**Tp03 : Conteneurisation des Microservices avec Docker**

**Objectif**

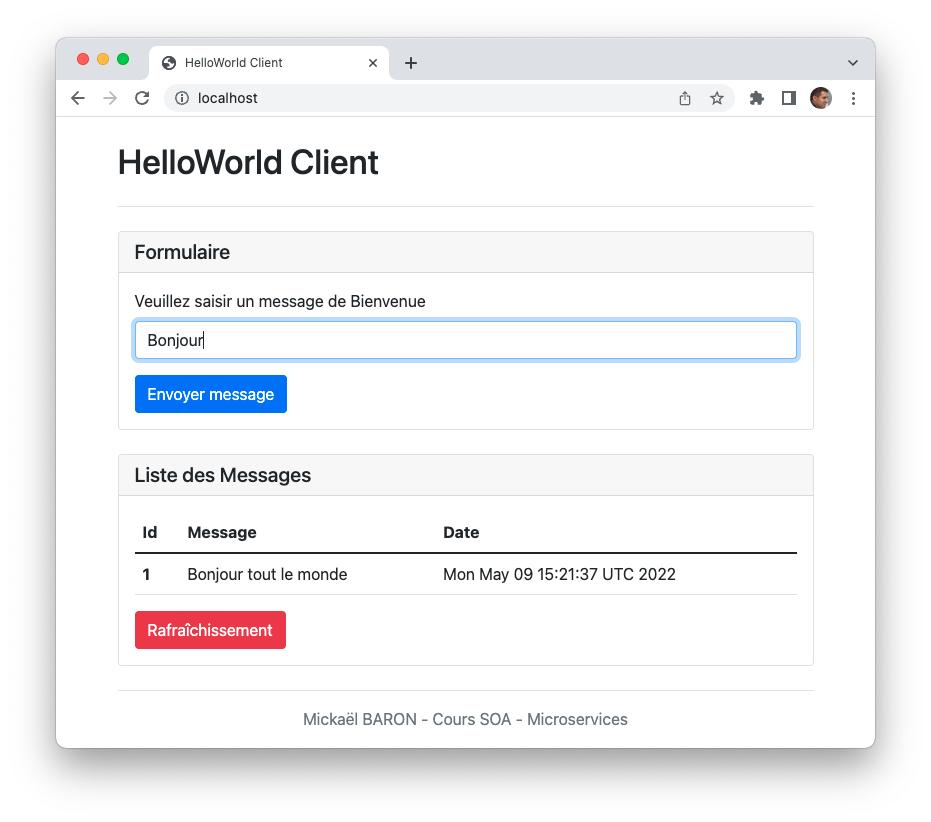
L'objectif de ce **TP** est d'apprendre à construire une application en respectant une architecture à base de microservices en se focalisant sur le langage Java.

Nous utiliserons pour cela plusieurs technologies :

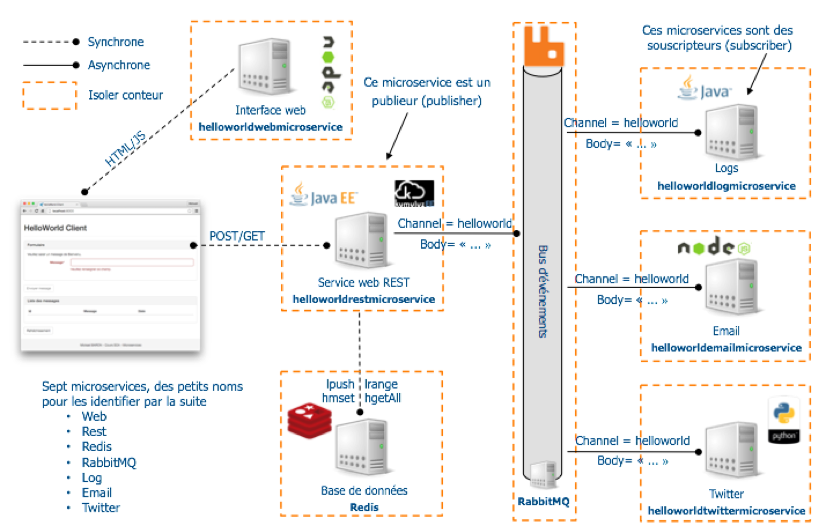
* la bibliothèque ***KumuluzEE*** pour packager et exécuter une application microservice respectant la spécification MicroProfile ;
* l'outil ***Docker*** pour l'isolation des microservices ;
* la bibliothèque et le serveur***RabbitMQ***pour la gestion d'un bus d'événements afin que les microservices communiquent de manière ***asyncrhone*** ;
* l'outil **Docker Compose** pour la composition des microservices.

