lignes

projet: kafka-streams-scala-examples

Le meilleur ami du backend/data engineer

Problématique : 11 services à faire communiquer ensemble



Demo: https://asciinema.org/a/WpJf8Yi7d9ILWFI12vToNwCyD

Le meilleur ami du frontend

PARISTESTCONF

Problématique :

Mon app dépend d'une API, elle crash depuis X sur mon serveur Y

L'équipe API publie des images à disposition sous différentes versions (prod/branche spécifique/...)

Bonus si c'est automatisé via l'intégration continue: GitlabCI et son docker registry

Le meilleur ami du PO testeur agile

PARISTESTCONF

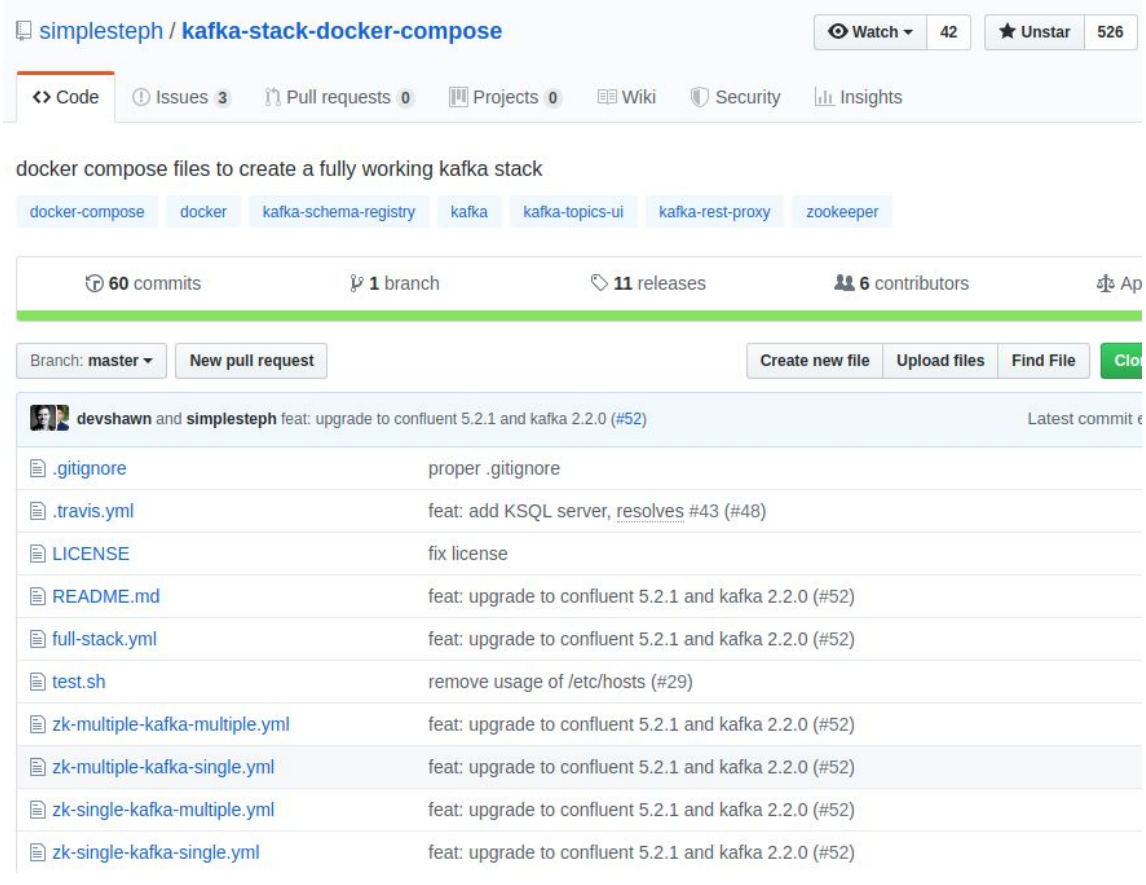
Problématique :

Comment tester sans attendre la livraison la fonctionnalité déployée?

