TD 1
Software Architecture for Cloud Computing

Objectifs

Ce premier TD vous fera découvrir les solutions PaaS/CaaS/FaaS vus en cours tout en mettant en application certains grands principes des 12 facteurs vus en cours (https://12factor.net/). Dans un premier temps, vous développerez une application web minimaliste, vous la déploierez dans un environnement local à des fins de développements puis vous la mettrez "en ligne" sur la solution PaaS puis dans un environnement CaaS. Vous aborderez enfin le minimum à comprendre en ce qui concerne le modèle FaaS.

L'application

L'application web minimaliste devra être réalisée dans un des langages supportés par les buildpacks officiels d'Heroku (https://devcenter.heroku.com/articles/buildpacks#officially-supported-buildpacks). L'idée est d'avoir, à terme, une application qui interagit avec une base de données en lecture ET écriture (PostgreSQL 14). Exemple : une page web affichant un compteur de visite qui s'incrémente à chaque appel ou une page permettant de soumettre un formulaire et une listant les informations soumises. A vous de choisir la finalité, nous vous conseillons d'aller au plus simple !

Vous prendrez soin d'utiliser git lors de votre développement, le projet devra être poussé sur github. Deux versions fonctionnelles de l'application devront être accessibles d'un simple checkout sur un commit :

- Une première version sans utilisation de la base de données (Une page hello world suffit)
- Une seconde version qui utilise la base de données (Compteur, Insert…)

Une version locale pour développer

Dans un premier temps vous travaillerez sur votre machine et vous veillerez à rendre l'application et la base de données déployables en local facilement au moyen d'un fichier docker-compose (https://github.com/NassimBounouas/si5-sacc-td1-docker-compose comprenant la base de données). Vous devrez pour se faire conteneuriser votre application au moyen d'un fichier Dockerfile (https://docs.docker.com/language/).

PaaS-ons au cloud

Vous allez maintenant déployer la première version de votre application (sans base de données) sur la solution PaaS Heroku (https://www.heroku.com/home). Pour cela, il vous faudra :

- Créer un compte Heroku
- Vous rendre sur le dashboard (https://dashboard.heroku.com/)
- Créer une nouvelle application
- Choisir une méthode de déploiement (Heroku Git)

- Installer ou mettre à jour le client CLI de Heroku
- Suivre les consignes présentées par Heroku

Connectez vous sur l'application et constatez qu'elle fonctionne comme en local. Si ce n'était le cas, faites en sorte d'avoir un fonctionnement similaire.

Vous allez ensuite déployer la seconde version (avec base de données). Pour cela, il faudra :

- Se rendre dans la section "Ressources" -> “Find more add-ons“
- Ajouter une ressource "Heroku Postgres"
- Se rendre dans les paramètres de l'application, section "Config vars" et constater l'apparition d'une variable *DATABASE_URL*
    - *Note : Pour ceux utilisant un buildpack Java, vous pouvez exécuter la commande “**heroku run -a mon_app echo \$JDBC_DATABASE_URL**”* cf : https://devcenter.heroku.com/articles/connecting-to-relational-databases-on-heroku-with-java#using-the-jdbc_database_url . La commande **“heroku run -a mon_app env”** vous listera l’ensemble des variables d’environnements y compris les “variables dynamiques” qui n’apparaissent pas sur le dashboard heroku.

- Déployer la seconde version de l'application (avec base de données). Cependant, cette fois-ci, en utilisant la connexion à github et en utilisant un déploiement automatique depuis une branche autre que **develop**.

Dans la limite du possible, vous ne devriez pas avoir à modifier le code source de votre application entre la version locale et la version "Cloud".

Du PaaS Au CaaS

Nous allons voir qu'il est possible d'utiliser Heroku comme une solution CaaS (container as a service). Reprenez votre application ou créez en une nouvelle (heroku vous propose de créer 5 applications gratuitement avec un compte non vérifié : https://www.heroku.com/free).

Vous avez construit dans la première partie du TD une image Docker pour votre application à utiliser dans votre docker-compose. Vous allez maintenant pousser cette image sur la Registry Docker Heroku (https://devcenter.heroku.com/articles/container-registry-and-runtime).

Votre image initiale nécessite certainement des modifications, assurez-vous que celles-ci n'altèrent pas le fonctionnement en local en modifiant si besoin le dockerfile et le docker-compose.

Parmi les changements que vous devriez/pouvez rencontrer :

- Sur Heroku les conteneurs ne doivent pas tourner en root (Bonne pratique docker)
- Votre application doit récupérer son port d’écoute de l’environnement fourni à votre image
- Voir:https://devcenter.heroku.com/articles/container-registry-and-runtime#dockerfile-commands-and-runtime

Face à FaaS

L'expérimentation du modèle FaaS se fera sur Google Cloud. Pour se faire, vous avez accès à des crédits google. **CES CRÉDITS VOUS SERONS UTILES POUR LE PROJET FIL-ROUGE DU COURS. VEILLEZ À NE PAS PUBLIER DE CREDENTIALS SUR GITHUB SI CES CREDENTIALS DEVAIENT ÊTRE UTILISÉ PAR VOTRE CODE (cf 12 factors 😉). DE PLUS, PENSEZ À ARRÊTER LES POTENTIELLES MACHINES CONSOMMANT TROP DE RESSOURCES LORSQU'ELLES NE SONT PAS UTILISÉES DURANT LE PROJET.**

L'objectif de cette partie est de voir le déploiement d'une fonction dans le modèle FaaS, pas de reproduire la totalité de l'application précédente. Vous êtes, selon votre vitesse d'avancement, libre de pousser plus loin l'expérimentation (voir partie "Pour aller plus loin").

Obtention des crédits google : Voir lien sur slack permettant d'activer vos crédits personnels et de les associer à un compte Google.

Déploiement Function as a Service :

- Créer un projet google cloud et y associer le compte de facturation correspondant à votre crédit
- Se rendre sur la console google cloud
- Rechercher le produit "Google Cloud Functions" (https://console.cloud.google.com/functions)
- Activer les éventuelles API nécessaires
- Sélectionner : créer une fonction
- Nommer votre fonction
- Choisir une région de déploiement
- Sélectionner un déclencheur de type HTTP
- Autoriser les appels non authentifiés
- Enregistrer la déclaration de la fonction et conserver son URL de déclenchement
- Passer à la fenêtre suivante
    - Comparer dans l'éditeur intégré la fonction Hello World par défaut en Java 17 et en Node.js 16
    - Que pouvez-vous constater des différents implémentations ?
- Déployer une fonction de votre choix, l'appeler et regarder les informations fournies par Google.

Pour aller plus loin (et éventuellement reproduire l'application déployée chez Heroku) : https://cloud.google.com/sql/docs/mysql/connect-functions