## **Prérequis logiciels : Java, Maven, Eclipse, cURL, Docker desktop.**

## **Présentation de l'étude de cas :** L'étude de cas utilisée est une application permettant de diffuser des messages «***HelloWorld*** ». Elle fournit une interface web pour la saisie et pour la consultation des messages (voir figure ci-dessous).



Sur le schéma proposé sur la figure ci-dessous, nous détaillons la décomposition en microservices de cette application. L'architecture dispose de sept microservices.

* Le microservice **Web** (contenu dans le projet *helloworldwebmicroservice*) fournit à l'utilisateur une interface web. La technologie utilisée sera du HTML/JavaScript pour le client et NodeJS pour créer un petit serveur web.
* Le microservice **Rest** (contenu dans le projet *helloworldrestmicroservice*) a pour rôle de fournir une API de type service web pour le microservice **Web**. Lors de la réception d'un message « HelloWorld » celui-ci est envoyé au microservice **Redis** à des fins de stockage. Il publie également un événement vers le microservice **RabbitMQ**.
* Le microservice **Redis** fournit un serveur Redis pour le stockage des messages « HelloWorld ».
* Le microservice **RabbitMQ** fournit un bus d'événements basé sur RabbitMQ.
* Le microservice **Log** (contenu dans le projet *helloworldlogmicroservice*) s'abonne au bus d'événement et affiche sur la sortie console les événements envoyés.
* Les microservices **Email** et **Twitter** s'abonnent au bus d'événement et envoient respectivement un email ou un « Tweet ». Ces deux microservices ne seront pas traités 

**Etapes :**

**1-Préparer le programme Java du SW HelloWorld (Rest)**

**2 - Préparer le Microservice Redis (créer un conteneur à partir d'une image existante)**

**3 - Tester le service web HelloWorld (Rest) avec le Microservice Redis**

**4 - Préparer le Microservice Rest (écrire un Dockerfile et créer sa propre image Docker)**

**5 - Lier les Microservices Redis et Rest**

**6 - Emettre et recevoir des événements**

**7- Composer tous les Microservices avec DockerCompose**

**Référence** : <https://github.com/mickaelbaron/microservices-docker-java-tutorial>

**1-Préparer le programme Java du SW HelloWorld (Microservice Rest)**

Le code du projet Java est assez commun. Un package **service** pour la gestion des services web REST et un package **dao**pour la gestion des données avec la base de données [**Redis**](https://redis.io/). Dans cette partie, on va partir d'un code déjà tout prêt et nous allons nous attacher à le configurer pour l'utiliser avec [KumuluzEE](https://ee.kumuluz.com/).

## Étapes à suivre

* Démarrer l'environnement de développement Eclipse.
* Importer le projet Maven *helloworldrestmicroservice* (**File -> Import -> General -> Existing Maven Projects**, choisir le répertoire du projet puis faire **Finish**).
* Ouvrir le fichier ***pom.xml*** et compléter le contenu de la balise <**dependencies**> par les dépendances suivantes :

...