Au choix:

- L’équipe publie une image sur un serveur distant
- Le PO utilise une commande docker 😮

Trouver des projets recettes



Docker-compose en prod?

C’est possible, avec une stack spécifique docker et de l’expertise interne

- https://docs.docker.com/compose/production/
- Kubernetes

Conclusion

PARISTESTCONF

- **La rapidité pour monter une stack**, nous empêche d'être fainéant lors de nos tests, et augmente la qualité et la durée de vie des projets : plus de tests écrits, intégration continue poussée...

- **Prise en main des projets** facilitée du débutant au sénior

Test d'un système réparti

Ingénieur logiciel (très) senior

PARISTESTCONF

Développeur depuis plus de 20 ans

- *initialement* : Java, J2EE, OOP, Design patterns…
- *puis* : eXtreme Programming, TDD, Qualité logicielle, Process de développement, Agile…
- *enfin* : Haskell, Functional Programming, Functional Architecture, Distributed Systems, Docker…

Lead Software Engineer chez **Symbiont**

- Développeur dans l'équipe plate-forme
- Lead dev d'une application de gestion de prêts immobiliers

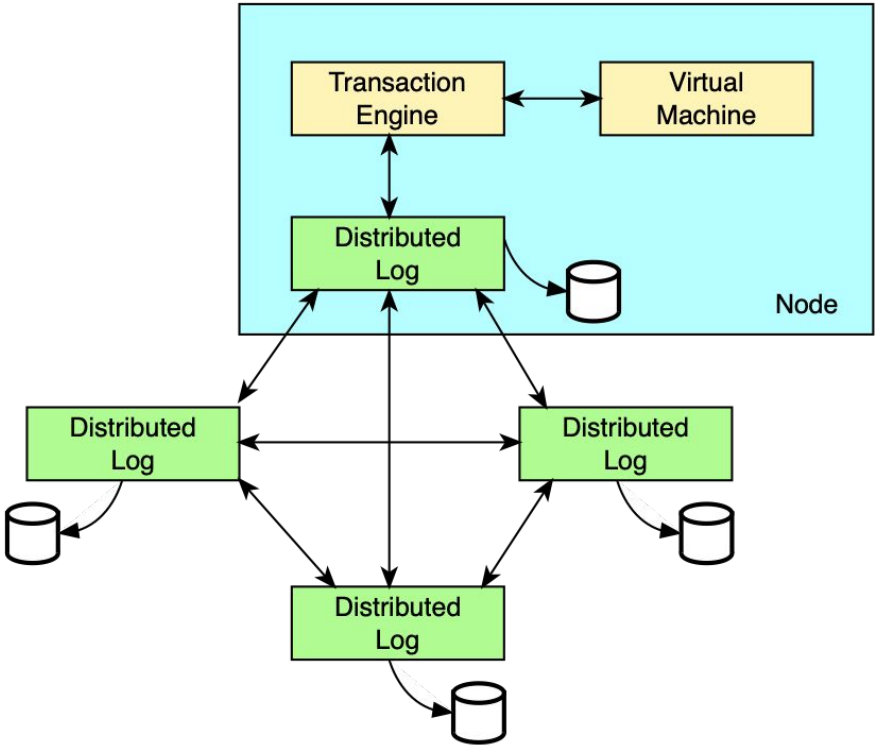
symbiont

PARISTESTCONF

Startup US développant une solution de **Smart Contracts**

- Système réparti de type *Blockchain privée* : l'accès au réseau est contrôlé et limité (consortium)
- Basé sur un protocole de consensus réparti et le chiffrement asymétrique des transactions
- Permettant le déploiement et l'exécution de *contrats* de manière asynchrone dans une *machine virtuelle* déterministe
- Nom de code : *Assembly*

