TP5 - Déploiement et surveillance

Dans ce TP, nous allons déployer l'application web et créer un tableau de bord simple pour la surveillance de notre application.

Base de code

Vous allez utiliser le même dépôt de code que lors de vos TPs précédents.

Objectifs

Dans la partie 1, nous allons déployer l'application web.

Dans la partie 2, nous allons créer un tableau de bord et une alerte.

Pour compléter ce TP, vous devrez suivre les étapes décrites dans les sections suivantes.

Note: Pour certaines parties de ce TP, les modifications dans Google Cloud peuvent prendre du temps à s'activer (ex: la création du dépôt dans le Artifact Repository ou les nouvelles métriques). Il faut parfois patienter quelques minutes.

Aussi, lors de la première requête au backend après un certain temps d'inactivité, il faut attendre que la machine virtuelle soit activée, donc il est normal que l'opération prenne quelques secondes.

Partie 1 - Déploiement de l'application web

Étape 1 - Crédits GCP

Dans ce TP, nous utilisons des services payants de la plateforme Cloud de Google.

Voici l'URL à laquelle vous devrez accéder pour demander un coupon Google Cloud. Il vous sera demandé de fournir l'adresse électronique et le nom de votre école. Un e-mail vous sera envoyé pour confirmer ces détails avant qu'un coupon ne vous soit envoyé.

Lien de récupération des coupons pour les élèves.

Il vous sera demandé de fournir un nom et une adresse électronique, qui doit correspondre au domaine de votre école. Un courriel de confirmation vous sera envoyé avec un code de coupon.

**** Associez le compte de paiement au compte Google que vous utilisez pour développer votre projet.**

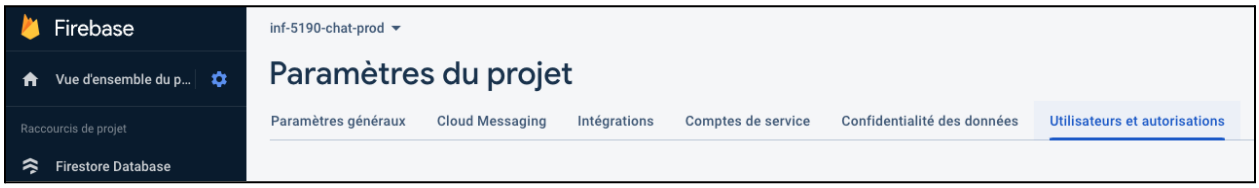
Étape 2 - Environnement de production

Avant de déployer notre application, nous allons créer un environnement de production en créant un projet **Firebase** dédié.

Pour la création du nouveau projet **Firebase**, suivez les mêmes étapes que lors du TP3. Dans ce cas-ci, vous n'aurez pas besoin de télécharger la clé privée.

En plus de **Firestore** et **Cloud Storage**, activez aussi **Hosting**. Pour **Hosting**, vous pouvez ignorer les étapes proposées pour le moment, nous y reviendrons plus tard.

Ajoutez trepanier.felix-etienne@uqam.ca comme `Lecteur` dans la section `Utilisateurs et autorisations` (accessible via le bouton avec l'icône d'engrenage).

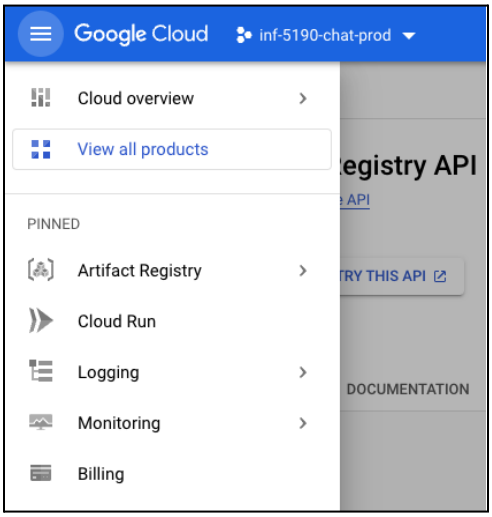


Étape 3 - Création du Artifact Registry

Sur https://console.cloud.google.com/, connectez-vous avec le compte Google que vous utilisez pour le projet.

