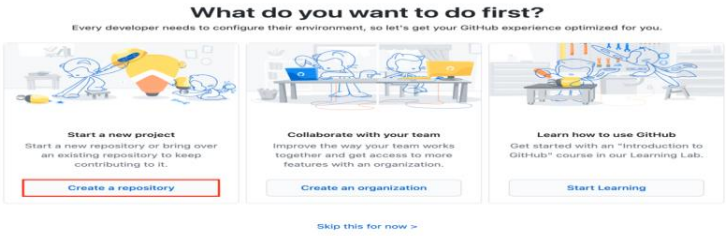
TP n°6 en Systèmes distribués (DevOps)

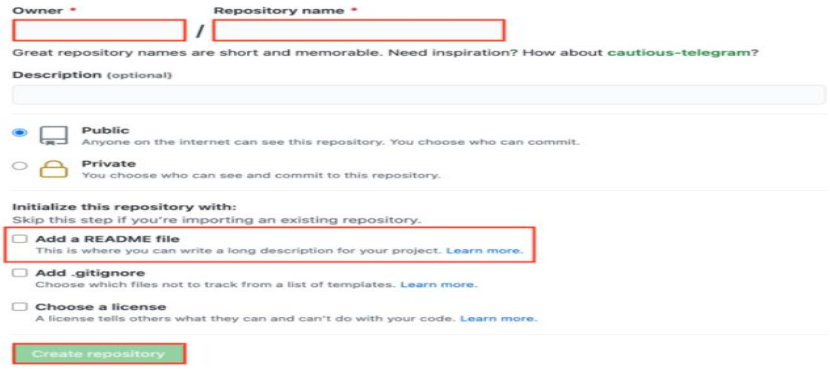
Exercice 1 : Créer un dépôt GitHub

Un dépôt, ou repo, sera le centre du projet. Il peut s'agir d'un fichier ou d'une collection de fichiers contenant du code, des images, du texte ou tout autre élément.
Pour commencer le processus, suiver les étapes suivantes :

Cliquer sur **Créer un dépôt** pour démarrer un nouveau projet (Ex. India2324)



La section **Owner** aura déjà votre nom de compte. Créer un **nom de dépôt**. Vérifier s'il est défini sur **Public** pour le rendre open-source, puis cocher la case **Ajouter un fichier README**. Enfin, cliquer sur **Créer un dépôt**.

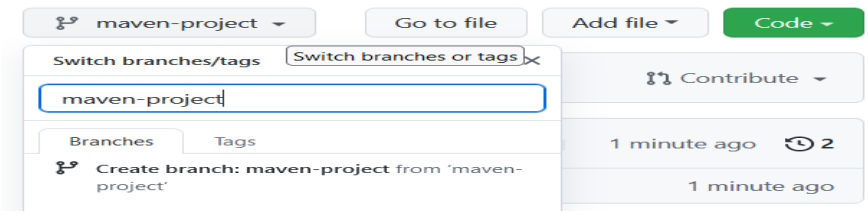


1. Créer de branches GitHub

Git offre des branches de fonctionnalités. Cela signifie que chaque ingénieur logiciel de l'équipe peut créer une branche de fonctionnalité qui fournit un dépôt local isolé pour apporter des modifications au code.
Les branches de fonctionnalités n'affectent pas la branche principale, où se trouve le code original du projet. Une fois les modifications effectuées et le code mis à jour, la branche de fonctionnalité peut être fusionnée avec la branche principale, et c'est ainsi que les modifications apportées au projet deviennent effectives.

Voici comment vous pouvez générer une branche de fonctionnalité :

Aller dans votre nouveau dépôt. Appuyer sur le bouton **principal** et entrer le nom de votre nouvelle branche de fonctionnalité. Cliquer sur **Créer une branche.**



Vous avez maintenant créé une branche de fonctionnalité qui est identique à la branche principale. Vous pouvez commencer à y apporter des modifications librement sans affecter le projet.