Architecture de *Symbiont Assembly*



Architecture de *Symbiont Assembly*

PARISTESTCONF

- Un journal de transactions réparti et tolérant aux fautes byzantines (basé sur le protocole BFT-Smart)
- Un moteur de transaction gérant les identités de chaque noeud du réseau et la confidentialité des transactions
- Une machine virtuelle stockant des contrats et exécutant des appels de fonctions
- L'ensemble des composants est packagé sous forme d'images *Docker* et la solution déployée sur un cluster *Kubernetes*

Propriétés du système *Assembly*

PARISTESTCONF

Sûreté

Le système doit supporter des erreurs arbitraires et résister à la malveillance potentielle de certains noeuds

Cohérence

Le journal de transaction doit être identique sur tous les noeuds, aux délais de transmission près

Sécurité

Le système garantit la confidentialité des transactions

Le problème

- Les temps de déploiement d'un noeud ou d'un réseau sont longs et allongent la boucle de feedback au cours du développement
- On voudrait pouvoir tester les propriétés de *cohérence* et de *résilience* du système
- On voudrait pouvoir écrire des tests permettant de manipuler la plate-forme "simplement"
- … en s'abstrayant des détails du déploiement
- … tout ça avec une infrastructure minimale (eg. un poste de développeur)

Source d'inspiration : Jepsen

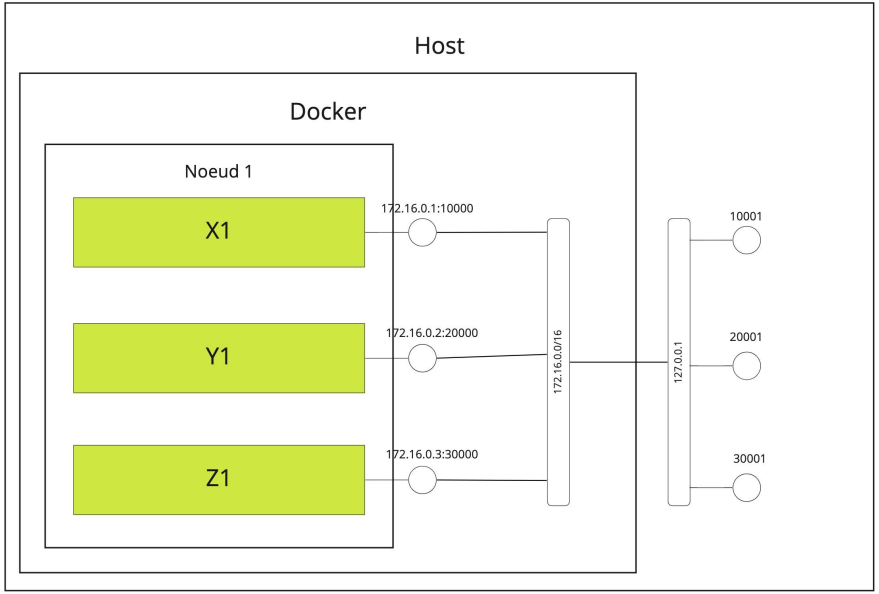
Principes des tests

PARISTESTCONF

- Déployer tout ou partie d'un noeud comme des containers *docker* sur un *réseau dédié*
- Écrire un scénario de test utilisant un langage dédié embarqué dans Haskell
- Manipuler la topologie et les propriétés de QoS du réseau de manière de manière à *injecter des erreurs*
- Vérifier les propriétés de cohérence du système

Déploiement

- Les conteneurs sont reliés par un réseau privé géré par docker
- Chaque conteneur est aussi exposé sur un port unique sur le réseau de la machine hôte
- Avoir un réseau indépendant permet de manipuler les tables de routages de chaque machine



Un langage spécialisé pour les tests

PARISTESTCONF

```
test
  :: Int -> Test e ()
test numNodes = do
  when (numNodes < 5) $
    exit "Split brain test should be run with at least 5 nodes"
  (adminKey, procs) <- setup "assembly" numNodes
  let mid = numNodes `div` 2
  run1 <- mconcat <$>
          feedTransactions 1  (10 ///) (batchOf 10) (seconds 10)
  nemesis $ Partition [[ 1 .. mid - 1 ], [ mid .. numNodes ]]
```

