# quarkus-workshop

Hero Microservice

### Bootstrapping the Hero REST Endpoint

```
cd quarkus-workshop-super-heroes/super-heroes
mvn io.quarkus:quarkus-maven-plugin:1.9.1.Final:create \
    -DprojectGroupId=io.quarkus.workshop.super-heroes \
    -DprojectArtifactId=rest-hero \
    -DclassName="io.quarkus.workshop.superheroes.hero.HeroResource" \
    -Dpath="api/heroes"
```

### Running the Application : Development Mode

```
cd rest-hero
./mvnw quarkus:dev
```

### Testing the Application

```
./mvnw test
```

### Packaging and Running the Application

```
./mvnw package
java -jar target/rest-hero-1.0-SNAPSHOT-runner.jar
```

### Transactions and ORM

The Hero API's role is to allow CRUD operations on Super Heroes.

In this module we will create a Hero entity and persist/update/delete/retrieve it from a Postgres database in a transactional way.

This microservice:

- interacts with a PostGreSQL database - so it needs a driver
- uses Hibernate with Panache - so need the dependency on it
- validates payloads and entities - so need a validator

- consumes and produces JSON - so we need a mapper

Hibernate ORM is the de-facto JPA implementation and offers you the full breadth of an Object Relational Mapper. It makes complex mappings possible, but it does not make simple and common mappings trivial. Hibernate ORM with Panache focuses on making your entities trivial and fun to write in Quarkus.

Because JPA and Bean Validation work well together, we will use Bean Validation to constrain our business model.

To add the required dependencies, just run the following command:

```
./mvnw quarkus:add-extension -Dextensions="jdbc-h2,jdbc-postgresql,hibernate-orm-panache,hibernate-validator,resteasy-jsonb"
```

Running the Infrastructure

```
docker-compose -f docker-compose-linux.yaml up -d
```

## Hero Entity

Hero.java

```
package io.quarkus.workshop.superheroes.hero;

import io.quarkus.hibernate.orm.panache.PanacheEntity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import java.util.Random;
@Entity
public class Hero extends PanacheEntity {

    @NotNull
    @Size(min = 3, max = 50)
    public String name;
    public String otherName;
    @NotNull
    @Min(1)
    public int level;
    public String picture;

    @Column(columnDefinition = "TEXT")
    public String powers;


    @Override
    public String toString() {
        return "Hero{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", otherName='" + otherName + '\'' +
            ", level=" + level +
            ", picture='" + picture + '\'' +
            ", powers='" + powers + '\'' +
```

```
            '}';
    }
}
```

```
// creating a hero
Hero hero = new Hero();
hero.name = "Superman";
hero.level = 9;

// persist it
hero.persist();

// getting a list of all Hero entities
List<Hero> heroes = Hero.listAll();

// finding a specific hero by ID
hero = Hero.findById(id);

// counting all heroes
long countAll = Hero.count();
```

```
public static Hero findRandom() {
    long countHeroes = Hero.count();
    Random random = new Random();
    int randomHero = random.nextInt((int) countHeroes);
    return Hero.findAll().page(randomHero, 1).firstResult();
}
```

### Configuring Hibernate

```
quarkus.hibernate-orm.database.generation=drop-and-create
quarkus.hibernate-orm.log.sql=true
```

### HeroService Transactional Service

HeroService.java

```
package io.quarkus.workshop.superheroes.hero;

import javax.enterprise.context.ApplicationScoped;
import javax.transaction.Transactional;
import javax.validation.Valid;
import java.util.List;

import static javax.transaction.Transactional.TxType.REQUIRED;
import static javax.transaction.Transactional.TxType.SUPPORTS;

@ApplicationScoped
@Transactional(REQUIRED)
public class HeroService {


    @Transactional(SUPPORTS)
    public List<Hero> findAllHeroes() {
        return Hero.listAll();
    }

    @Transactional(SUPPORTS)
```

```
    public Hero findHeroById(Long id) {
        return Hero.findById(id);
    }

    @Transactional(SUPPORTS)
    public Hero findRandomHero() {
        Hero randomHero = null;
        while (randomHero == null) {
            randomHero = Hero.findRandom();
        }
        return randomHero;
    }

    public Hero persistHero(@Valid Hero hero) {
        Hero.persist(hero);
        return hero;
    }

    public Hero updateHero(@Valid Hero hero) {
        Hero entity = Hero.findById(hero.id);
        entity.name = hero.name;
        entity.otherName = hero.otherName;
        entity.level = hero.level;
        entity.picture = hero.picture;
        entity.powers = hero.powers;
        return entity;
    }

    public void deleteHero(Long id) {
        Hero hero = Hero.findById(id);
        hero.delete();
    }
}
```