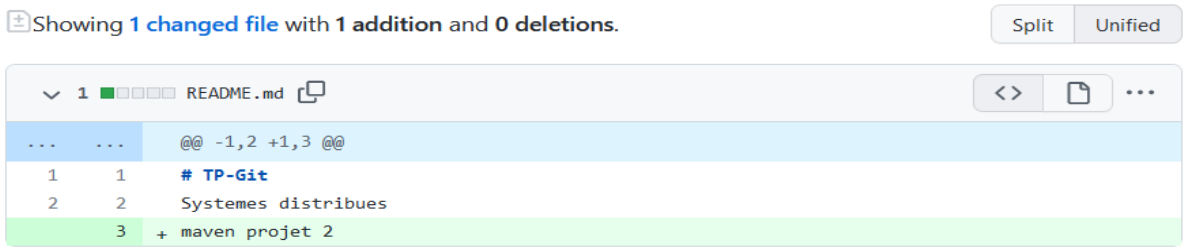
2. Comprendre les commits GitHub

Les commits sont ce qu'on appelle les modifications enregistrées sur GitHub. Chaque fois que vous modifier le fichier de la branche de fonctionnalité, vous devez le **« commiter »** pour le conserver.
Voici comment effectuer et valider un changement :
Accéder à la branche de fonctionnalité en cliquant sur **principal** et en sélectionnant votre branche nouvellement créée dans le menu déroulant.
Cliquer sur l'icône « crayon » pour commencer à modifier le fichier. Une fois que vous avez terminé, écrire une brève description des modifications apportées. Cliquer sur **Commiter les modifications**.

3. Créer des demandes de retrait sur GitHub

Pour proposer les modifications que vous venez d'apporter aux autres développeurs travaillant sur le même projet, vous devez créer une demande de retrait (**Pull request**). C'est ce qui rend le travail en commun sur les projets si faciles, car c'est le principal outil de collaboration sur GitHub.
Les demandes de retrait vous permettent de voir les différences entre le projet original et votre branche de fonctionnalité. C'est ainsi que vous demandez à vos pairs de les réviser. Si les autres développeurs approuvent, ils peuvent fusionner **la demande de retrait**, ce qui appliquera ces changements au projet principal.
Pour faire une requête pull, suivre les étapes ci-dessous :

1. Cliquer sur **Pull requests** -> **Nouvelle pull request**. Dans la section **Comparaisons d'exemples**, sélectionner **la branche de fonctionnalité** sur laquelle vous étiez en train de travailler.



2. Examiner les modifications une fois de plus et cliquer sur **Créer une pull request**. Sur la nouvelle page, écrire le titre et fournir une courte description de ce sur quoi vous avez travaillé pour encourager la fusion. Cliquer sur **Créer une pull request** .

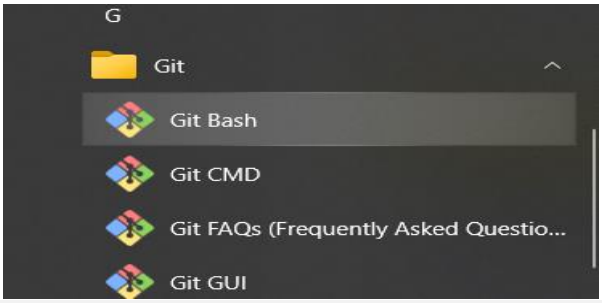
Exercice 2 : Dépôt local sur Git

1. Télécharger et installer Git

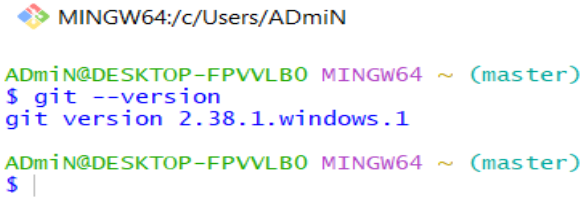
Suivre les instructions fournies par GitHub pour créer une copie locale du dépôt. Dans l'ordre:

1. **git init** permet d'initialiser un dépôt local vide dans votre répertoire courant ;
2. **git add** permet d'indexer un fichier ou un répertoire
3. **git commit** permet de valider les modifications sur votre dépôt local ;
4. **git remote add <nom> <url> :** permet de lier votre dépôt local (initie par init) `a un dépôt distant (url sur GitHub) ;
5. **git push** permet d'envoyer les modifications indexées au dépôt distant.