Dans la barre de menu, sélectionnez le projet que vous venez de créer.

Dans le menu de gauche, sélectionnez **Artifact Registry**.

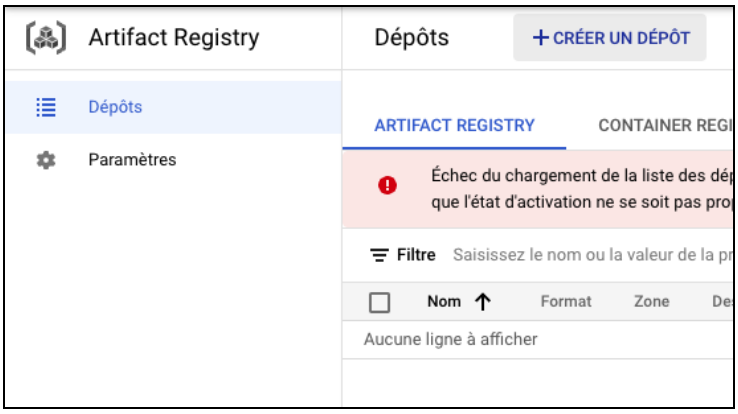


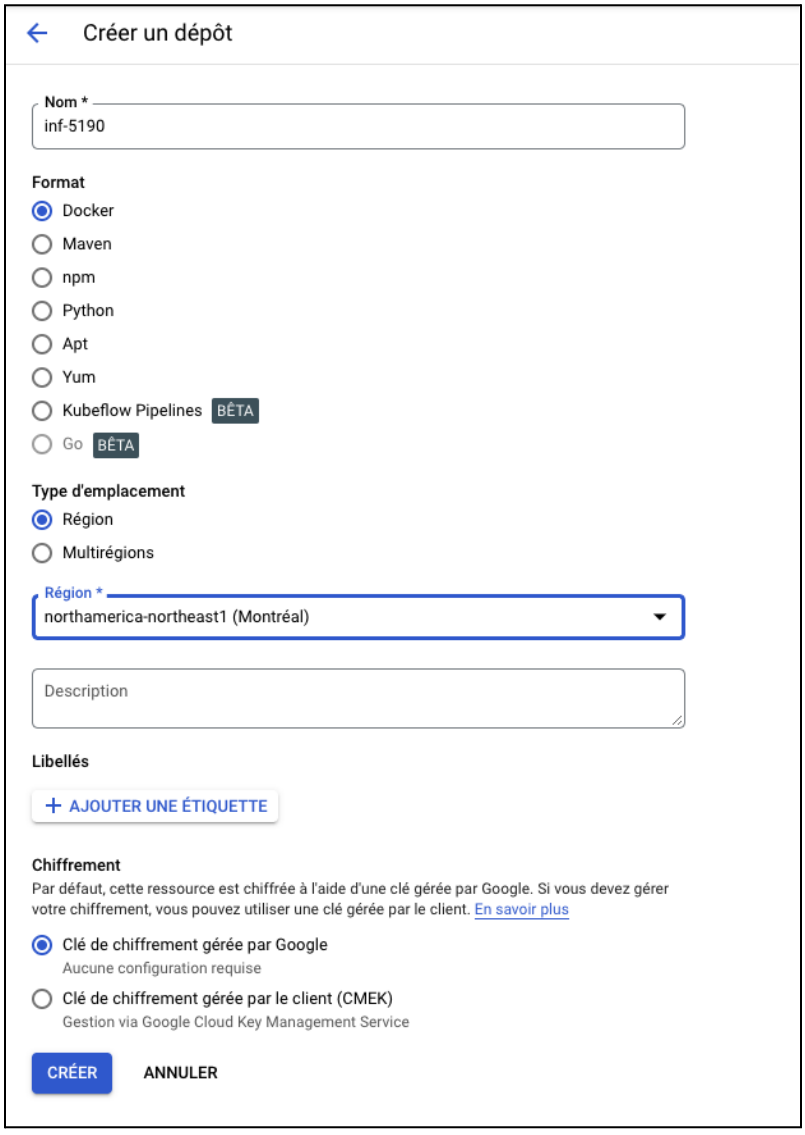
Activez l'API et ensuite la facturation.

