Architecture DevOps

- Automatisez vos déploiements et votre workflow

Bonjour 👋

- Présentations
- Comment travaillerons-nous ?
- Les bonnes pratiques. Lesquelles connaissez-vous dans le métier ? Celles que je connais
- État des lieux : que connaissez-vous ?
- Que connaissez-vous du devops ?

Bonnes pratiques

- Il n'y a **pas de mauvaise question** (PAS DU TOUT)
- **Les moteurs de recherches** sont nos amis
- **L'anglais** 🇬🇧
- **ChatGPT** ? 🤔 C'est bien pour apprendre, mais pas pour produire (interdit) !
- **La pratique** est cruciale !
- **Collaborer** donne des ailes
- **Faire de la veille technologique**, tout change tout le temps (et YouTube c'est sympa 📺)

Dès ce premier cours nous allons mettre en pratique ces réflexes d'**auto-apprentissage** (nécessaires tout au long de la carrière).

Pendant ce cours (et les suivants)

- **“Coupez”-moi la parole** 👌 : le but n’est pas de vous transmettre un monologue, on discute et on construit ensemble !
  - Aussi, tout sera inconnu, alors toute question est la bienvenue.
  - Enfin, il est possible que j’aille vite par habitude. On peut ralentir ou répéter.
- **Prenez des notes** : il y a beaucoup d’informations à digérer, et il faut pouvoir reproduire ce qu’on fait en autonomie.
- **Faites des recherches** : c’est un bon réflexe d’ouvrir de nouveaux onglets pour les **mots clés inconnus** : enregistrer les ressources, articles, et autres à regarder plus tard en dehors du cours.

Objectifs

Le… DevOps ? Qu’est-ce que ça veut dire ?

Que va-t-on faire ?

Objectifs du cours

Avant d’aller dans le détail, voici la vue d’ensemble des notions que l’on va apprendre :

- Docker
- CI/CD sur GitHub
- Tests Unitaires

Nos objectifs sont donc de connaître l'architecture DevOps, et de savoir superviser une architecture DevOps.

Comment définir et configurer une architecture devops ?

Faisons une intro en douceur

Définir et configurer une architecture DevOps

Une architecture DevOps est un ensemble d'outils et de pratiques qui permettent aux équipes de développement et d'opérations de collaborer efficacement.

Avec **Docker**, on peut créer des conteneurs pour nos applications, ce qui facilite leur déploiement et leur gestion. Nous creuserons cela un peu plus tard.

Automatisation du cycle de vie d'une application

Grâce au **CI** / **CD** (Intégration Continue / Déploiement Continu) sur GitHub, vous pouvez automatiser le processus de test et de déploiement de votre application.

Cela signifie que **chaque fois que vous apportez des modifications, des tests unitaires sont exécutés automatiquement** pour s'assurer que tout fonctionne avant de déployer les changements !

Outils & Techniques de DevOps

Quels sont les outils et Techniques pour le Développement et le Déploiement ?

- **Docker** : Utilisé pour créer des environnements isolés pour vos applications.
- **Jenkins** : Un outil d'automatisation qui peut être utilisé pour gérer des pipelines CI/CD.
- **Kubernetes** : Gère le déploiement et l'évolutivité des conteneurs Docker.
- **Travis CI** : Un autre outil de CI/CD qui s'intègre bien avec GitHub.

Supervision et Performance des Applications

Il est essentiel de surveiller nos applications et notre infrastructure pour garantir leur disponibilité et leurs performances.

Des **outils de monitoring** peuvent nous alerter en cas de problème.

Docker

Facilitez la portabilité de vos environnements de développement

Intro à Docker

On ne le nomme plus : **Fireship** ! Si vous voulez démystifier un sujet rapidement allez sur sa chaîne. Pour le cours du jour : Docker in 100 Seconds

De nos jours, le développement d'applications ne se limite pas à l'écriture de code. La multiplicité des langages, des cadres, des architectures et des interfaces discontinues entre les outils pour chaque étape du cycle de vie crée une **énorme complexité**. 👉 Docker simplifie et accélère votre flux de travail, tout en donnant aux développeurs la liberté d'innover en choisissant leurs outils, leurs piles d'applications et leurs environnements de déploiement pour chaque projet.

Les conteneurs sont une unité logicielle standardisée qui permet aux développeurs d'isoler leur application de son environnement, ce qui résout le problème du "ça marche sur ma machine".

Pourquoi docker ? -> Why Docker | Docker
Très bonne ressource à suivre : Docker Simplified: A Hands-On Guide for Absolute Beginners

Pourquoi utiliser Docker en tant que développeur?

Avantages de Docker :

- Rapidité: Démarrage et arrêt rapides de l'application.

- Multiplateforme: Fonctionne sur n'importe quel système.

- Construction et destruction rapides des conteneurs.

- Configuration facile: Évite les problèmes d'installation manuelle des dépendances.

- Environnements isolés: Maintient la propreté de l'espace de travail.

- Déploiement facilité: Simplifie la mise en ligne du projet sur le serveur.

Pourquoi utiliser Docker en tant que développeur?

Inconvénients de Docker

- Complexité réseau : Gestion parfois délicate des connexions entre conteneurs.

- Taille des images : Les images peuvent être volumineuses, occupant de l'espace.

- Sécurité : Risques potentiels liés à la sécurité, nécessitant une configuration appropriée.

- Pas idéal pour tous les types d'applications : Certains scénarios peuvent nécessiter des solutions différentes.

- Compatibilité du noyau : Des problèmes peuvent survenir si le noyau de l'hôte n'est pas compatible avec le conteneur.

- Gestion des volumes : Peut être complexe pour les applications nécessitant une gestion intensive des fichiers.

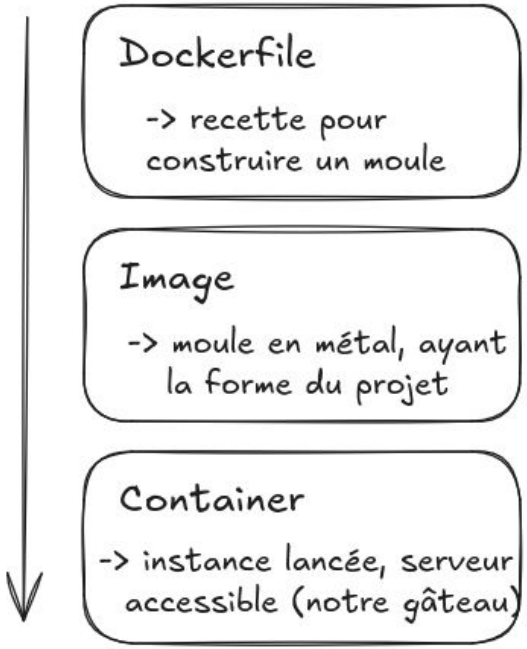
Docker : Les trois termes les plus importants !