2. Démarrer Git, afficher la version de git



$ git –version



3. Récupérer sur votre dossier les sources de votre projet sur GitHub.

$ git clone https://github.com/.../ India2324.git

4. Indiquer le dépôt officiel :

$ git remote add official https://github.com/.../ India2324.git

5. Créer dans le dossier INDIA2324 un fichier texte « demo.txt »
6. Afficher le contenu de ce fichier
7. Configurer le nom utilisateur et l'adresse email

$ git config --global user.name "demo"
$ git config --global user.email demo@gmail.com

8. Enregistrer une image du fichier dans git

$ git add demo.txt
$ git commit -m "git pour devops"

9. Vérifier l'état du fichier

$ git status

10. Afficher l'historique

$ git log

11. Restaurer le premier état du projet

$ git checkout Id-Fichier

12. Publier-le dans votre espace public

$ git push

13. Récupérer et tirer depuis des dépôts distants

$ git pull

14. Dans GitHub faire une demande d'intégration (pull request).
15. La branche par défaut dans Git s'appelle master et elle pointe vers le dernier des commits réaliser. Créer une nouvelle branche

$ git branch spring-project

16. Basculer vers une branche existante

$ git checkout spring-project

Tous les commits à ce moment sont fait sur la branche courante.

17. Retourner sur la branche principale

$ git checkout master

18. Mettre à jour votre disque dur jusqu'à ce que vous récupériez les modifications de quelqu'un d'autre.

$ git pull official master

19. Régler le conflit. Modifiez la mise en page de la liste pour qu'elle soit correcte.

git merge <branche>

20. Une fois que le conflit est réglé par une révision, publiez l'ensemble des révisions dans votre espace public :

```
$ git push
```

21. Dans GitHub faire une demande d'intégration (pull request).

Exercice 3: Getting started with Docker

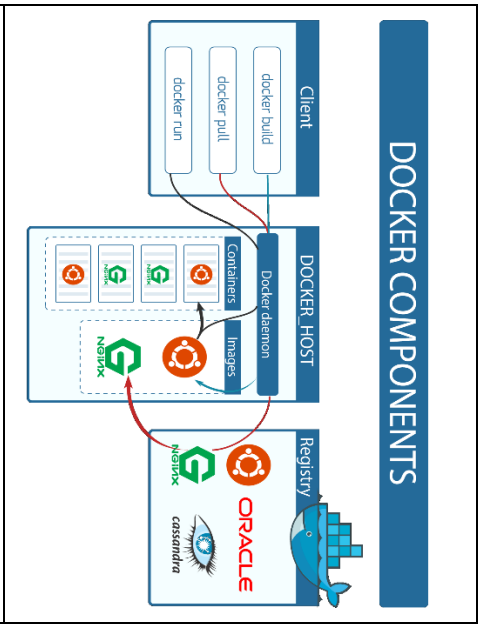
– Docker est un projet open source (Apache 2.0) écrit en GO et hébergé sur GitHub: https://github.com/docker (https://github.com/docker).

Docker est composé de trois éléments :

- le daemon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (Containerd avec runC)
- une API de type REST qui permet de communiquer avec le daemon
- Le client en CLI (command line interface) : commande docker

Lexique

- **Conteneur** : Image exécutable d'un environnement complet incluant code, librairies, outils et configuration
- **Image** : template de conteneur en read-only contenant un systeme de base et une application.
- **Docker HUB** : Dépôt public d'images mises à disposition par Docker DockerHub (https://store.docker.com)
- **Dockerfile** : fichier texte de description d'une image
- **Docker Compose** : fichier texte (yaml) de description d'un ensemble de conteneurs
- **Docker Machine** : Outil de déploiement des hôtes Docker sur différentes plateformes (Mac, Windows) : https://docs.docker.com/machine/overview/ (https://docs.docker.com/machine/overview/)
- **Orchestrateur** : gère un pool de ressources serveurs (Swarm, Kubernetes, Mesos, Rancher…)
- **Registry** : Dépôt privé d'images Docker