Pour la facturation, vous choisissez le compte de paiement défini à l'étape 1 (compte de paiement **Billing Account for Education**).

Dans la page de l'**Artifact Registry**, créez un dépôt.

Voici les paramètres à utiliser:





L'address de votre dépôt devrait être

`northamerica-northeast1-docker.pkg.dev/<PROJECT_ID>/inf-5190`

Ce dépôt contiendra les images **Docker** que nous déploierons sur **Cloud Run**.

Étape 4 - Modifications du backend

Pour pouvoir déployer notre application dans un autre environnement, nous allons devoir modifier le code afin de permettre de configurer certains paramètres en utilisant des variables d'environnement.

Dans la classe `ChatApplication`, modifiez d'abord la méthode `getFirestore` et ajoutez les deux méthodes `getAllowedOrigins` et `getStorageBucketName` en vous aidant du code du gist suivant: https://gist.github.com/coderunner/58aa74bc3af114a3bacfc37454edd94d.

N'oubliez pas d'ajouter l'annotation `@PropertySource("classpath:cors.properties")` pour avoir accès au fichier de configuration des CORS.

Ensuite, dans les classes `CorsConfig` et `WebSocketConfig` utilisez le code suivant pour obtenir les allowedOrigins:

```
@Autowired
@Qualifier("allowedOrigins")
private String[] allowedOrigins;
```

Ajoutez une configuration `firebase.storage.bucket.name=<votre bucket name>` dans le fichier `firebase.properties`.

Dans la classe `MessageRepository`, utilisez le code suivant pour obtenir le nom du storage bucket:

```
@Autowired
@Qualifier("storageBucketName")
private String storageBucketName;
```

Ces modifications nous permettent de spécifier ces valeurs de configuration en utilisant des variables d'environnement et d'utiliser les valeurs définies dans les fichiers de configuration seulement si les variables d'environnement ne sont pas définies (comme pour l'environnement de développement).

Vérifiez que votre application fonctionne toujours correctement en utilisant simplement votre environnement de développement et en roulant les tests avant de passer à l'étape suivante.

Étape 5 - Création de l'image Docker

Dans le fichier `pom.xml`, ajouter le plugin jib dans la section <build><plugins>.

```
<plugin>
   <groupId>com.google.cloud.tools</groupId>
   <artifactId>jib-maven-plugin</artifactId>
   <version>3.3.1</version>
   <configuration>
      <to>
         <image>[[L'address de votre dépôt Docker défini à l'étape 3]]/chat-app</image>
      </to>
   </configuration>
</plugin>
```

Installez l'outil de ligne de commande gcloud: https://cloud.google.com/sdk/docs/install

Dans le répertoire backend, exécutez la commande suivante pour vous authentifiez à votre dépôt docker.

```
gcloud auth configure-docker northamerica-northeast1-docker.pkg.dev
```

Créer l'image docker avec la commande suivante. L'image devrait être téléchargée dans votre dépôt automatiquement.

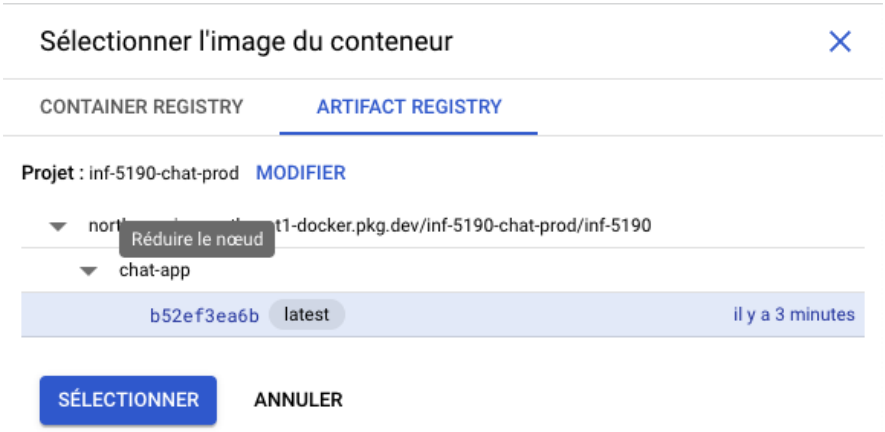
```
./mvnw clean compile jib:build
```

Étape 6 - Déploiement du backend

Sur la console cloud (https://console.cloud.google.com/) allez dans Cloud Run et activez l'API si ce n'est pas déjà fait.