Dockerfile :
Script qui décrit les étapes de construction d'une image Docker.

Image :
Snapshot (ou instantané de stockage, permet de réaliser une copie de données stockées sur un système de stockage, ou une copie des modifications apportées à ces données) immuable et exécutable d'une application et de son environnement. (template for running containers)

Container :
Instance en cours d'exécution d'une image Docker, isolée et portable. (running process)

Visuellement : Dockerfile, Image, Container



Dockerfile

Recette sur "comment construire un atelier de sculptures"

Recette sur "comment construire un atelier de peintures"

script Dockerfile…

Image

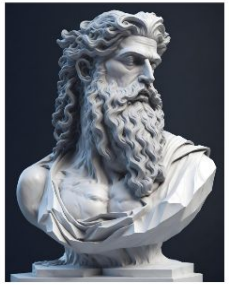
=> atelier permettant de produire des statues

=> atelier permettant de produire des peintures

docker build …

Container

L'instance finale : la statue



L'instance finale : la peinture

docker run …

Mais j’ai pas compris, c’est quoi une image ?

C’est comme un modèle ou un "plan" qui contient tout ce qu’il faut pour faire fonctionner une application : le code, les bibliothèques, les dépendances et les configurations nécessaires

Une image est en lecture seule et ne change pas : elle sert simplement de base.

À partir de cette image, Docker peut créer un conteneur (une instance en cours d’exécution de l’image), c’est-à-dire un environnement isolé où l’application tourne vraiment

Cela permet de garantir que l’application marchera toujours pareil, peu importe où on la lance

On crée souvent une image à partir d’un fichier spécial appelé Dockerfile, qui décrit étape par étape comment construire l’image

But de Docker

Le but c'est de pouvoir reproduire les environnements : faire un Dockerfile personnalisé, pour que n'importe qui le réutilise pour build son environnement grâce à cet immuable snapshot. On met cette image en ligne pour que n'importe qui puisse la récupérer et générer son propre container

Puis swarm et kubernetees arrivent en jeu.

Kubernetes simplifie la gestion des applications, améliore la disponibilité et facilite la mise à l'échelle, le tout grâce à une automatisation intelligente.

Imaginez **Kubernetes** comme un chef d'orchestre pour vos applications. Vous avez plusieurs applications emballées dans des conteneurs, et Kubernetes s'occupe de les déployer, de les mettre à l'échelle (ajuster automatiquement le nombre de copies en fonction de la demande), et de s'assurer qu'elles fonctionnent correctement. Il gère également la répartition du trafic entre les différentes parties de votre application, garantissant une disponibilité continue.

Docker Swarm est une autre technologie liée à la gestion de conteneurs, mais elle se concentre spécifiquement sur l'orchestration de conteneurs Docker. Contrairement à Kubernetes, qui est plus complexe et offre une gamme plus étendue de fonctionnalités, Docker Swarm est une solution d'orchestration plus légère et plus simple, développée directement par Docker.

Installer docker

Aller sur le site officiel de Docker : https://www.docker.com/get-started Télécharger selon votre OS.
Suivre les instructions d'installations spécifique à l'OS.

Vérifier que ça a fonctionné :

```
docker --version

# télécharger une image Docker de test et exécuter un conteneur :
docker run hello-world

# commande super importante à connaître !
docker ps # liste tous les containers actifs sur mon système
```

“Hello world ne marche pas chez moi”

Si certaines personnes ont un message contenant `401 unauthorized`, c’est que la requête fonctionne, mais que l’accès est non autorisé. Il faut se connecter :

Sur terminal (Git Bash ou Powershell) :

```
docker login -u <mon_username>
```

Installez Docker sur VSCode

Cela permet d'**avoir un support langage**, pendant qu'on rédige nos fichiers docker. Pratique pour avoir les bonnes couleurs et de l'aide.

Nous pouvons également faire un lien vers des **remote registeries**.

Créer un projet test en NodeJS

Pour s'échauffer, on va faire un projet backend quasi-vide, mais fonctionnel, en Node.js. Puis on va le conteneuriser.

Il permet de voir qu'on lance un projet fonctionnel sur une machine et que grâce à docker il est reproductible ailleurs à l'identique.

=> Suivez les étapes indiquées dans mon repository.

Bonus

Éditer notre projet dans le terminal

Pour les personnes qui aiment bien creuser le terminal

Voilà quelques commandes utiles

Comment naviguer dans le système de fichiers ?

- `pwd` : savoir où on se trouve
- `cd chemin_du_dossier` : aller dans un dossier (et `cd ..` pour revenir au dossier parent)
- `ls`, ou `ls -l`, ou `ls -la` : lister les éléments

Commandes de manipulation de fichiers :

- `mkdir nom_de_dossier`: créer un dossier
- `touch nom_du_fichier.txt` : créer un fichier
- `cat nom_du_fichier` : afficher le contenu du fichier

Commandes très pratiques :

- ajouter `--help` à la suite de la commande sur laquelle on se pose des questions
- écrire `man le_nom_de_ma_commande` pour en savoir plus sur la commande (`man` veut dire manual)

On peut éditer nos fichiers directement dans le terminal ?

-> Oui, carrément : des outils d'éditions de texte comme nano et vim permettent de manipuler nos fichiers. Je conseille **vim**, que j'utilise beaucoup !

Le Dockerfile

Définition. C’est quoi un Dockerfile ? 👉 le Dockerfile est un script texte qui contient une série d'instructions permettant de construire une image Docker.

Il spécifie

- l'environnement de travail,
- les dépendances,
- les étapes nécessaires à la configuration d'une application à l'intérieur d'un conteneur Docker.
- ex: sélection d'une image de base, définition du répertoire de travail, copie des fichiers du projet, l'installation des dépendances, etc.

Voilà un Bon résumé écrit de fireship

Mise en pratique
À la racine du projet Node.js, dans un fichier nommé Dockerfile, écrire le code suivant.

Les images

Définition : l'image est un **package léger et autonome qui contient tout le nécessaire pour exécuter une application**, y compris :

- le code,
- les bibliothèques,
- les dépendances,
- les variables d'environnement,
- les fichiers de configuration

Les images sont construites à partir d'un ensemble d'instructions définies dans un Dockerfile. Elles sont immuables, ce qui signifie qu'une fois créées, elles ne sont pas modifiées. Les conteneurs Docker sont ensuite créés à partir de ces images et représentent l'instance en cours d'exécution de l'application.

Comment créer une image ?

```
docker build -t mon_pseudo/projet-docker:1.0 .
```

- l'usage de `-t` permet de donner un nametag à l'image
- un bon réflexe de donner mon pseudo suivi du nom de l'app
- on remarque qu'il suit les étapes de mon Dockerfile

Obtenir l’image du docker

Pour obtenir l’image du docker qu’on vient de créer, faire la commande `docker images`

On voit une liste d’images, dont celle qu’on vient tout juste de créer

```
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
onyj/projet-docker            1.0                 d4ef76a0eac7        7 minutes ago       918MB
```

Maintenant il suffit de copier le code dans la colonne “IMAGE ID”.