**<dependencies>**

<dependency>

<groupId>com.kumuluz.ee</groupId>

<artifactId>kumuluzee-core</artifactId>

<version>${kumuluzee.version}</version>

</dependency>

<dependency>

<groupId>com.kumuluz.ee</groupId>

<artifactId>kumuluzee-servlet-jetty</artifactId>

<version>${kumuluzee.version}</version>

</dependency>

<dependency>

<groupId>com.kumuluz.ee</groupId>

<artifactId>kumuluzee-jax-rs-jersey</artifactId>

<version>${kumuluzee.version}</version>

</dependency>

<dependency>

<groupId>com.kumuluz.ee</groupId>

<artifactId>kumuluzee-cdi-weld</artifactId>

<version>${kumuluzee.version}</version>

</dependency>

...

* Compléter dans la balise **<properties>**le numéro de version de KumuluzEE

...

<properties>

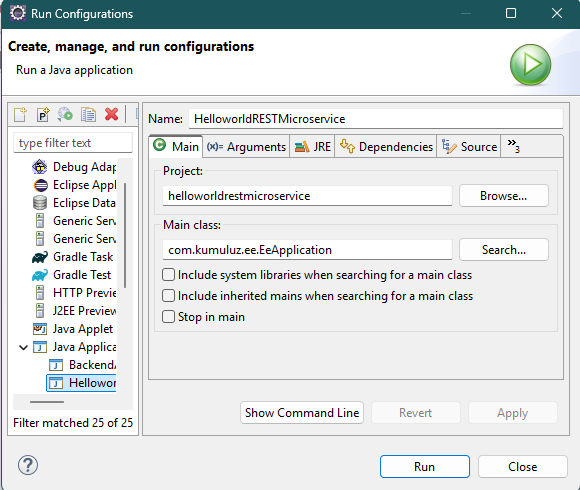
<kumuluzee.version>4.0.0</kumuluzee.version>

...

</properties>

...

* Pour exécuter le projet depuis Eclipse, créer une configuration d'exécution que vous appellerez HelloworldRESTMicroservice et dont la classe principale (Main class) sera com.kumuluz.ee.EeApplication puis faire **Run**.



* Pour tester nous pourrions utiliser l'adresse <http://localhost:8080/helloworld>
* Ouvrir le fichier ***pom.xml*** et ajouter dans la balise ***<plugins>*** le plugin ***maven-dependency-plugin*** comme montré sur le code suivant.

...

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<version>${maven-dependency-plugin.version}</version>

<executions>

<execution>

<id>copy-dependencies</id>

<phase>package</phase>

<goals>

<goal>copy-dependencies</goal>

</goals>

</execution>

</executions>

</plugin>

* Depuis Eclipse, exécuter la configuration d'exécution appelée **all dependencies (package)**
* Ouvrir une invite de commande à la racine du projet *helloworldrestmicroservice*, puis exécuter la ligne de commande suivante :
* java -cp 'target/classes:target/dependency/\*' com.kumuluz.ee.EeApplication

**2 - Préparer le microservice Redis (créer un conteneur à partir d'une image existante)**

Le microservice **Redis** a pour objectif de conserver l'état des différents messages *HelloWorld* traités par le microservice **Rest**.

**Étapes à suivre**

* Ouvrir une invite de commande et se placer à la racine du dossier ***workspace*** Et saisir la ligne de commande suivante pour télécharger la dernière version de l'image Docker *Redis (****nécessite internet****)* :

**docker pull redis**

* S'assurer que l'image a été correctement téléchargée en utilisant la commande :

**docker images**

* Se placer à la racine du répertoire **workspace** et créer un conteneur à partir de l'image [Docker](https://www.docker.com/) de [Redis](https://redis.io/) en saisissant la commande suivante :

**docker ps**

**3 - Tester le service web HelloWorld (Rest) avec le microservice Redis**

Pour assurer la **communication** entre ce projet **Rest** (pas encore isolé dans un conteneur Docker) et le microservice **Redis** la solution est de rediriger le port **6379** de l'hôte vers le port **6379** du conteneur (le port par défaut de Redis).

