Tutorial Quarkus

Quarkus Facile : Episode 1 - Introduction

https://youtu.be/JfEnYZoJu1s

Créer un projet quarkus :

Intaller l'extension quarkus sur vscode. Ctrl + Shift + p → Créer un nouveau projet quarkus → suivre les étapes d'installation en sélectionnant l'outil de build maven .

Lancer un projet quarkus :

$> mvn quarkus:dev

$> curl localhost:8080/hello-world

Annotation d'un point d'accès REST :

@GET

Annotation d'une propriétée de configuration :

@ConfigProperty(name = "helloworld")
String helloworld;

fichier resources/application.properties
--> helloworld=hello-world

Annotation d'un endpoint :

@Path("/hello")

Quarkus Facile : Episode 2 - Compilation Native

https://youtu.be/8DKqfbwtYUM

Package to jar :

- Vérifier les fichiers java dans le dossier test

$> mvn clean package

- (sauvegarder dans /target)

Créer une application native :

Compilation en binaire.

graalVM (https://github.com/graalvm/graalvm-ce-builds/releases/tag/vm-20.2.0)

- Télécharger / dézip / $> ./gu install native-image (récup l'image nécessaire en fct de votre plateforme).

$> export GRAALVM_HOME=/home/…

$> mvn clean package -P native (P = profil maven)

Executable dans /target

Environ 16 fois plus rapide à l'exécution et 5 fois moins de mémoire utilisée.

(0.587s à 0.037s).

Quarkus Facile : Episode 3 – Rest Client

https://youtu.be/yIcYlR6HXls

Extensions : REST Client & RESTEasy JAX-RS

Créer une interface REST Client :

```
@RegisterRestClient(baseUri = « http://localhost:8081 »)
@Path(« /hello »)
public interface RemoteService() {

        @GET
        @Produces(MediaType.TEXT_PLAIN)
        public string getRemoteHello();

}
```

Injecter le service et appeler le endpoint distant :

```
@RestClient
    RemoteService remoteService;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return remoteService.getRemoteHello();
    }
```

Créer une nouvelle application quarkus (côté API REST):

Spécifier l'utilisation du port 8081 :

```
quarkus.http.port=8081
```

Lancer le projet :

$> mvn quarkus:dev

$> curl http://localhost:8080/hello

Quarkus Facile : Episode 4 – Fault Tolerance

https://youtu.be/De_ILrCQYVc

Extensions : Fault

Relancer l'exécution en cas d'erreurs :

```
@Retry (
        maxRetries = 3,
        delay = 2000
)
```

Retour s'il y a une erreur :

```
@Fallback(
        fallbackMethod = « fallback »
)

Private String fallback()  {
        return « hello from feedback » ;
}
```

CircuitBraker (Retry mais avec ouverture du circuit) :

```
@CircuitBraker(
        requestVolumeThreshhold = 4, // sur un total de 4 appels
        failureRatio = 0.75, // si 75% des appels échouent (ici 3/4)
        delay = 5000 // attends 5s avant de refaire un appel
)
```

Après 4 appels avec erreur un CircuitBreakerOpenException est retourné.

Quarkus Facile : Episode 5 – Hibernate Panache

https://youtu.be/rFoKfL5FhU0

Extensions : Hibernate ORM with Panache / JDBC Driver – H2 / RESTEasy JSON-B / SmallRye OpenAPI
But : enregistrer des épisodes.
Nom de la classe principale : EpisodeRessource.java

Créer une classe Episode :

```java
package main.java.org.matbra;

import javax.persistence.Entity;
import io.quarkus.hibernate.orm.panache.PanacheEntity;

@Entity
public class Episode extends PanacheEntity {
    public String title;
    public String description;
}
```

Les variables public title et description sont automatiquement transformées en prop get ; set ; car la classe episode est une sous-classe de PanacheEntity.

Lister tous les épisodes :

```java
@Path("/episodes")
public class EpisodeResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Episode> episodes() {
        return Episode.listAll();
    }
}
```

Configurer la BDD java en mémoire :

```
quarkus.datasource.jdbc.url=jdbc:h2:mem:default
quarkus.datasource.db-kind=h2
quarkus.hibernate-orm.database.generation=drop-and-create
```

drop-and-create : quand on régénére l'application la BDD est drop puis recréée.
h2 : SGBD java.
hibernate-orm : Framework gérant la persistance des objets en BDDR.

Permettre l'enregistrement de données :

```
@POST
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    @Transactional
    public Response saveEpisode(Episode episode) {
        episode.persist();
        return Response.status(Status.CREATED).entity(episode).build();
    }
```

Aller sur la page localhost:8080/q/swagger-ui pour tester.

Trier les épisodes avec paramètre :

```
 @GET
    @Path("/{episode}")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Episode> episodes(@PathParam("episode") String episode) {
        if (episode != null) {
            return Episode.findByEpisode(episode);
        }
        return Episode.listAll();
    }
```

```
public static List<Episode> findByEpisode(String episode) {

        return find("title", episode).list();
    }
```

Services

```
@ApplicationScoped

public class EpisodeService
```

```
@Inject

    EpisodeService episodeService;
```

Quarkus Facile : Episode 6 – Hibernate Panache suite …

https://youtu.be/CZiLWJrurkc

Créer un fichier import.sql dans les resources avec les inserts nécessaires au lancement de l'app.
INSERT INTO Episodes…

Quarkus Facile : Episode 7 – Docker (& JIB)

https://youtu.be/oKnJGI-j1W8

docker/Dockerfile.native : pour applis native
docker/Dockerfile.fast-jar : jar avec fast-jar
docker/Dockerfile.jvm : classique

faire :

$> mvn clean package

Avant de build l'image docker. (Commandes présentes dans les fichiers Dockerfile).