Si on fait `docker run id_de_mon_image_docker` (pour moi, d4ef76a0eac7) on voit qu'il lance `localhost:8080` . Le problème : quand on le lance depuis le navigateur, ce n’est pas accessible !

On expose le 8080 depuis le docker mais c'est pas accessible depuis le monde extérieur.

Dockerfile FROM

Donc finalement, ce qui est en face de FROM dans notre Dockerfile c'est une image de base (base image) à partir de laquelle on pourra faire toutes les opérations précisées dans la suite de notre Dockerfile.

Ok mais comment trouver la bonne image de base ? Pour ça, la plupart du temps, on part de notre environnement de projet (le langage, le framework), puis on le trouve sur Docker Hub.

Les plus utilisées sont ubuntu, python, node, hello-world… on peut les trouver ici.

Lancer le serveur depuis docker

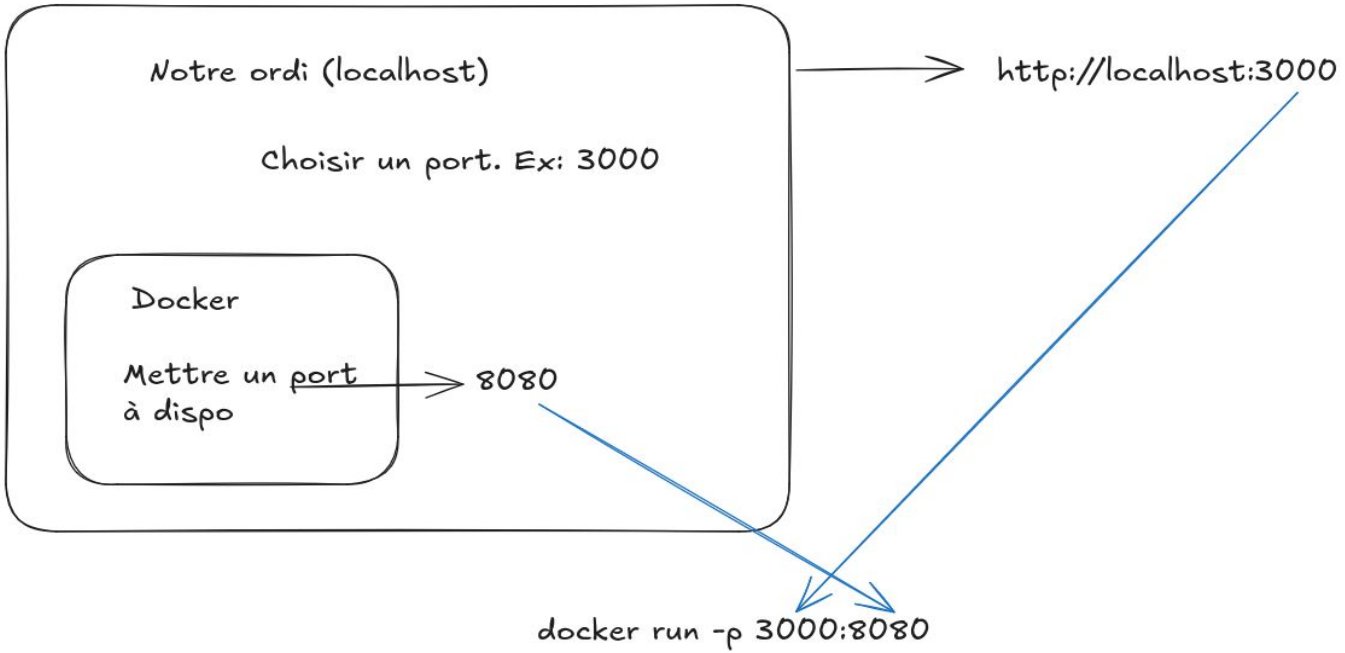
Donc la bonne méthode, c'est de lancer :

```
docker run -p 5000:8080 d4ef76a0eac7
# attention à mettre votre id d'image à vous à la fin, pas d4ef... !
```

- 5000 : c'est le port sur notre machine
- 8080 : c'est le port du docker
- le code juste après : l'id de notre image, trouvée grâce à la commande `docker images` lancée dans le terminal
- maintenant ouvrir localhost:5000 sur navigateur, et ça marche !

Attends, je dois exposer 2 ports différents ?

Docker nous permet de lancer un environnement virtuel, isolé au sein de notre ordi.
Pour qu'il soit accessible au port local de notre ordi, on va exposer ce fameux environnement virtuel sur un port pour docker



Comment désactiver les ports ?

Mais si on ferme la page ça continue de tourner, il faut donc aller manuellement dans l'interface de docker visuelle (l’application) pour désactiver les ports.

Les Volumes Docker

Comment rendre les données persistantes et partageables, indépendamment de la vie des conteneurs ?

Un volume est utile pour :

- Garder tout ce qui doit survivre à un redémarrage ou à la suppression d'un conteneur
- Partager des fichiers entre plusieurs conteneurs

À quoi sert un volume Docker ?

À quoi sert un volume Docker ?

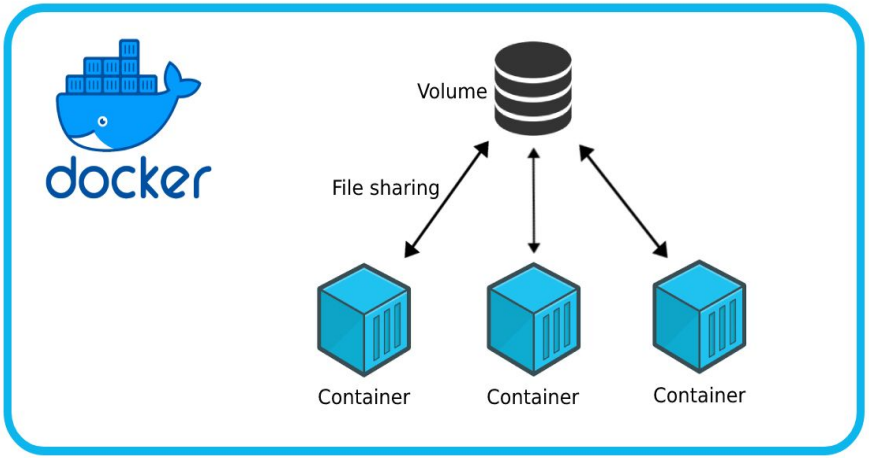
- **Persistance des données**
  Les conteneurs Docker sont éphémères : si on supprime ou recrée un conteneur, tout ce qui est stocké à l'intérieur (dans le système de fichiers du conteneur) est perdu.
  👉 Un **volume permet de stocker des données de façon persistante**, indépendamment du cycle de vie du conteneur.
  Par exemple, si ton application écrit des fichiers dans /stuff, ces fichiers resteront disponibles même si tu détruis et recrées le conteneur