**Étapes à suivre**

* Supprimer le conteneur nommé ***redis*** via la ligne de commande suivante.

**docker rm -f redis**

* Créer une nouvelle fois le conteneur ***redis***en ajoutant le paramètre **-p** **6379:6379** permettant de rediriger le port **6379** de l'hôte vers le port **6379** du conteneur.

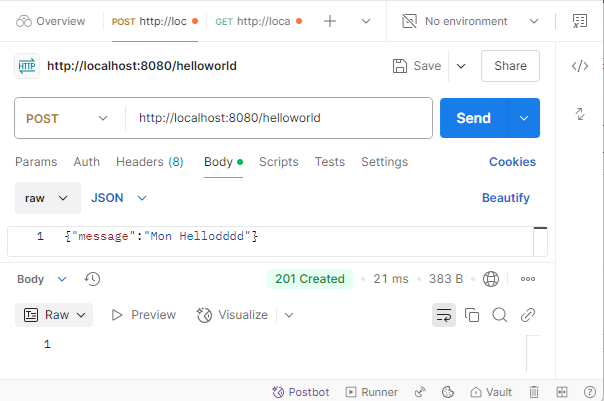
**docker run --name redis -p 6379:6379 -d redis redis-server --appendonly yes**

* Vérifier que le conteneur a été créé avec la commande : **docker ps**
* Pour obtenir plus d'information sur le conteneur nommé redis exécuter la commande:

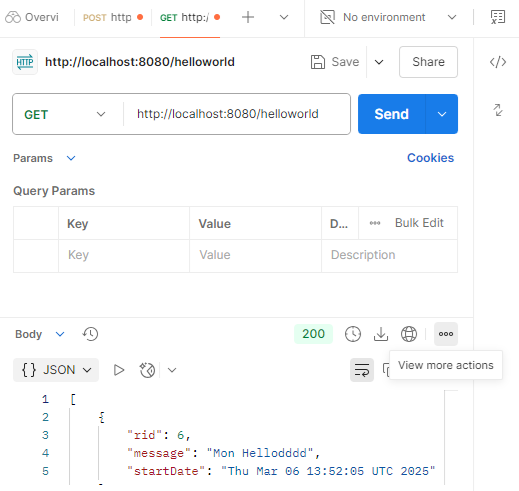
**docker inspect redis**

* Depuis la configuration d'exécution (**eclipse**), ajouter une variable d'environnement appelée (onglet Environment) **REDIS\_HOST** avec la valeur **tcp://0.0.0.0:6379**, puis faire **Run**.
* Pour tester le service web HelloWorld, nous utiliserons l'outil **Postman**

**Envoyer un message :**



**Recevoir un message :**

****

**4 - Préparer le microservice Rest (écrire un Dockerfile et créer sa propre image Docker)**

## **Étapes à suivre**

 Créer un fichier **Dockerfile**à la racine du projet **helloworldrestmicroservice**.

 Ouvrir un éditeur de texte et saisir le contenu présenté ci-dessous.

***FROM openjdk:11-slim***

***LABEL MAINTAINER="Mickael BARON"***

***ENV MAVEN\_VERSION 3.8.5***

***RUN apt-get update -y***

***RUN apt-get install -y curl***

***RUN curl -fsSLk https://dlcdn.apache.org/maven/maven-3/3.8.8/binaries/apache-maven-3.8.8-bin.tar.gz | tar xzf - -C /usr/share \***