Créez un service.

Sélectionnez l'image du conteneur à partir du **Artifact Registry**



Choisissez la région `northamerica-northeast1`.

Sélectionnez l'option pour allouer le CPU seulement sur le traitement d'une requête.

Sélectionnez un minimum de 0 et un maximum de 1 instance.

Sélectionnez: `Autoriser tout le trafic`
Sélectionnez: `Autoriser les appels non authentifiés`

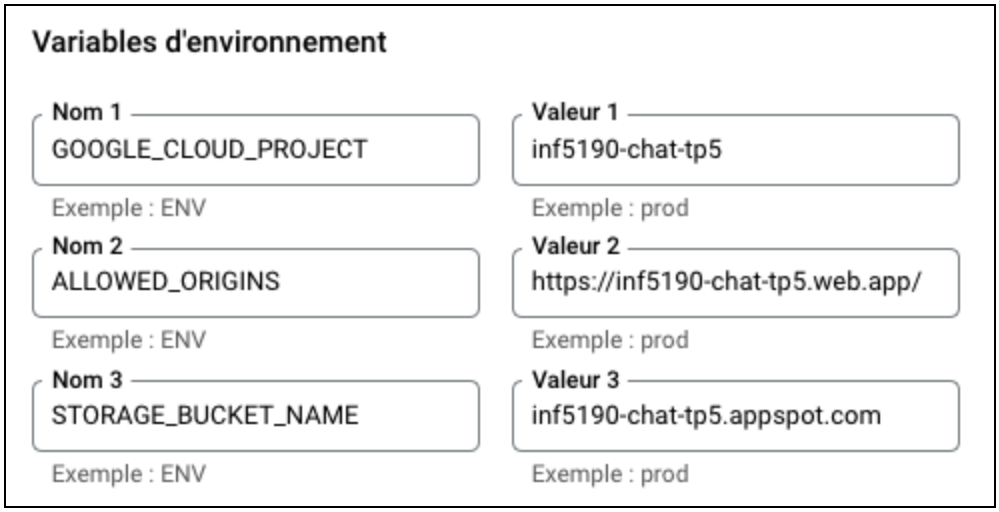
Ouvrez la section **`Conteneur, Connexions, sécurité`**.

Ajoutez les trois variables d'environnement suivantes. Assurez-vous de mettre les bonnes valeurs pour votre projet.

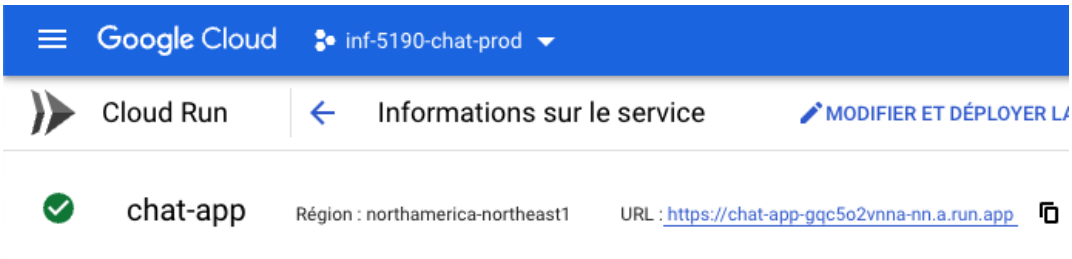
`GOOGLE_CLOUD_PROJECT`: l'id de votre projet

`ALLOWED_ORIGINS`: l'adresse de votre application web que vous trouverez dans la console firebase (https://console.firebase.google.com/) dans le service service Hosting dans la section Domains (prenez celle qui termine par .web.app).

`STORAGE_BUCKET_NAME`: le nom de votre bucket pour **Cloud Storage**.



Enfin, prenez note de l'adresse de votre backend.