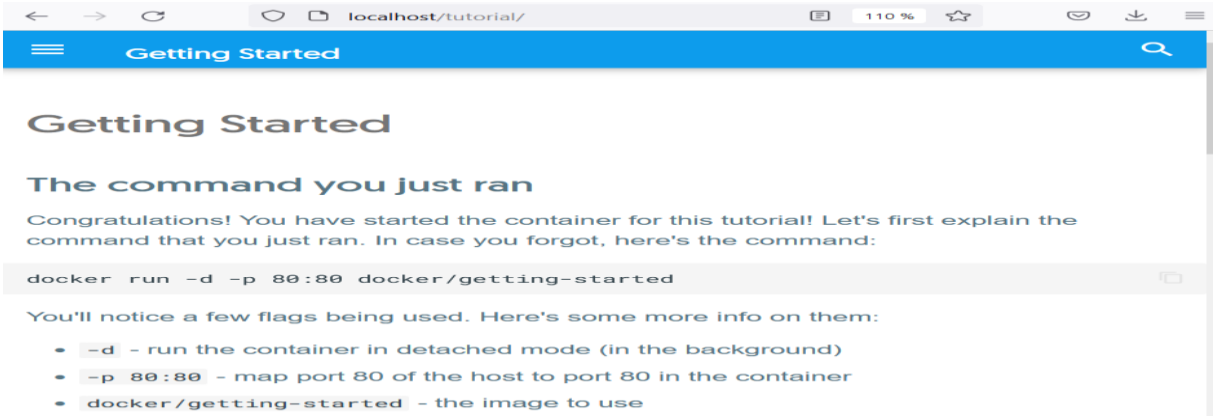
Quelques exemples de commandes docker :

- Connaître la version de Docker installé : **docker -- version**
- Lancer un conteneur: **docker run -d -p 80:80 docker/getting-started**
- Lister tous les conteneurs : **docker ps -a**
- Obtenir toutes les images présentes sur le système : **docker images**
- Construire une image à partir d'un Dockerfile : **docker build https://github.com/docker/rootfs.git#container:docker**
- Supprimer un conteneur : **docker rm ID_du_conteneur**
- Supprimer une image : **docker rmi ID_de_l_image**
- Afficher les logs d'un conteneur avec leurs détails : **docker logs --details**

1. Télécharger et installer Docker Desktop
2. Télécharger et installer l'application Kitematic
   Kitematic est une application simple mais puissante pour gérer les conteneurs Docker sur Mac et Windows. Il dispose d'un nouveau tableau de bord de bureau Docker pour une expérience utilisateur encore meilleure, avec l'intégration de Docker Hub et de nombreuses fonctionnalités.
3. Apres l'installation du Docker, lancer votre premier conteneur

```
C:\WINDOWS\system32>docker run -d -p 80:80 docker/getting-started
```

C'est la commande run qui va exécuter l'image afin de créer le conteneur. Pour une première utilisation, Docker va tout d'abord télécharger l'image à partir de Docker Hub, comme l'image le montre



4. Renvoyer la liste des conteneurs en cours d'exécution avec des détails les concernant.
   docker ps
5. Télécharger l'image « *hello-world* » depuis le Docker Hub : Il faut savoir que dans la plupart des cas, vous allez vous baser sur des images existantes, y apporter quelques modifications afin de construire votre propre image.
   docker pull hello-world
6. Lister les images présentes sur votre machine
   docker images
7. Lister tous les ocnteneurs présents sur votre machine
   docker ps -a
8. Arrêter le conteneur « hello-word »
   docker stop id_du_conteneur
9. Supprimer le conteneur « hello-word »
   docker rm id_du_conteneur
10. Supprimer l'image « *hello-world* »
    docker rmi id_image

Exercice 4 : Création une nouvelle image à partir du fichier Dockerfile

Dockerfile : est la base sur laquelle tous les conteneurs sont construits. Il contient l'ensemble des commandes à exécuter afin de construire l'image à savoir le système d'exploitation de base du conteneur, les langages utilisés, l'environnement et ses variables, l'emplacement des fichiers, etc.

Image : Une fois le Dockerfile écrit, on peut maintenant créer l'image d'un conteneur. Une image est un modèle à lecture seule pour les conteneurs. On y retrouve tout le code source de l'application à conteneuriser, ses dépendances, les bibliothèques à utiliser ainsi que d'autres outils nécessaires à la création du conteneur.

Voici les commandes que l'on retrouve fréquemment dans un Dockerfile :

- **FROM** : c'est la commande qui sert à définir l'image de base de l'application ;
- **LABEL** : qui permet d'ajouter des métadonnées à l'image ;
- **RUN** : qui sert à exécuter des instructions dans le conteneur ;
- **ADD** : pour ajouter des fichiers dans l'image ;
- **WORKDIR** : afin de spécifier le répertoire dans lequel sera exécuté tout l'ensemble des instructions
- **EXPOSE** (facultatif) : qui sert à spécifier le port que l'on souhaite utiliser ;
- **VOLUME** (facultatif) : pour spécifier le répertoire à partager avec l'hôte ;
- **CMD** : qui sert à indiquer au conteneur la commande à exécuter lors de son lancement.