***&& mv /usr/share/apache-maven-3.8.8 /usr/share/maven \***

***&& ln -s /usr/share/maven/bin/mvn /usr/bin/mvn***

***ENV MAVEN\_HOME /usr/share/maven***

***ADD pom.xml /work/pom.xml***

***WORKDIR /work***

***RUN ["mvn", "dependency:go-offline"]***

***ADD ["src", "/work/src"]***

***RUN ["mvn", "package"]***

***EXPOSE 8080***

***ENTRYPOINT ["java", "-cp", "target/classes:target/dependency/\*", "com.kumuluz.ee.EeApplication"]***

* Nous allons construire l'image à partir du fichier **Dockerfile**, exécuter la ligne de commande suivante (**nécessite internet).**

**docker build -t mickaelbaron/helloworldrestmicroservice .**

* S'assurer que l'image a été correctement construite en exécutant :

**docker images**

**5 - Lier les microservices Redis et Rest**

Nous allons nous intéresser dans cet partie à créer le conteneur du microservice **Rest** et lui associer le conteneur du microservice **Redis**. Pour réaliser cette association, nous utilisons des **réseaux Docker**.

## **Étapes à suivre**

* Créer un **réseau** Docker appelé **helloworldnetwork** en exécutant la ligne de commande suivante dans la racine de dossier workspace.

**docker network create helloworldnetwork**

* Afficher la liste des réseaux Docker avec la commande : **docker network ls**
* Nous allons connecter le conteneur ***redis***précédemment créé au réseau Docker ***helloworldnetwork***.

**docker network connect helloworldnetwork redis**

* Pour vérifier que le conteneur Redis est bien connecté au réseau Docker helloworldnetwork deux (2) solutions sont possibles :

**docker inspect redis**

ou bien

**docker network inspect helloworldnetwork**

* Créer un conteneur pour le microservice **Rest** en exécutant la ligne de commande suivante.

***docker*** *run --name rest -p 8080:8080 -d --network helloworldnetwork --env* ***REDIS\_HOST****=tcp://redis:6379 mickaelbaron/helloworldrestmicroservice*

* Assurons-nous que les deux conteneurs ***redis*** et***rest*** sont connectés dans le **réseau** Docker ***helloworldnetwork***

**docker network inspect helloworldnetwork**

* Il nous reste plus qu'à tester le service **web** contenu dans le conteneur ***rest***. Essayer de poster un message « **HelloWorld** » et de récupérer les messages « **HelloWorld** » envoyés avec **Postman.**

**6 - Emettre et recevoir des événements**

Nous allons créer un nouveau microservice appelé **Log** (contenu dans leprojet helloworldlogmicroservice) qui se chargera de réceptionner les événements envoyés au bus d'événements de RabbitMQ.

## **Étapes à suivre**

* Avant de continuer, nous allons arrêter et supprimer les conteneurs *rest* et *redis*. Exécuter les lignes de commande suivantes.

**docker rm -f rest**

**docker rm -f redis**

* Depuis l'environnement de développement Eclipse, modifier la classe HelloWorldResource pour ajouter l'attribut currentProducer et le contenu dans la méthode addHelloWorld.

@Path("/helloworld")

@Produces(MediaType.APPLICATION\_JSON)

@Consumes(MediaType.APPLICATION\_JSON)

public class HelloWorldResource {

@Inject

@Named("redis")

private IHelloWorldDAO currentDAO;

@Inject

private IHelloWorldEventProducer currentProducer;

@GET

public Response getHelloWorlds() {

return Response.ok(currentDAO.getHelloWorlds()).build();

}

@POST

public Response addHelloWorld(HelloWorld newHelloWorld) {

if (newHelloWorld != null) {

newHelloWorld.setStartDate(new Date().toString());

}

currentDAO.addHelloWorld(newHelloWorld);

currentProducer.sendMessage(newHelloWorld);

return Response.status(Status.CREATED).build();

}

}

* Mettre à jour l'image du microservice **Rest** défini dans leprojet helloworldrestmicroservice, se placer à la racine du projet helloworldrestmicroservice et exécuter la ligne de commande suivante

