





Travaux pratiques, consignes et aides

 Owner	 louis reynouard
 Tags	
 Created time	@February 28, 2024 10:39 AM

TP - Chapitre 1 : Création de l'Application de Todo List

[Prérequis, objectifs et déroulé](#)

[Le tp pas à pas : Installation de Django](#)

[Création d'un Nouveau Projet Django](#)

[Lancement du Serveur de Développement](#)

[Création de l'Application Todo List](#)

TP - Chapitre 2 : Création de l'Application Web et Empaquetage dans une Image Docker

[Prérequis, objectifs et déroulé](#)

[Préparation du Dockerfile](#)

[Construction de l'Image Docker](#)

[Exécution de l'Application dans un Conteneur Docker](#)

[Utilisation de Docker Compose](#)

[Écriture et Intégration des Tests](#)

TP - Chapitre 3 : Déploiement de l'Application Dockerisée sur Kubernetes avec Minikube

[Prérequis, objectifs et déroulé](#)

[Mise en place de minikube pour exécuter un cluster Kubernetes localement](#)

TP - Chapitre 4 : Création d'un Pipeline CI/CD dans GitLab pour l'Application

[Prérequis, objectifs et déroulé](#)

[Documentation officielle Django](#)

[Pousser l'image Docker sur Docker Hub](#)

[Avant d'aller plus loin](#)

[Création et configuration de l'Environnement GitLab](#)

[Création du Projet GitLab](#)

[Configuration des Variables d'Environnement dans GitLab](#)

[Mise en Place du Pipeline CI/CD](#)

[Création du fichier `.gitlab-ci.yml`](#)

[Développement de la fonctionnalité de modification de todo](#)

[Publication et Déploiement Automatique](#)

[Mise à Jour Locale de l'Image Docker](#)

[Écriture et Intégration des Tests](#)

TP - Chapitre 1 : Création de l'Application de Todo List

Maintenant que nous avons une compréhension de base de l'ingénierie logicielle et du développement web avec Django, il est temps de mettre en pratique ces connaissances. Vous allez démarrer le projet TP en créant une application web de todo list simple avec Django. Cette section vous guidera à travers les étapes initiales pour mettre en place votre projet et créer l'application.



Avant toute chose, écrivez le SDLC de notre projet. Celui ci sera demandé en complément de votre TP.

Prérequis, objectifs et déroulé

Assurez-vous d'avoir Python installé sur votre système. Django peut être installé à l'aide de pip, le système de gestion de paquets pour Python.

- **Objectif:** Développer une application web simple de todo list permettant aux utilisateurs de créer, afficher et signaler comme faite les tâches
- **Fonctionnalités de base:**
 - Interface utilisateur pour ajouter de nouvelles tâches.
 - Liste des tâches affichant toutes les entrées disponibles.
 - Options pour marquer les tâches comme complétées



Points supplémentaires: Introduire des fonctionnalités avancées comme la priorisation des tâches, le tri basé sur l'état (complété/incomplet)

Le tp pas à pas : Installation de Django

1. Ouvrez un terminal ou une invite de commande.
2. Vérifiez que conda est installé

```
conda --version
```

Si pas installé:

1. Téléchargez la version Windows de Miniconda à partir du site officiel de Conda : <https://docs.conda.io/en/latest/miniconda.html>
 2. Choisissez la version correspondant à votre système d'exploitation (32 bits ou 64 bits). Si vous n'êtes pas sûr, vous pouvez vérifier en faisant un clic droit sur "Ce PC" dans l'Explorateur de fichiers, puis en sélectionnant "Propriétés" pour afficher les informations système.
 3. Une fois le fichier d'installation téléchargé, double-cliquez dessus pour l'exécuter.
 4. Suivez les instructions de l'installateur. Vous pouvez accepter les options par défaut ou les personnaliser selon vos préférences.
 5. Lorsque vous arrivez à l'étape "Installation Type", sélectionnez "Just Me" pour installer Conda pour votre utilisateur uniquement, ou "All Users" si vous souhaitez une installation globale accessible à tous les utilisateurs de l'ordinateur. Cliquez sur "Next" pour continuer.
 6. Sur la page suivante, choisissez le répertoire d'installation de Miniconda. Par défaut, il est installé dans votre répertoire utilisateur (par exemple, `C:\Users\YourUsername\Miniconda3`). Vous pouvez le modifier si vous le souhaitez, puis cliquez sur "Next".
 7. Sélectionnez les options "Add Miniconda3 to my PATH environment variable" et "Register Anaconda as the system Python 3.7". Cela permettra d'utiliser Conda et Python à partir de la ligne de commande. Cliquez sur "Install" pour lancer l'installation.
 8. Attendez que l'installation se termine. Une fois terminée, vous verrez un message indiquant que l'installation est terminée. Cliquez sur "Next" puis "Finish" pour fermer l'installateur.
 9. Pour vérifier si Conda est correctement installé, ouvrez une nouvelle fenêtre de commande et exécutez la commande suivante ci-dessus
3. Créer un environnement virtuel dédié à notre projet

```
conda create -n todo_env  
conda activate todo_env
```

4. Installez Django en exécutant la commande suivante :

```
pip install django
```

Création d'un Nouveau Projet Django

1. Une fois Django installé, créez un nouveau projet en exécutant :

```
django-admin startproject nom_de_votre_projet
```



Evidemment, remplacez `nom_de_votre_projet` par le nom que vous souhaitez donner à votre projet....

2. Accédez au dossier de votre projet :

```
cd nom_de_votre_projet
```

Lancement du Serveur de Développement

Pour vérifier que votre projet Django est correctement configuré, lancez le serveur de développement :

```
python manage.py runserver
```

Ouvrez un navigateur web et accédez à `http://127.0.0.1:8000/` . Vous devriez voir la page d'accueil par défaut de Django, indiquant que le projet fonctionne correctement.

Création de l'Application Todo List



Avant de vous jeter sur votre prompt ChatGpt ou équivalent, je vous recommande d'ouvrir la [documentation officielle de Django](#), qui vous accompagnera pas à pas dans les différentes étapes ci-dessous

Maintenant, créez une nouvelle application Django au sein de votre projet pour gérer la todo list :

1. Exécutez la commande suivante pour créer votre application :

```
python manage.py startapp my_app_name
```



Remplacez `my_app_name` par le nom que vous souhaitez donner à votre application de gestion des tâches.

2. Pour inclure votre application dans le projet, ouvrez le fichier `settings.py` situé dans le dossier du projet Django (`nom_de_votre_projet/nom_de_votre_projet/settings.py`), et ajoutez l'application à la liste `INSTALLED_APPS` :

```
INSTALLED_APPS = [  
    ...  
    "my_app_name",  
]
```

3. Ecrivez vos modèles de données dans `models.py`
4. Faites vos migrations

```
python manage.py makemigrations my_app_name  
python manage.py migrate
```



Quelle est la différence entre ces deux lignes? Pourquoi 2 étapes?

5. Inclure les routes dans le dossier de l'application
 - a. créez le fichier `urls.py`

- b. Ajoutez les différentes urls dont vous avez besoin
- 6. Ajoutez les urls du point précédent dans les urls du projet globale
- 7. Faites vos templates
- 8. Ajoutez les vues correspondantes

TP - Chapitre 2 : Création de l'Application Web et Empaquetage dans une Image Docker

Prérequis, objectifs et déroulé

- **Objectif:** Empaqueter l'application web dans un conteneur Docker pour faciliter le déploiement et l'exécution dans différents environnements.
- **Étapes:**
 - Création d'un fichier `Dockerfile` pour l'application Django.
 - Construction de l'image Docker de l'application
 - Exécution du conteneur localement



Points supplémentaires: la page d'accueil 127.0.0.1:8000 ne doit pas renvoyer une erreur