https://github.com/adamhalasz/docker-node-example.git

```
FROM node:6.10.3

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 9000
CMD [ "npm", "start" ]
```

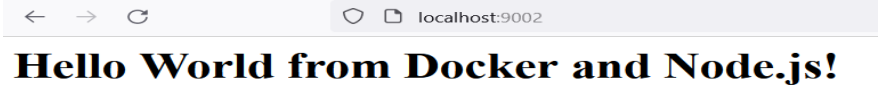
1. Créer le dossier tpdocker
2. Récupérer sur votre dossier « tpdocker » les sources du projet "https://github.com/adamhalasz/docker-node-example.git »
3. Lister le contenu du dossier « docker-node-example"
4. Construire l'image « nodeimage »

```
C:\Windows\System32\devopsdocker\docker-node-example>docker build -t nodeimage .
```

   - **-t** permet de spécifier le nom que l'on donne à l'image
   - "**.**" à la fin de la commande désigne l'emplacement de la source de l'image. Cet emplacement vient de l'instruction que vous avez écrite sur la commande COPY du dockerfile "."

5. Exécuter ce conteneur docker « nodeimage » sur le port 9002

```
C:\Windows\System32\devopsdocker\docker-node-example>docker run -d --name nodeimage -p 9002:9000 nodeimage
c8f6698518c1886cd0b61e33e252351ef5303c46d9f935b848f70d54ab5d4c1f
```



6. Lancer l'application dans un serveur Nginx

Nginx, que l'on prononce "engine-x", est un logiciel open source sous licence 2-clause BSD qui peut prendre plusieurs rôles. En effet, il peut être un serveur proxy inverse, un serveur web, un cache HTTP, un proxy de messagerie et un équilibreur de charge sur lequel plusieurs serveurs web s'appuient.
C'est en sa qualité de serveur web que Docker a décidé de l'intégrer dans sa plateforme, car il permet de développer des applications web et de les héberger, quel que soit le système d'exploitation utilisé.

Pour que le serveur Nginx puisse héberger correctement l'application, il faut que l'on effectue quelques modifications à l'application que nous venons de lancer dans le conteneur ci-dessus.
Tout d'abord, nous allons configurer notre propre serveur Nginx pour y intégrer notre application.

L'objectif est de pouvoir lancer le conteneur du Nginx de base. Mais au lieu d'afficher la page d'accueil de celui-ci, nous allons le rediriger vers notre application.

Pour cela, créer un dossier nommé "*nginx*" à la racine du projet. Ensuite, créer un fichier nommé "*nginx.conf*" dans lequel vous allez copier les instructions suivantes :

```
server {
    listen [::]:81;
    listen 81;
    server_name nodeserver;
    charset utf-8;
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_pass http://nodeimage:9002;
    }
}
```

Toujours à l'intérieur de ce dossier, vous allez également créer un Dockerfile. Celui-ci contient la commande qui remplace *default.conf* (la configuration de base de Nginx) par celui que vous venez de créer. Voici les commandes que vous allez saisir à l'intérieur du fichier :

```
FROM nginx
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Maintenant, nous allons créer un docker-compose qui va lancer les deux conteneurs simultanément en référence aux deux Dockerfile du projet. Pour ce faire, créer un fichier appelé "*docker-compose.yml*" à la racine du projet puis ajoutez les instructions ci-dessous :

```
version: "3.8"
services:
    nodeimage:
        build:
            context: .
        ports:
            - "9002:9002"
    nginx:
        restart: always
        build:
            context: ./nginx
        ports:
            - "81:81"
```

Voilà à quoi ressemble un docker-compose.yml en général.
À la première ligne, il faut spécifier la version utilisée. Ensuite, on spécifie les propriétés de chaque conteneur à lancer à l'intérieur de l'instruction « *services* ». Dans notre cas, *nodeserver* est l'application à lancer. Le contexte indique l'emplacement de cette dernière et le port sur lequel elle démarre.
Ensuite, on spécifie également les propriétés du serveur Nginx de la même manière.
Une fois que tout cela est fait, vous êtes prêt à lancer votre application dans un serveur Nginx.
Pour cela, tapez ce qui suit dans une invite de commande :

```
docker-compose up --build
```

Dès que l'exécution se termine, rendez-vous sur votre navigateur et tapez localhost, l'adresse utilisée lorsque vous lancez un serveur Nginx de base. Vous allez constater que vous allez voir la même interface que nous avons lancée auparavant.