**docker build -t mickaelbaron/helloworldrestmicroservice .**

* Importer le projet Maven *helloworldlogmicroservice* (**File -> Import -> General -> Existing Maven Projects**, choisir le répertoire du projet puis faire **Finish**).
* Examiner la classe **HelloWorldLogMicroservice.** Les événements reçus sont récupérés et affichés sur la sortie console.
* Créer un fichier ***Dockerfile***à la racine du projet *helloworldlogmicroservice*.
* Ouvrir un éditeur de texte et saisir le contenu présenté ci-dessous :

# Build env

FROM maven:3-jdk-11-slim AS build-java-stage

LABEL MAINTAINER="Mickael BARON"

ADD pom.xml /work/pom.xml

WORKDIR /work

RUN ["mvn", "dependency:go-offline"]

ADD ["src", "/work/src"]

RUN ["mvn", "package"]

# Run env

FROM openjdk:11-jre-slim

COPY --from=build-java-stage /work/target/classes /classes/

COPY --from=build-java-stage /work/target/dependency/\*.jar /dependency/

ENTRYPOINT ["java", "-cp", "classes:dependency/\*", "fr.mickaelbaron.helloworldlogmicroservice.HelloWorldLogMicroservice"]

CMD [localhost]

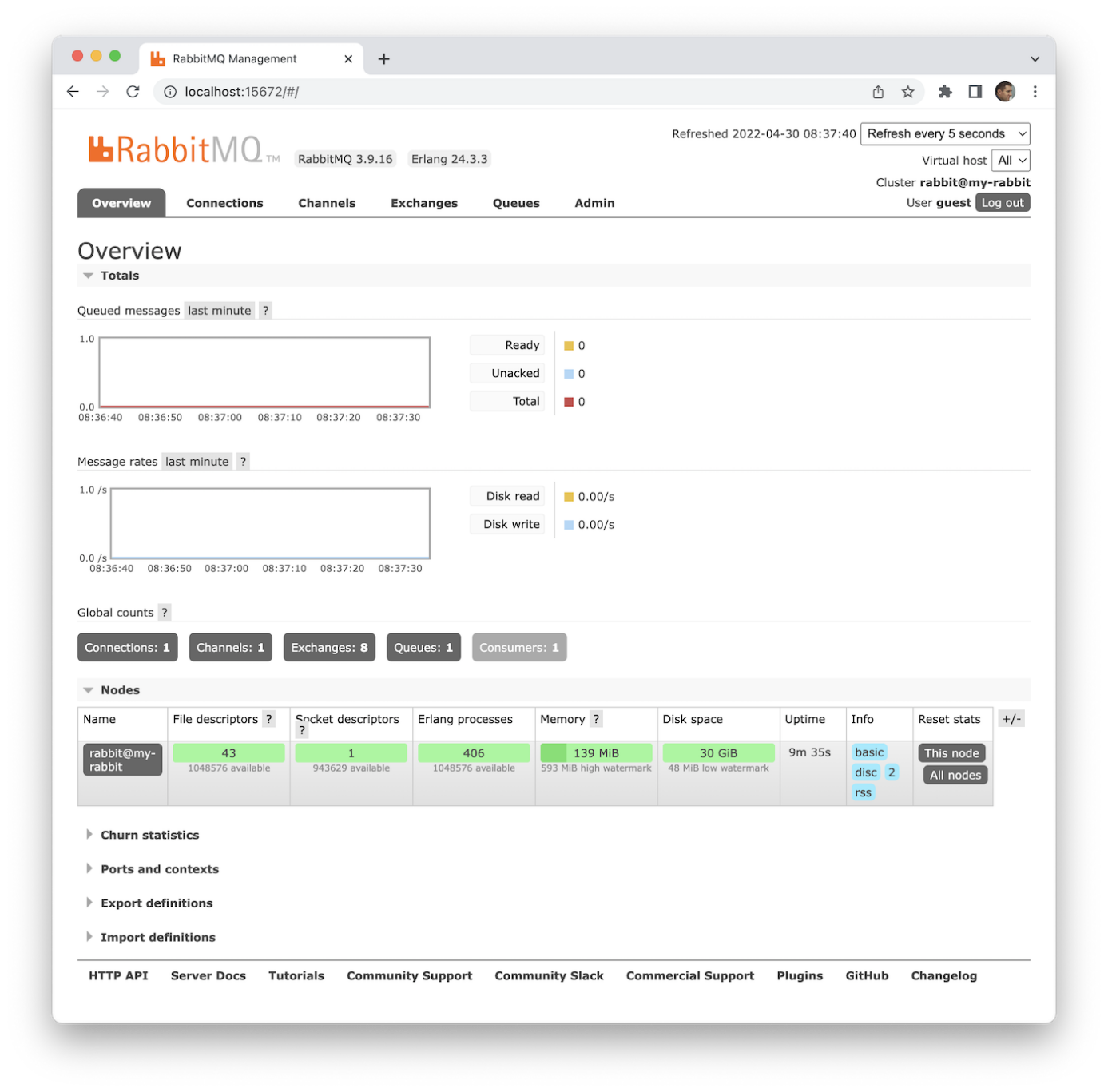
* Nous allons construire l'image à partir de fichier **Dockerfile**, exécuter la ligne de commande suivante depuis la racine du projet **helloworldlogmicroservice.**