Préparation du Dockerfile

Créez un fichier `Dockerfile` à la racine de votre projet Django. Le Dockerfile doit inclure les instructions pour installer Python, copier les fichiers de votre application dans le conteneur, installer les dépendances spécifiées dans le fichier `requirements.txt`, et lancer l'application Django.

```
# Utiliser l'image officielle Python comme image parente
FROM python:3.8
```

```
# Définir le répertoire de travail dans le conteneur
```

```
WORKDIR /app
```

```
# Copier les fichiers du projet dans le conteneur
```

```
COPY . /app
```

```
# Installer les dépendances
```

```
RUN pip install -r requirements.txt
```

```
# Exposer le port sur lequel l'application s'exécute
```

```
EXPOSE 8000
```

```
# Commande pour démarrer l'application
```

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```



Si le fichier `Dockerfile` vous permet d'installer les dépendances spécifiées dans le fichier `requirements.txt` il vous faudra créer ce dernier... et s'assurer qu'il soit à jour !

Construction de l'Image Docker

Dans votre terminal, naviguez jusqu'au répertoire de votre projet et exécutez la commande suivante pour construire l'image Docker de votre application :

```
docker build -t todo-app .
```



Que signifie le "." dans la ligne de commande ci-dessus?

Exécution de l'Application dans un Conteneur Docker

Une fois l'image construite, lancez un conteneur à partir de cette image :

```
docker run -d -p 8000:8000 todo-app
```

Cette commande lance l'application en mode détaché et mappe le port 8000 du conteneur au port 8000 de l'hôte, permettant ainsi d'accéder à l'application via <http://localhost:8000> .

Utilisation de Docker Compose

Hot reload, connecte le dossier courant avec le dossier de l'appli dans le docker

```
# compose.yaml
services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./app
```

Écriture et Intégration des Tests

- Écrivez les tests pour la totalité de votre application Django

TP - Chapitre 3 : Déploiement de l'Application Dockerisée sur Kubernetes avec Minikube

Prérequis, objectifs et déroulé

- **Objectif:** Utiliser minikube pour déployer et gérer l'application conteneurisée au sein d'un cluster kubernetes local
- **Étapes:**
 - Configuration d'un cluster Kubernetes local avec Minikube.
 - Déploiement de l'application sur Kubernetes à l'aide des Deployments et Services appropriés.
 - Rapport précis du déploiement, du state et des choix techniques fait

Mise en place de minikube pour exécuter un cluster Kubernetes localement

Minikube est un outil qui permet de lancer un cluster Kubernetes localement. Il est idéal pour le développement et le test de vos applications Kubernetes.

Poursuivez le TP en suivant les prochaines étapes:

1. Installez kubectl sur votre machine
2. Installez Minikube sur votre machine
3. Démarrez un cluster Minikube. (ajouter si nécessaire `—driver=docker`)

```
minikube start
```



Il peut être nécessaire de rajouter `—driver=docker` pour préciser de lancer minikube sur docker dans le cas où votre OS est W10

4. Vérifiez que minikube a bien démarré

```
minikube status
```

5. Vérifiez la liste des composants

```
kubectl get all
```

6. Si certains composants sont déjà en place, plusieurs possibilités pour les supprimer

```
# Supprimer le deployment
```

```
kubectl delete deployment deployment-name
```

```
# Supprimer le service
```

```
kubectl delete service service-name
```

```
# Supprimer tous les pods, services et deployments dans l'espace de nom
```

```
kubectl delete all --all
```

7. Créez un fichier yaml de configuration pour notre déploiement de to do list

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon_nom
spec:
  selector:
    matchLabels:
      app: mon_nom
  template:
    metadata:
      labels:
        app: mon_nom
    spec:
      containers:
      - name: mon_nom
        image: mon_image_docker_hub
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: numero_de_port
```

8. Créez un fichier yaml de configuration du service permettant d'y accéder

```
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mon_nom
spec:
  type: LoadBalancer
  selector:
    app: mon_nom
  ports:
```

```
- protocol: "TCP"
  port: 80
  targetPort: numero_de_port
```



Pensez à mettre à jour les noms des variables svp...

9. Modifier la variable d'environnement suivante pour permettre à Minikube et Docker d'utiliser le même daemon (cf chapitre 2)