Étape 7 - Déploiement du frontend

Dans le répertoire frontend, exécutez

```
firebase init
```

Sélectionnez:

```
Hosting: Configure files for Firebase Hosting and (optionally) set up
GitHub Action deploys
```

puis

`use existing project` et sélectionnez le bon projet.

Ensuite…

`What do you want to use as your public directory? (public)` **dist/app-chat**

`Configure as a single-page app (rewrite all urls to /index.html)? (y/N)` **y**

`Set up automatic builds and deploys with GitHub?` **No**

Modifiez votre fichier `environment.prod.ts` pour y ajouter (remarquez **https** et **wss**).

```
export const environment = {
 production: true,
 backendUrl: 'https://[[adresse backend]]',
 wsServer: 'wss://[[adresse backend]]',
};
```

Ensuite exécutez

```
ng build
```

et enfin

```
firebase deploy
```

Allez ensuite sur l'adresse de votre application web avec un navigateur pour valider que tout fonctionne bien.

N'hésitez pas à ajouter des journaux (logs) dans votre application et à la redéployer si jamais vous rencontrez des problèmes.

Partie 2 - Surveillance et métriques

Étape A - Tableau de bord

À l'aide de la vidéo suivante https://youtu.be/u146Qd0QG2o, créez un tableau de bord nommé INF-5190-APP-CHAT. Dans ce tableau de bord, ajoutez 3 graphiques.

Le premier montre le temps de réponse des requêtes **GET /messages** réussies. Affichez la moyenne et le 95e percentile (comme dans la vidéo).

Le deuxième montre la charge totale du système en requête par seconde. Utilisez un compteur comme type de métrique et le filtre suivant:

```
resource.type="cloud_run_revision"
logName="projects/<id de votre projet>/logs/run.googleapis.com%2Frequests"
```

Le troisième montre le taux d'erreur sur les opérations de connexion (login).

Étape B - Alerte

Comme dernière étape nous allons configurer une alerte si le taux d'erreur pour les opérations de connexion dépasse les 0.01 erreur par seconde.

Dans la section **Monitoring**, choisissez **Alertes**.

Appuyez sur **Create Policy**.

Choisissez la métrique d'erreur de connexion créée à l'étape A.

Sélectionnez une fenêtre de 1 minute et la fonction **`rate`**.

Ajoutez un seuil de 0.01 et configurez un canal de notification par mail avec votre adresse courriel.

Testez que votre alerte se déclenche si vous entrez le mauvais mot de passe plusieurs fois consécutivement dans un court laps de temps.

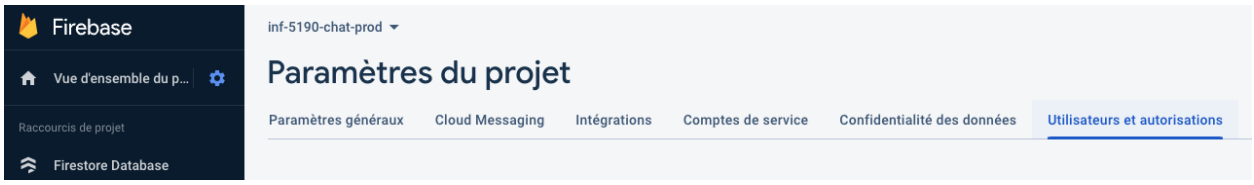
Remise

La date de remise pour ce TP est le 16 décembre à 23h59.

Ajoutez l'adresse de votre application web dans le fichier equipe.txt pour que je puisse me connecter.

Avant cette date limite, vous devrez créer une nouvelle distribution (release) avec le nom *tp5* le tag *tp5*.

N'oubliez pas d'ajouter trepanier.felix-etienne@uqam.ca comme `Lecteur` dans la section `Utilisateurs et autorisations` (accessible via le bouton engrenage) pour que je puisse valider votre tableau de bord.



Pondération

Partie 1

L'image **docker** est disponible dans l'**Artifact Repository** - 3pts
Le backend est déployé et accessible à l'aide du service **Cloud Run** - 3pts
Le frontend est déployé et accessible sur firebase **Hosting** - 3 pts
L'application est fonctionnelle - 4 pts

Partie 2

Chaque graphique dans le tableau de bord donne 2pts (pour un total de 6pts).
L'alerte vaut 1pt.