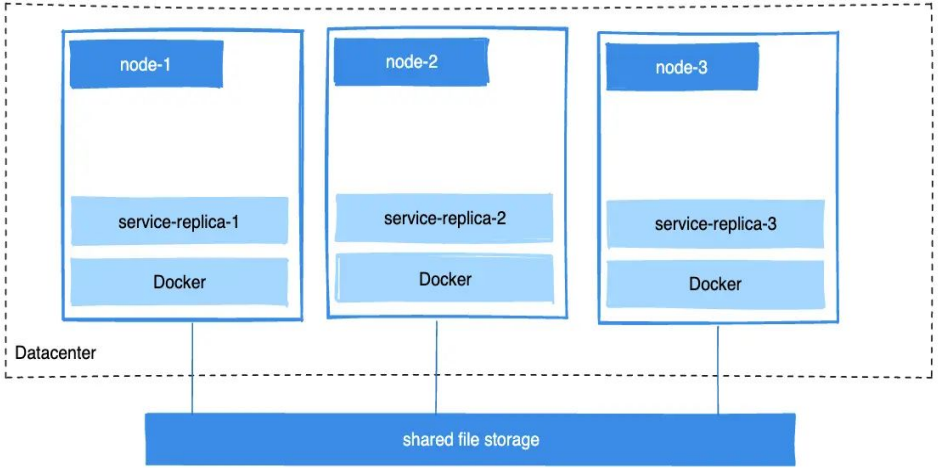
- **Partage de données**
  Un volume peut être monté dans plusieurs conteneurs en même temps. Cela permet de partager des fichiers ou des données entre plusieurs services ou applications qui tournent dans des conteneurs différents

- **Séparation des données et de l'application**
  Le volume est stocké sur l'hôte Docker, dans un emplacement géré par Docker (généralement sous /var/lib/docker/volumes).
  Cela sépare clairement les données de l'application, ce qui facilite la sauvegarde, la migration ou la gestion des données sans toucher à l'application elle-même

- **Sécurité et portabilité**
  Les volumes sont isolés du système de fichiers principal de l'hôte, ce qui réduit les risques de conflits ou de suppression accidentelle. Ils sont aussi plus portables que les montages de type "bind" (qui lient un dossier précis de l'hôte)

Voilà ce qu'il se passe, visuellement





Partageons les données grâce aux volumes

Partager les données entre plusieurs containers (persist files).

1. Créons un volume Docker nommé shared-stuff :
   docker volume create shared-stuff

2. Lorsque qu'on lance un conteneur avec *docker run*, utiliser l'option *--mount* pour lier le volume créé au conteneur. Par exemple :
   docker run --mount source=shared-stuff, target=/stuff mon_conteneur mon_port

Plus de détails ici

Ok, mais que faire de mon volume ?

- **Stocker des données d'application**
  Utiliser le volume pour y stocker tout ce que notre application doit conserver : bases de données, fichiers de configuration, fichiers utilisateurs, logs, etc.

- **Sauvegarder ou restaurer**
  On peut sauvegarder le contenu du volume (par exemple pour des backups réguliers) ou le restaurer sur un autre hôte ou environnement.

- **Partager entre plusieurs conteneurs**
  Si on a besoin que plusieurs conteneurs accèdent aux mêmes données, on peut monter le même volume dans chacun d'eux.

- **Nettoyer ou supprimer**
  Si on n'as plus besoin du volume, on peut le supprimer avec *docker volume rm shared-stuff* (après avoir arrêté et supprimé les conteneurs qui l'utilisent).

Manipuler le volume

Concrètement, voilà ce qu'on peut faire sur notre volume (dans un terminal) :

- Voir le contenu d'un volume :
  docker exec -it mon_conteneur sh
  ou bash si disponible :
  docker exec -it mon_conteneur bash
  afficher le fameux stuff : *ls /stuff*
- Copier un fichier depuis le volume vers notre machine hôte
  docker cp mon_conteneur:/stuff/nom_du_fichier ./local_destination/
- Copier un fichier de notre machine hôte vers le volume
  docker cp ./mon_fichier mon_conteneur:/stuff
- Lister tous les volumes Docker
  docker volume ls
- Inspecter un volume pour voir où il est stocké sur l'hôte
  docker volume inspect shared-stuff
- Supprimer le volume (après avoir supprimé tous les conteneurs qui l'utilisent)
  docker rm mon_conteneur
  docker volume rm shared-stuff
- Sauvegarder le contenu du volume

```
docker run --rm \
  -v shared-stuff:/stuff \
  -v $(pwd):/backup \
  alpine \
  tar czf /backup/backup-stuff.tar.gz -C /stuff .
```

- Restaurer une sauvegarde dans le volume

```
docker run --rm \
  -v shared-stuff:/stuff \
  -v $(pwd):/backup \
  alpine \
  tar xzf /backup/backup-stuff.tar.gz -C /stuff
```

Appliquer le DevOps sur un vrai projet 🔥

Plongeons dans l'application concrète du DevOps sur un projet réel

Projet

Ok on a vu comment fonctionne Docker avec un projet Node.js.

Pour moins s'embrouiller la tête (et ensuite aller plus loin) on va se refaire un projet, qui nous servira de modèle toute notre vie de dev (lorsqu'on voudra se remettre sur du Docker)

On va créer un jeu nommé ClickFast⏩

- En HTML CSS & JavaScript
- On conteneurise le projet dans **Docker**
- On implémente du **CI/CD** pour la mise en ligne
- On ajoute des **Tests** qui vont vérifier notre code à notre place

Commencez ClickFast

Commençons par créer la base du projet : une page fonctionnelle, permettant de cliquer et afficher le score de clics.

Suivez donc l'***Exercice I*** du README.

Voilà ce que vous y ferez en gros :

- Ajouter un bouton sur votre site
- Créer un script.js, qui écoute votre bouton pour détecter les clics.
- Faire qu'à chaque clic, votre score augmente de 1 (et s'affiche dans le HTML)
- Ajouter un Chrono permettant de jouer max 5 secondes

Implémentation de Docker

En suivant l'Exercice II du Readme, on peut implémenter Docker, comme on l'avait fait précédemment.

Docker compose

“*C’est bien drôle de passer ma vie à taper des commandes dans le terminal, mais il n’y a pas plus rapide ?*”

Eh bien si, au lieu de retenir plusieurs commandes complexes par projet, on a un moyen de standardiser les règles dans un fichier nommé docker-compose.yml.

Voici comment créer et configurer docker-compose.yml.
(Et voici une autre ressource similaire)

Pour un récap complet sur les notions Docker vues jusqu’à présent, voilà un tutoriel complet.