```
eval $(minikube docker-env)
```

10. Construisez à nouveau votre image dans le daemon Docker de Minikube

```
docker build -t my-image .
```

11. Assurez vous que le nom de l'image dans le fichier de configuration de déploiement est bien le même que celui du build (e.g., `my-image`)
12. Configurez dans le fichier de configuration du déploiement le `imagePullPolicy` à `Never` pour préciser à Kubernetes de ne pas essayer de télécharger l'image.



Si vous souhaitez vérifier que l'image est bien présente dans le daemon Minikube, utiliser les commandes:

```
minikube ssh
docker images
exit
```

13. Ajoutez vos composants via la commande

```
kubectl create --filename my_file.yaml
```

14. Vérifiez la liste des composants

```
kubectl get all
```

15. Vérifiez le bon fonctionnement de tous vos composants avec l'UI de minikube

```
minikube dashboard
```

16. Obtenez l'adresse et le port de votre service

```
minikube service nom_de_mon_app
```

TP - Chapitre 4 : Création d'un Pipeline CI/CD dans GitLab pour l'Application

Nous allons maintenant appliquer ces principes CI/CD dans notre projet actuel

Prérequis, objectifs et déroulé

- **Objectif:** Automatiser le processus de déploiement de l'application sur docker puis kubernetes en utilisant GitLab CI/CD.
- **Étapes:**
 - Déploiement de l'image de notre projet sur docker hub
 - Configuration d'un pipeline CI/CD dans GitLab pour l'intégration et le déploiement continus.
 - Création des classes test de l'application de To Do List
 - Automatisation des tests, de la construction des images Docker, et du déploiement sur Kubernetes.
 - Utilisation des variables d'environnement et des secrets pour sécuriser le pipeline.
 - Ajout d'une fonctionnalité de modification de todo dans l'application via le pipeline précédemment construit



Bonus: écrivez un script shell pour automatiser la mise à jour de votre docker local avec votre nouvelle image

Documentation officielle Django



Retrouvez [les tests avec Django](#)

Pousser l'image Docker sur Docker Hub

Avant de poursuivre, nous allons publier notre application sur docker hub qui est un répertoire d'images docker en ligne. Cela nous permettra de complètement automatiser notre pipeline CI/CD. Voici les étapes à suivre:

1. **Créer un compte sur Docker Hub** : Si vous n'avez pas déjà de compte Docker Hub, rendez-vous sur <https://hub.docker.com/> et créez un compte.
2. **Se connecter à Docker Hub** : Ouvrez un terminal et exécutez la commande suivante pour vous connecter à Docker Hub en utilisant votre nom d'utilisateur et votre mot de passe :

```
docker login
```

3. **Construire l'image Docker** : Assurez-vous que vous avez construit l'image Docker que vous souhaitez pousser sur Docker Hub en suivant les étapes précédentes du tp.
4. **Tagger l'image** : Avant de pousser l'image, vous devez la marquer avec le nom de l'utilisateur Docker Hub et le nom du référentiel (repository) que vous avez créé. Utilisez la commande suivante pour tagger l'image :

```
docker tag nom-de-l-image:tag nom-d-utilisateur/nom-du-referentiel:tag
```

Par exemple, si votre nom d'utilisateur est "monutilisateur", le nom du référentiel est "monapp" et le tag est "v1", vous utiliserez la commande suivante :

```
docker tag nom-de-l-image:v1 monutilisateur/monapp:v1
```

5. **Pousser l'image** : Une fois que l'image est taggée, vous pouvez la pousser sur Docker Hub en utilisant la commande suivante :

```
docker push nom-d-utilisateur/nom-du-referentiel:tag
```

Reprenant l'exemple précédent :

```
docker push monutilisateur/monapp:v1
```

6. **Vérifier sur Docker Hub** : Après avoir poussé l'image, rendez-vous sur votre compte Docker Hub et accédez au référentiel que vous avez créé. Vous devriez voir l'image avec le tag correspondant.

Avant d'aller plus loin

1. Importez votre image depuis docker docker hub