**docker build -t mickaelbaron/helloworldlogmicroservice .**

* Nous allons installer le serveur [RabbitMQ](https://www.rabbitmq.com/) en l'isolant dans un conteneur. Le microservice résultat s'appellera **Rabbitmq**. Nous utiliserons une image contenant une interface web pour la gestion des événements reçus et envoyés (rabbitmq:management).

**docker pull rabbitmq:management**

* Il ne nous reste plus qu'à créer tous les conteneurs **rest**, **redis** et **rabbitmq** et de les connecter au réseau Docker***helloworldnetwork***. Exécuter les lignes de commande suivantes en faisant attention d'être à la racine du répertoire ***workspace***.
* **docker run --name redis -d --network helloworldnetwork redis redis-server --appendonly yes**
* **docker run --name rabbitmq -d --network helloworldnetwork -p 5672:5672 -p 15672:15672 --hostname my-rabbit rabbitmq:management**
* **docker run --name log -d --network helloworldnetwork mickaelbaron/helloworldlogmicroservice rabbitmq**
* **docker run --name rest -d --network helloworldnetwork -p 8080:8080 --env REDIS\_HOST=tcp://redis:6379 --env RABBITMQ\_HOST=rabbitmq mickaelbaron/helloworldrestmicroservice**
* Assurons-nous que tous les conteneurs soient opérationnels en affichant le statut des conteneurs. **docker ps**
* Assurons-nous également que l'interface d'administration de RabbitMQ fonctionne. Ouvrir un navigateur web et saisir l'adresse (utilisateur : **guest**, mot de passe : **guest**) [http://localhost:15672](http://localhost:15672/).



* Appeler le service web***HelloWorld***pour tester la chaîne complète des microservices en envoyant des messages post avec **Postman.**
* Afficher le contenu des logs du conteneur *log* en exécutant la commande suivante.

**docker logs log**

**7- Composer tous les Microservices avec DockerCompose**

Nous employons la commande **compose** de l'outil **docker** lorsqu'il y a plus de deux conteneurs. Nous allons également introduire un nouveau **microservice** qui se chargera de fournir une interface **web** à notre application : le microservice **Web**. Le code de cette application est dans le projet **helloworldwebmicroservice**. Le contenu du code est réalisé en HTML et [Vue.js](https://vuejs.org/).