Exemple basique de la structure d'un docker-compose.yml

```yaml
version: 'X'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
  redis:
    image: redis
```

Source : https://www.educative.io/blog/docker-compose-tutorial

Comment utiliser mon docker-compose ?

L’avantage ultime c’est qu’on n’a plus que 3 commandes principales à retenir :

- docker-compose build
  Est l’équivalent de la (longue) commande build que l’on faisait dans le terminal

- docker-compose up
  Équivalent de la commande *docker run…* que l’on faisait

- docker-compose down
  Permet de stopper le container qui était lancé avec *docker-compose up*

Mise en ligne du projet & CI/CD

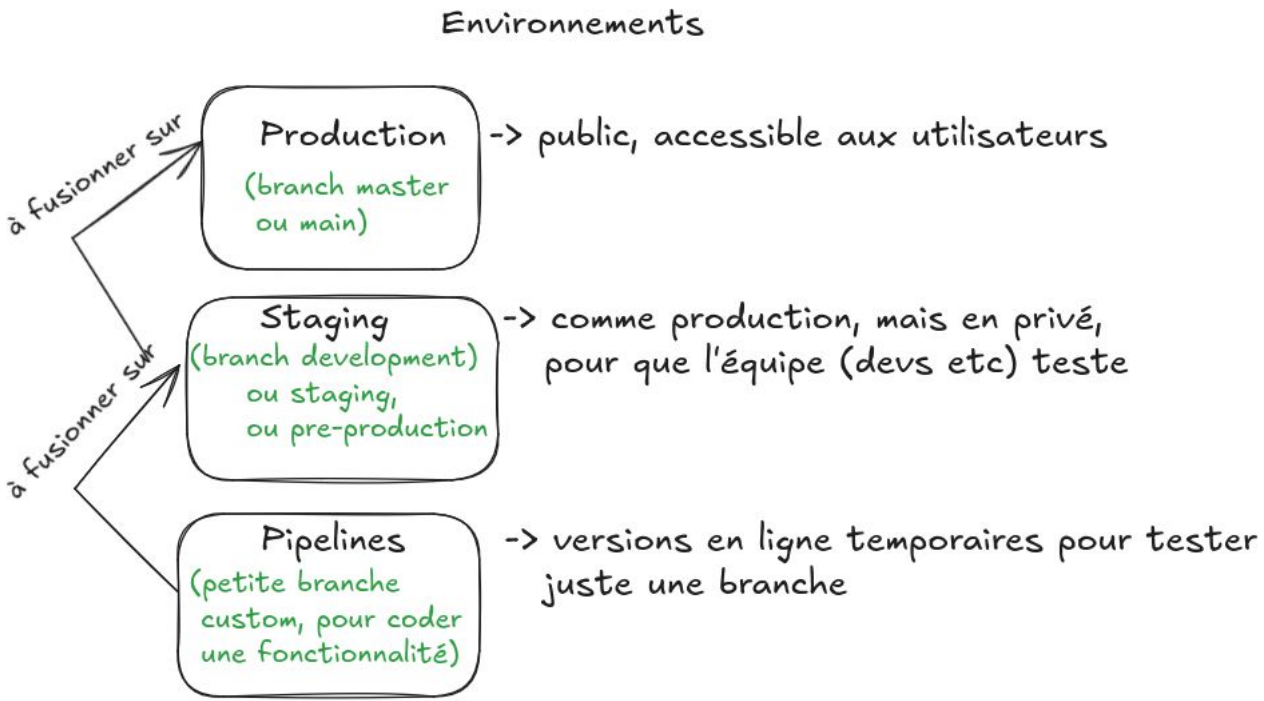
Exercice III, on peut enfin automatiser certaines choses !

Automatiser les pipelines CI / CD

Les outils d'automatisation pour gérer des pipelines CI/CD

Les environnements

Avant toute chose, petit récap sur les environnements, lorsqu'on travaille sur un ayant plusieurs embouts accessibles en ligne :



Les branches Git : Bonnes pratiques

Forcément, cela nous mène à se faire un petit récap sur les branches

- `master` ou `main` : est une branch à garder protégée : tout travail ajouté à celle-ci doit être totalement propre et fonctionnel.
- `staging` : idéalement on a une branche staging (ou `pre-production`, ou `development`), pour y joindre le travail du / des développeurs de l'équipe. On essaie de voir si toutes les nouvelles fonctionnalités sur les nouvelles branches fonctionnent toujours, une fois fusionnées
- `autres branches` : il est super important de réaliser le travail sur des branches à part, pour ne pas polluer ce qui marche déjà. Une fois le travail satisfaisant, on push la branch, pour faire une pull request. Elle sera ensuite reviewed (revue) par le/la/les devs afin de garantir une vérification supplémentaire.

Tests Unitaires
(et tests d'intégration)

Du code qui… vérifie mon code à ma place ?

Les Tests (Tests Unitaires, Tests d'intégration)

Franchement bravo d'être arrivé(e)s jusque-là !

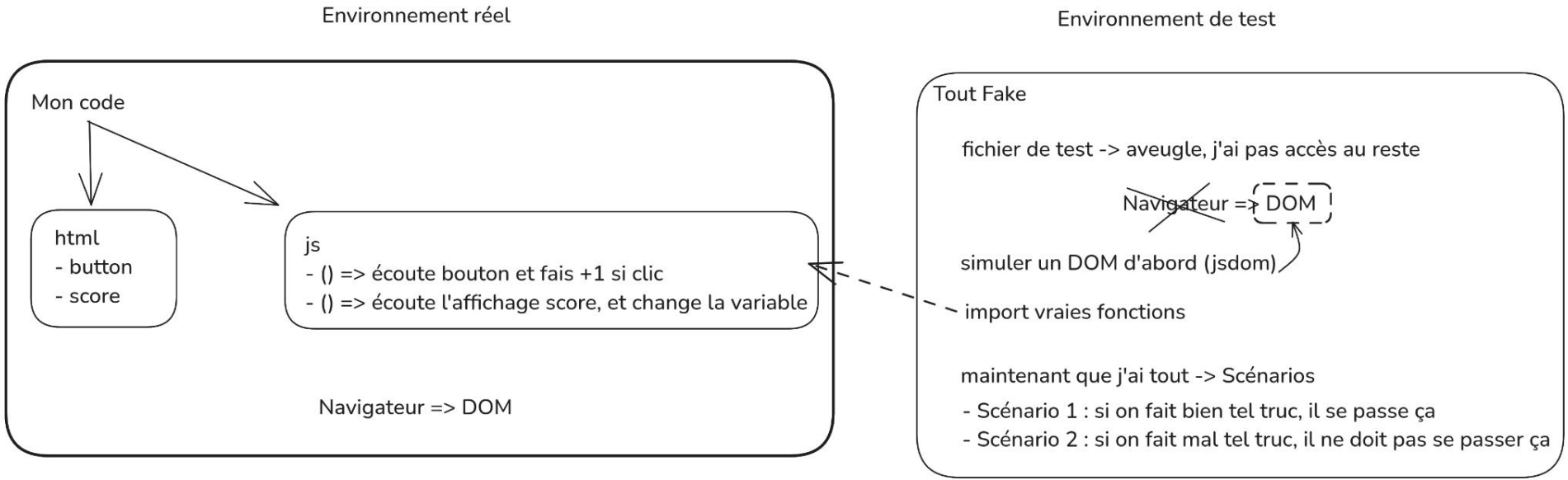
On a atteint un stade où on peut être aussi flemmard (et intelligent) que les devs qui écrivent un code qui vérifie lui-même notre propre code.