```
docker pull monutilisateur/monapp:v1
```

2. Démarrez un docker local utilisant cette image

```
docker run --name docker-from-hub -d -p port:port monutilisateur/monapp
```

Création et configuration de l'Environnement GitLab

Si vous n'en avez pas déjà un, [créez votre compte gitlab](#)

Création du Projet GitLab

- Créez un nouveau projet sur GitLab.
- Importez le code de votre application Django (via commit)

Configuration des Variables d'Environnement dans GitLab

Dans **Settings > CI/CD > Variables**, ajoutez les variables suivantes :

- **DOCKER_USERNAME** : Votre nom d'utilisateur sur Docker Hub.

- **DOCKER_PASSWORD** : Votre token d'accès personnel sur Docker Hub.



Pour générer votre token c'est [ici](#)

Mise en Place du Pipeline CI/CD

Création du fichier `.gitlab-ci.yml`

À la racine de votre projet, créez ou modifiez le fichier `.gitlab-ci.yml` qui définira les étapes de votre pipeline CI/CD. Partez de la base ci-dessous :

```
stages:
  - build
  - test
  - deploy

variables:
  IMAGE_TAG: $DOCKER_USERNAME/my-django-app:latest

build:
  stage: build
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
    - rm -f /root/.docker/config.json

only:
  - main

test:
  stage: test
  script:
    - echo "Exécution des tests..."
```

```
only:
  - main

deploy:
  stage: deploy
  script:
    - echo "Le déploiement peut être géré ici si nécessaire."
  only:
    - main
```



Vérifiez dans votre interface Gitlab la bonne exécution de votre pipeline. Que fais la partie build ci-dessus?

Développement de la fonctionnalité de modification de todo

- Implémentez la nouvelle fonctionnalité permettant de modifier des to-dos existants.
- Développez les tests correspondants
- Testez la nouvelle fonctionnalité localement pour s'assurer de son bon fonctionnement.

Publication et Déploiement Automatique

- Une fois les modifications terminées et testées localement, poussez-les sur la branche principale de votre projet GitLab.
- Le pipeline CI/CD se déclenchera automatiquement, exécutant les étapes de construction, de test, et de déploiement.



Votre fonctionnalité doit se déployer et passer votre pipeline sans aucun problème

Mise à Jour Locale de l'Image Docker

Pour mettre à jour l'image Docker localement après un déploiement réussi :

```
docker pull $DOCKER_USERNAME/my-django-app:latest
```



Quelles étapes pour mettre à jour votre docker local avec la nouvelle image de votre application?

Testez la mise à jour de votre image en local et l'utilisation de votre nouvelle fonctionnalité après avoir pull votre image depuis DockerHub



Une fois cette partie finalisée, automatisez ce déploiement dans votre cluster minikube

Écriture et Intégration des Tests

- Intégrez vos tests dans le pipeline en ajustant la section `test` du fichier `.gitlab-ci.yml`



Assurez vous que la non validation de vos tests fasse échouer votre pipeline

- Ajouter une couverture minimale de test de 90% dans le fichier `.gitlab-ci.yml`



La qualité et la couverture de vos tests entrera dans la notation de ce TP.

Rendu final

Code TP

Partage de votre projet gitlab avec moi même. Ce dernier sera testé via:

1. Pull en local sur ma machine

2. Construction de l'image docker et lancement de container en local sur ma machine
3. Modification d'un titre sur votre page d'accueil
4. Commit du changement sur gitlab
 - a. Le pipeline ne doit pas renvoyer d'erreur
5. Pull de la nouvelle image depuis votre Docker Hub, relance du container local
 - a. le changement de titre doit être visible
6. Inspection du code de votre projet
 - a. Doit être commenté
 - b. couverture de test supérieure à 95%

Document projet TP

Ce document reprends:

1. Le SDLC de votre projet
2. Les choix techniques effectués
3. L'intégralité (surtout les commandes) du process à suivre pour que je puisse tester votre projet sur ma machine selon le process ci-dessus

Ex: étape 1, pull code gitlab local:

```
git init
git remote add origin https://....
git pull origin main
```