Un langage spécialisé pour les tests

PARISTESTCONF

- Définit comme en DSL (Domain Specific Language) embarqué dans le langage Haskell
- Permet de bénéficier de l'infrastructure fournie par le langage sous-jacent : typage fort, compilation, évaluation paresseuse…
- L'interpréteur (GHCi) permet d'exécuter les tests en mode interactif
- Expose l'API "métier" de la plate-forme permettant d'interagir avec ses différents composants et une API de *Nemesis* pour injecter des erreurs

Objectif:

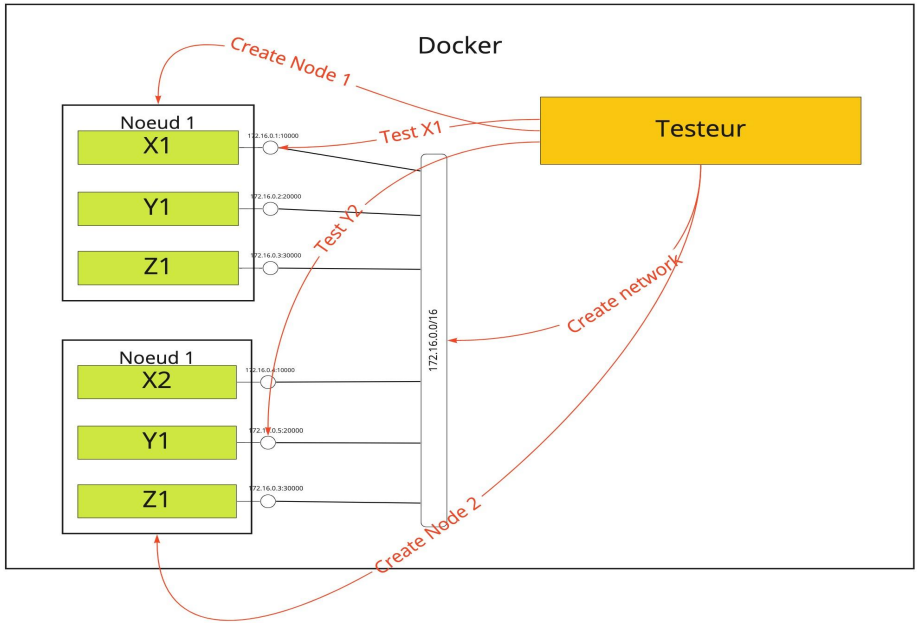
- Construire progressivement un langage représentant l'*usage opérationnel* de la plate-forme

Injection d'erreurs

- Utilisation de blockade, un utilitaire écrit en python
- Fonctionne en utilisant iptables pour manipuler la pile réseau de chaque conteneur
- Écoute le flux d'événements du serveur docker pour connaître la topologie du réseau
- Fournit plusieurs types de commandes
  - `slow` : ajoute un retard aléatoire sur chaque paquet
  - `flaky` : perd des paquets de manière aléatoire
  - `partition` : partitionne le réseau en sous réseaux déconnectés

Exécution des tests

- Les tests sont compilés et packagés dans une image docker
- Ils sont exécutés à partir d'un conteneur docker qui contrôle les différentes phases de l'exécution : déploiement du "réseau" (*Setup*), exécution des "scripts", collecte du résultat, destruction des conteneurs déployés (*Tear-down*)
- Ils peuvent intégrés dans un processus d'*Intégration continue*



Conclusion

- *docker* offre un environnement idéal pour tester le comportement de systèmes répartis à moindre frais
- Il permet de "packager" de manière portable tout les composants d'un système et de déployer des topologies arbitraires y compris sur du matériel léger (laptop de dev)
- La complexité gagne à être abstraite dans un *DSL de test* offrant une interface de haut-niveau pour écrire des tests
- Utiliser un langage puissant (Haskell) pour développer et maintenir ce DSL permet de garantir sa sûreté et sa maintenabilité

Questions?