- Qu'est-ce qu'un test unitaire ?
- Qu'est-ce qu'un test d'intégration ?

Comment fonctionne un test ?

Depuis le début, nous travaillons dans un environnement “réel”. C’est notre code + son exécution dans le navigateur => cela crée un nouveau DOM.

En revanche, l’environnement de test lui est totalement vide : on va donc reconstruire les éléments qui nous intéressent, pour simuler divers scénarios de “ce code devrait se comporter/réagir ainsi”, “ce code devrait ne pas faire ci/ça”



Ready to test 😎

Rendez-vous dans l’Exercice IV du Readme pour créer vos propres tests.

Bonus : Requêtes API

On va continuer de pimper le jeu

Big Boss - Création de la ScoreBoard

Maintenant qu'on a une bonne grosse base et qu'on a vu tous les concepts qu'on souhaitait voir, on va pouvoir s'amuser un peu plus : on va créer un système de scoreboard, permettant à tout le monde de soumettre son score et voir celui des autres.

Pour l’Exercice V, c’est ici que ça se passe.

Bonus Plus PLus PLUS

On est libres d’ajouter tout ce qu’on veut à notre projet !

- Clean Code : Appliquer les SOLID principles
  (Faire kiffer les recruteurs en entretien technique, en montrant du code exemplaire)
  - est-ce que votre code est séparé dans plusieurs fichiers ? (ranger dans des dossiers avec des noms cohérents)
  - les fonctions font plein de choses en même temps, ou est-ce que votre projet applique bien le Single Responsibility Principle (SRP) ?
  - votre code réutilise plusieurs fois des morceaux de code très similaires, ou bien il est DRY (Don’t Repeat Yourself)
- Front-end
  - Ajout d’animations stylées
  - Ajout de fonds de couleurs, styles de boutons, polices d’écritures, emojis…
- Fonctionnalités
  - Proposer divers choix de difficultés
  - Mini-jeux : faire apparaître des boutons cliquables à des endroits aléatoires au lieu d’un bouton fixe

Autres ressources

Jenkins

Qu’est-ce que Jenkins ?

D’accord mais c’est quoi une pipeline ?

Installons-le puis utilisons-le dans notre projet.

Travis CI

Leur site est excellent et très clair, il nous accompagne directement dans la compréhension et la prise en main de l’outil.

Kubernetees

Gérer le déploiement et l'évolutivité des conteneurs Docker.

Kubernetees, l’essentiel

Voir la doc officielle : https://kubernetes.io/

Notre ressource complète à suivre

Packet Tracer

Utilisons un simulateur réseau interactif pour créer, configurer et visualiser des réseaux virtuels

- afin de comprendre leur fonctionnement
- sans matériel réel

Packet Tracer, dans un cours Docker ?

On connaît Docker, c’est super pratique pour virtualiser des applications et leurs réseaux. Mais comprendre ce qui se passe vraiment derrière, côté réseau, c’est encore mieux

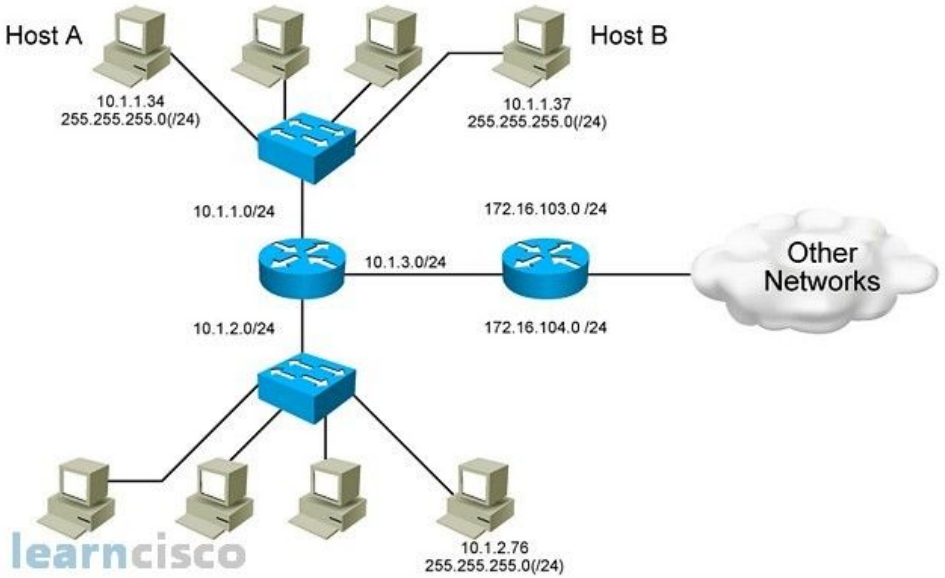
C’est là que Packet Tracer entre en jeu : il va nous permettre de **visualiser**, **expérimenter** et **comprendre** les réseaux qui relient nos conteneurs.

- Docker : On lance des applis dans des conteneurs, on définit des réseaux virtuels, mais tout reste “assez abstrait”.
- Packet Tracer : On simule des réseaux comme si on avait du vrai matériel (PC, switch, routeur…), on voit passer les paquets, on comprend comment tout circule

Premièrement, C'est quoi un réseau IP ?

Un réseau IP, c’est un ensemble d’appareils (PC, serveurs, etc.) qui communiquent entre eux grâce à des adresses uniques (les adresses IP).

C’est la base de tout Internet : chaque appareil a une adresse, et les données voyagent de l’un à l’autre grâce à des routeurs et des switches.



Tu ne m'as toujours pas dit ce qu'est Packet Tracer

Oui oui on y vient, après 3 slides

Packet Tracer, c’est un simulateur de réseaux développé par Cisco.

On construit des réseaux virtuels, on relie des équipements (PC, switchs, routeurs), on configure tout ça et on voit comment les données circulent.

On peut même observer les paquets, tester des pings, voir si la connexion marche ou non

- **Switch**, c’est un boîtier qui relie plusieurs appareils sur un même réseau local et envoie les données uniquement à l’appareil concerné
- **Routeur**, c’est l’appareil qui fait circuler les données entre différents réseaux et connecte le réseau local à Internet

Mais c’est uniquement virtuel ?