Quarkus Facile : Episode 8 – HealthChecks

https://youtu.be/7BcDF-b6fo4

Extensions : SmallRye Health

$> curl localhost:8080/health
$> curl localhost:8080/health/ready
$> curl localhost:8080/health/live
…

```
@Liveness / @Readynes / …
public class LivenessProbe {
        @Override
        public HealthCheckResponse call() {
                return HealthCheckResponse.up("I am alive !");
        }
}
```

Quarkus Facile : Episode 9 – Déployer son application sur Kubernetes

https://youtu.be/FgMxOxqlKIo

Extensions : Kubernetes

Changer des paramètres lors de la construction du container :

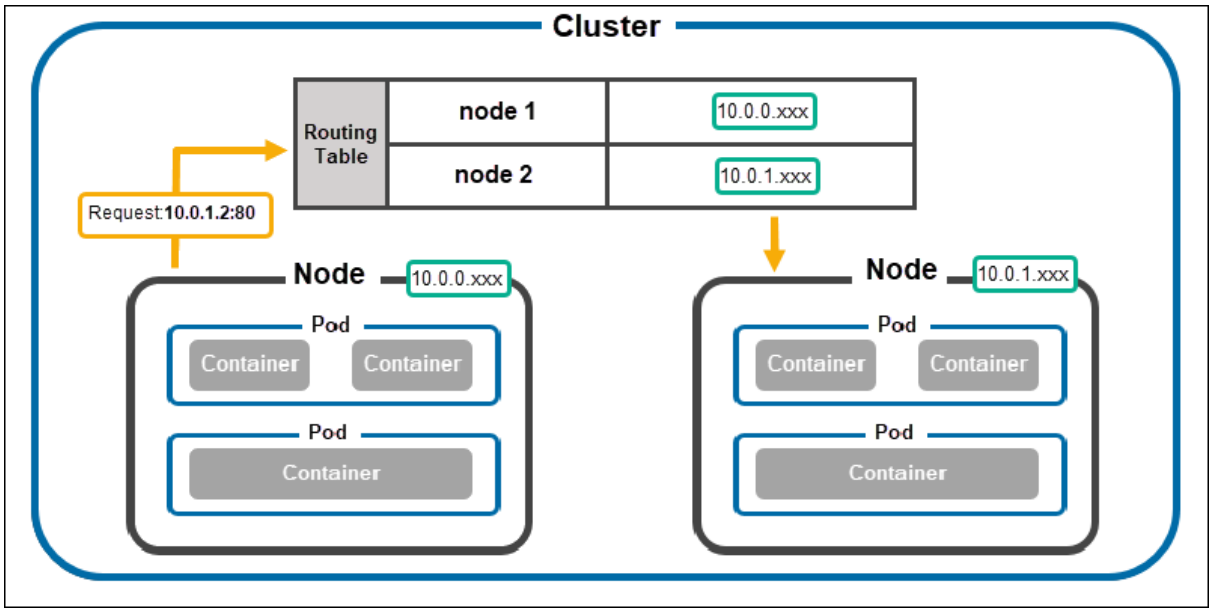
quarkus.container-image.group=matbra // change le nom d'utilisateur de container
quarkus.container-image.tag=v1 // change le tag de l'image
kubernetes.service-type=NodePort // Changer le type de service du cluster (de base ClusterIP, ici NodePort = ouvre un port du nœud)

$> mvn clean package –Dquarkus.container-image.push=true

1. Construit le jar
2. Construit l'image
3. Push l'image vers le registery Docker hub

Pods :

Cluster > Serveur > Pod > Container > Image (un pod à une ip)



Quarkus Facile : Episode 11 – Tester son application

https://youtu.be/x_IIMkD3O-0

Extension : REST Client

Fichiers dans les dossier test.

```
@QuarkusTest
public class EpisodeResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/hello")
          .then()
             .statusCode(200)
             .body(is("Hello RESTEasy"));
    }

}
```

given() : quand tu donnes les données
when.get : quand tu vas visité l'endpoint hello
then : alors vérifie le statuscode et si on voit le texte « Hello RESTEasy » dans le body.

Tester l'application :

$> mvn test

Mocker (simuler) des tests :

Créer un REST Client.

Ajouter manuellement mockito au dépendences maven dans le pom.xml

```
<dependency>
   <groupId>io.quarkus</groupId>
   <artifactId>quarkus-junit5-mockito</artifactId>
   <scope>test</scope>
</dependency>
```

Puis injecter le service client rest et simuler l'appel de l'endpoint remoteHello sur le server rest en retournant la valeur « hello ».

```
@RestClient
    @InjectMock
    EpisodeRestClient episodeRestClient;

    @Test
    public void testHelloEndpoint() {
        Mockito.when(episodeRestClient.remoteHello()).thenReturn("hello");
```