## **Étapes à suivre**

* Avant de commencer faire « table rase » en supprimant tous les conteneurs précédemment créés, exécuter la ligne de commande suivante

docker rm -f $(docker ps -q)

* Parcourir les fichiers contenus dans le projet helloworldwebmicroservice. Vous remarquerez dans le fichier App.vue l'URL du microservice **Rest**. Dans notre cas il s'agit de l'URL <http://localhost:8080/helloworld>.
* Créer un fichier **docker-compose.yml** à la racine du répertoire **workspace**
* Éditer le fichier et le compléter comme ci-dessous

services:

redis:

image: redis:latest

command: redis-server --appendonly yes

volumes:

- ./data:/data

networks:

- helloworldnet

rabbitmq:

image: rabbitmq:management

hostname: my-rabbit

ports:

- 5672:5672

- 15672:15672

networks:

- helloworldnet

rest:

build:

context: helloworldrestmicroservice/

dockerfile: DockerfileMSB

image: mickaelbaron/helloworldrestmicroservice:latest

depends\_on:

- rabbitmq

- redis

ports:

- 8080:8080

environment:

REDIS\_HOST: tcp://redis:6379

RABBITMQ\_HOST: rabbitmq

networks:

- helloworldnet

web:

build: helloworldwebmicroservice

image: mickaelbaron/helloworldwebmicroservice:latest

ports:

- 80:80

networks:

- helloworldnet

log:

build: helloworldlogmicroservice/

image: mickaelbaron/helloworldlogmicroservice:latest

depends\_on:

- rabbitmq

command: rabbitmq

networks:

- helloworldnet

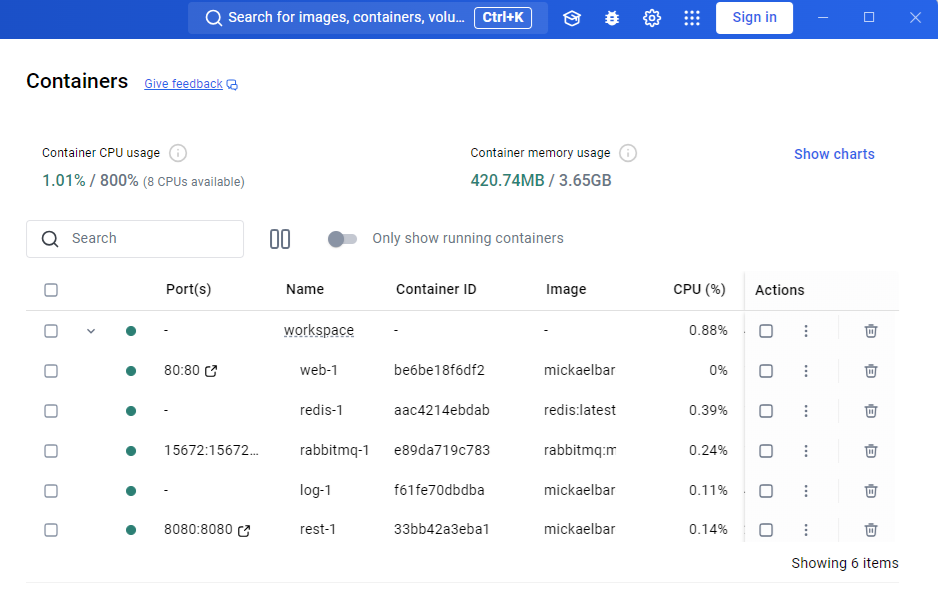
networks:

helloworldnet:

name: helloworldnetwork

* Nous pouvons donc exécuter ce fichier en utilisant la commande  **docker compose up -d** comme précisé ci-dessous (s'assurer d'être à la racine du répertoire workspace)

**docker compose up -d**



* Afficher les logs des conteneurs en exécutant la ligne de commande suivante.

**docker compose logs**

* Vérifier que tous les conteneurs ont été correctement créés en exécutant la ligne commande suivante**. docker compose ps**
* Ouvrir un navigateur et rendez-vous à cette adresse : [http://localhost](http://localhost/)