Packet Tracer ça permet de simuler des environnements virtuels mais est-ce qu’on peut aussi gérer des réseaux réels ?

Eh non ! Ça ne permet que de simuler des réseaux, mais pas de manipuler des réseaux réels. C’est uniquement un simulateur, pour créer et configurer des réseaux virtuels, s'entraîner, et tester des scénarios, sans matériel physique. En revanche on y reproduit le comportement de vrais équipements, et on peut bien voir comment les paquets circulent.

Pour manipuler de vrais réseaux ou interagir avec du matériel réel, on peut utiliser d’autres outils comme GNS3, ou travailler sur des équipements physiques.

Une simulation réseau, à quoi ça sert ?

- On peut tester des configurations sans rien casser.
- On comprend comment les équipements (routeurs, switches) interagissent.
- On visualise les échanges de données (paquets, trames).
- On compare avec ce qu’on fait dans Docker (même logique, mais plus visuel)

Configurer des réseaux virtuels

Dans Packet Tracer, on ajoute des équipements, on les relie avec des câbles, on attribue des adresses IP, on configure les services (DHCP, DNS, etc.).

On peut aussi observer les tables de routage, voir comment les paquets sont transmis, et tester différentes topologies (en étoile, en bus, etc.)

Et côté Docker ?

Docker crée aussi des réseaux virtuels (bridge, overlay…).

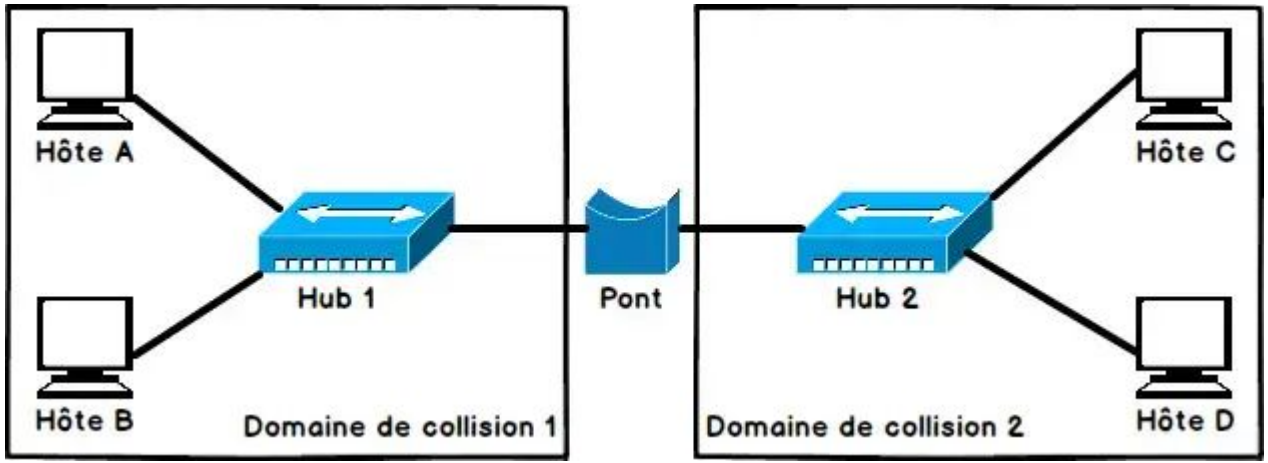
Par exemple, avec le réseau bridge par défaut, tous les conteneurs sont sur le même sous-réseau et peuvent communiquer entre eux.

On peut lister, créer et inspecter ces réseaux avec des commandes comme *docker network ls* ou *docker network inspect*

Qu’est qu’un bridge ?

Un pont est un type de périphérique réseau qui assure l’interconnexion avec d’autres réseaux utilisant le même protocole.

En savoir plus



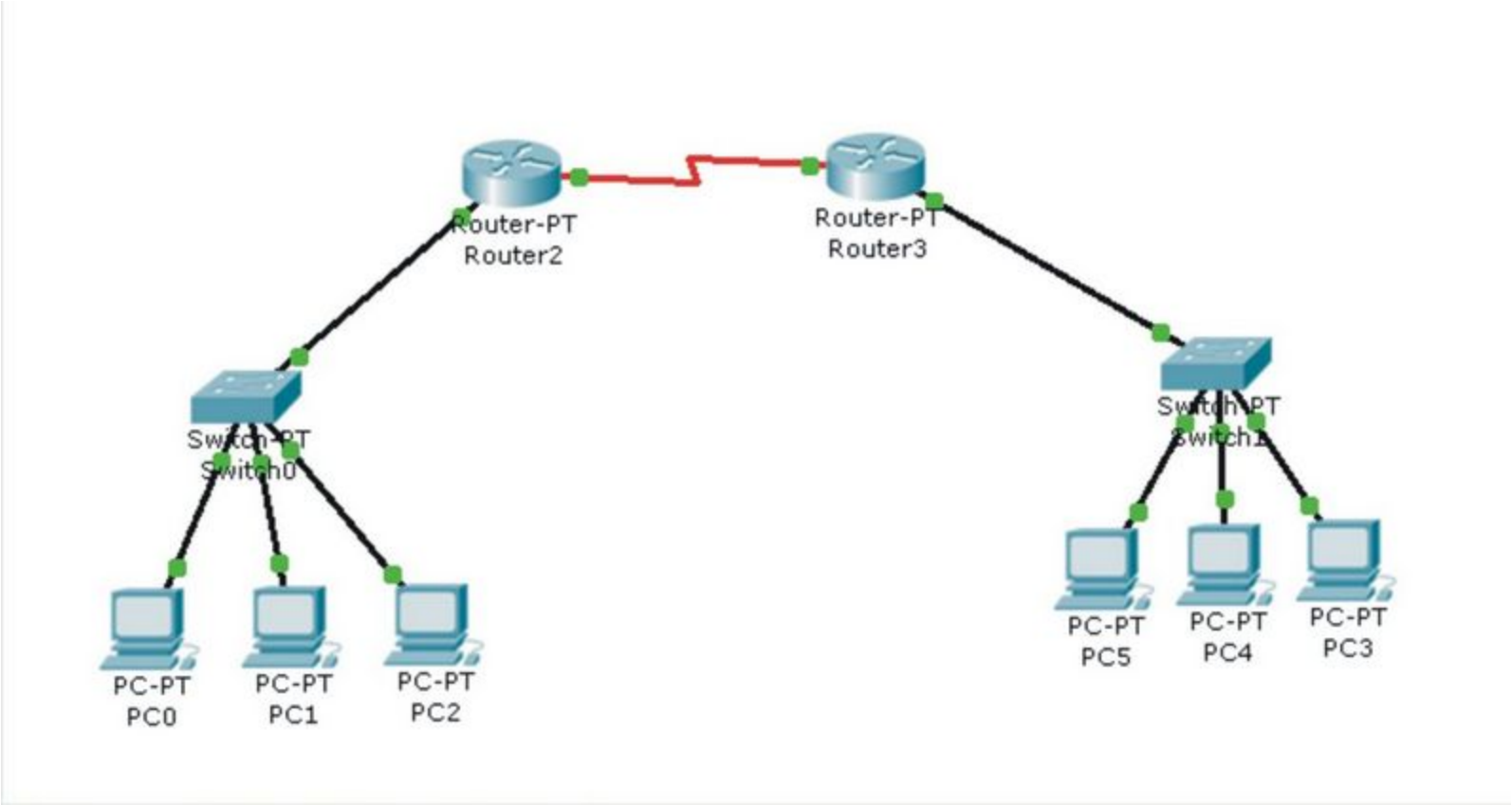
Exemples de schémas réseaux

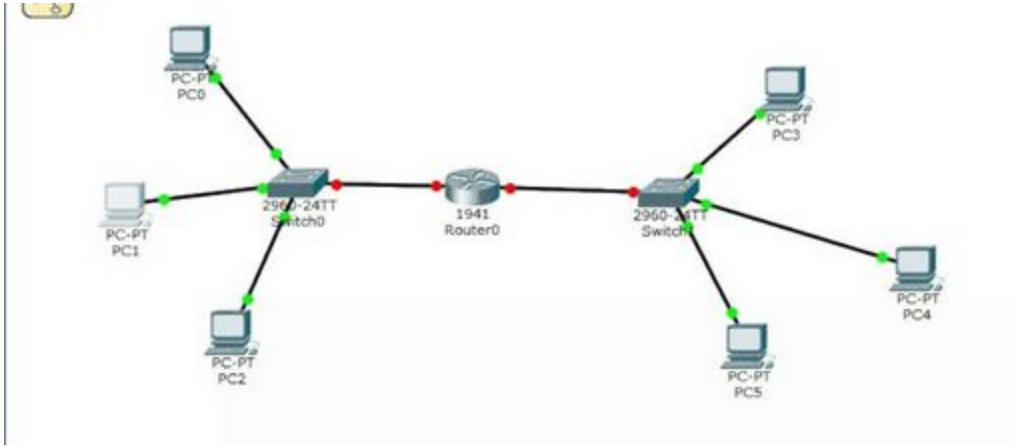
Pour illustrer, voici quelques exemples de schémas réseau simples :

- Deux PC reliés à un switch (équivalent à deux conteneurs sur un bridge Docker)
- Un PC, un switch, un routeur, et un serveur (pour simuler une architecture plus complète)

On peut trouver d’autres exemples :

- Directement dans Packet Tracer (exemples intégrés)
- En cherchant “exemples topologie réseau Packet Tracer” sur Google Images


Router-PT
Router2
Router-PT
Router3
Switch-PT
Switch0
Switch-PT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC5
PC-PT
PC4
PC-PT
PC3


PC-PT
PC0
PC-PT
PC3
PC-PT
PC1
1941
Router0
PC-PT
PC4
PC-PT
PC2
PC-PT
PC5

Installation

Le plus simple et conventionnel est d'installer packet tracer directement sur nos ordinateurs.

Il suffit d'un compte Cisco Skills for All (gratuit) pour télécharger et utiliser l'outil.

Faire un compte avant d'essayer de télécharger.

Installation sans inscription peut-être possible :

- Sur Windows, télécharger : https://archive.org/download/cpt822/CiscoPacketTracer822_64bit_setup_signed.exe
- Sur Windows et Linux : https://www.netacad.com/resources/lab-downloads?courseLang=en-US

Step By Step

Pour notre cours, nous suivrons ce guide de My It Knowledge

Autres Ressources :

- Le cours d'OpenClassrooms
- Tutoriel Youtube

Ressources Bonus

Ressources

Windows, Mac, Linux

- Installation du terminal Ubuntu sur Windows
- Retrouver ses dossiers de base Windows (commande: `cd /mnt/c/Users`)
- Je ne peux pas lancer la commande `code .` sur le terminal Mac : voir ici
- Il est possible d'installer Ubuntu en dual boot, pour coder dessus

Git / Terminal

- Git (cheat sheet ici, sections : git basics, git branches, remote repositories, git diff, git pull, git push)
- Fiche sur Git et le terminal, rédigée par mes soins
- Jeu : Terminus (mieux maîtriser le terminal)
- Jeu : Hacknet pour entrer dans le hacking 🏴‍☠️
- Plus de commandes bash (terminal)
- Apprendre à taper plus vite : keybr.com
- Vim : éditeur 100% dans le terminal

Ressources (random) : Des conseils de ma part pour faire de meilleures candidatures

Je trouve que votre candidature donne envie, mais j'y vois des améliorations possibles (selon mon avis purement personnel). À votre place j'aurais beaucoup aimé obtenir des conseils, alors je me permets de les citer !

- Votre portfolio est génial, il faudrait le mettre beaucoup plus en avant : c'est très visuel, plus facile à lire que le CV actuel. Aussi on peut y voir les expériences et projets, ce qui est un plus (il faut montrer qu'on a déjà fait des choses, que ça marche, que c'est bien)
- Dans mon CV, quand j'étais dev junior je ne précisais pas que c'étaient des "stages d'été", je mettais simplement que c'étaient des expériences "Dev Fullstack", et je pense que ça m'a aidé (ne pas amoindrir la valeur de son expérience)
- Attention, beaucoup d'entreprises n'ont pas le temps de lire tout ce que tu as envoyé, ça fait beaucoup. Ton "travail" est de donner très envie, mais en restant très synthétique, et de ne garder que ce qui correspond le mieux à la boîte que tu cible
- Si tu parles un petit peu de la boite ou tu veux aller, ça montre de l'intérêt : "j'imagine que vous faites telle ou telle chose chez nom_de_la_boite, ce qui m'attire particulièrement" etc
- Toujours pour montrer à la boîte que tu es fait pour elle : parle brièvement de tes projets ou expériences pro les plus alignées avec la boite que tu vises (ex: pour La Web Squad, tu peux mettre plus en avant le côté React etc), "j'ai fait tel projet, ainsi que ça et ça" en donnant les liens vers ceux-ci si possible
- Honnêtement, je ne sais pas si les personnes prennent le temps de lire les lettres de motivation en pièce jointe. Ce que j'ai donné juste avant permet surement dejà assez de solliciter suffisamment l'intérêt
- À la fin du mail, tu peux donner plus d'éléments de contact, et même un lien de prise de rendez-vous pour faciliter la réservation de visio ou appel (tu auras déjà amoindri le nombre d'étapes pour discuter avec toi : on saura déjà quelles sont tes dispos). Par exemple, utilise calendly.