## Configuring the Datasource

### H2

```
%dev.quarkus.datasource.db-kind=h2
%dev.quarkus.datasource.username=username-default
%dev.quarkus.datasource.jdbc.url=jdbc:h2:mem:heroes_database
%dev.quarkus.datasource.jdbc.max-size=13
```

### postgresql

```
quarkus.datasource.url=jdbc:postgresql://localhost:5432/heroes_database
quarkus.datasource.driver=org.postgresql.Driver
quarkus.datasource.username=superman
quarkus.datasource.password=superman
quarkus.datasource.max-size=8
quarkus.datasource.min-size=2
```

## HeroResource Endpoint

### HeroResource.java

```
package io.quarkus.workshop.superheroes.hero;

import org.jboss.logging.Logger;
```

```
import javax.inject.Inject;
import javax.validation.Valid;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.net.URI;
import java.util.List;

import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static javax.ws.rs.core.MediaType.TEXT_PLAIN;

@Path("/api/heroes")
@Produces(APPLICATION_JSON)
public class HeroResource {

    private static final Logger LOGGER = Logger.getLogger(HeroResource.class);

    @Inject
    HeroService service;

    @GET
    @Path("/random")
    public Response getRandomHero() {
        Hero hero = service.findRandomHero();
        LOGGER.debug("Found random hero " + hero);
        return Response.ok(hero).build();
    }

    @GET
    public Response getAllHeroes() {
        List<Hero> heroes = service.findAllHeroes();
        LOGGER.debug("Total number of heroes " + heroes);
        return Response.ok(heroes).build();
    }

    @GET
    @Path("/{id}")
    public Response getHero(
        @PathParam("id") Long id) {
        Hero hero = service.findHeroById(id);
        if (hero != null) {
            LOGGER.debug("Found hero " + hero);
            return Response.ok(hero).build();
        } else {
            LOGGER.debug("No hero found with id " + id);
            return Response.noContent().build();
        }
    }

    @POST
    public Response createHero(
        @Valid Hero hero, @Context UriInfo uriInfo) {
        hero = service.persistHero(hero);
        UriBuilder builder = uriInfo.getAbsolutePathBuilder().path(Long.toString(hero.id));
        LOGGER.debug("New hero created with URI " + builder.build().toString());
        return Response.created(builder.build()).build();
    }

    @PUT
    public Response updateHero(
        @Valid Hero hero) {
        hero = service.updateHero(hero);
        LOGGER.debug("Hero updated with new valued " + hero);
        return Response.ok(hero).build();
    }

    @DELETE
    @Path("/{id}")
    public Response deleteHero(
        @PathParam("id") Long id) {
        service.deleteHero(id);
        LOGGER.debug("Hero deleted with " + id);
        return Response.noContent().build();
    }

    @GET
    @Produces(TEXT_PLAIN)
    @Path("/hello")
    public String hello() {
        return "hello";
    }
}
```

## Adding Data

```
src/main/resources/import.sql
```

```
./mvnw quarkus:dev
```

```
curl http://localhost:8080/api/heroes/
curl http://localhost:8080/api/heroes/random
```

## CRUD Tests

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>1.12.2</version>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>postgresql</artifactId>
    <version>1.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <scope>test</scope>
</dependency>
```

DatabaseResource.java

```
package io.quarkus.workshop.superheroes.hero;

import io.quarkus.test.common.QuarkusTestResourceLifecycleManager;
import org.testcontainers.containers.PostgreSQLContainer;

import java.util.Collections;
import java.util.Map;

public class DatabaseResource implements QuarkusTestResourceLifecycleManager {

    private static final PostgreSQLContainer DATABASE = new PostgreSQLContainer<>("postgres:10.5")
        .withDatabaseName("heroes_database")
        .withUsername("superman")
        .withPassword("superman")
        .withExposedPorts(5432);

    @Override
    public Map<String, String> start() {
```

```
        DATABASE.start();
        return Collections.singletonMap("quarkus.datasource.url", DATABASE.getJdbcUrl());
    }

    @Override
    public void stop() {
        DATABASE.stop();
    }
}
```

HeroResourceTest.java

```
package io.quarkus.workshop.superheroes.hero;

import io.quarkus.test.common.QuarkusTestResource;
import io.quarkus.test.junit.QuarkusTest;
import io.restassured.common.mapper.TypeRef;
import org.hamcrest.core.Is;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;

import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.MediaType;
import java.util.List;
import java.util.Random;

import static io.restassured.RestAssured.get;
import static io.restassured.RestAssured.given;
import static javax.ws.rs.core.HttpHeaders.ACCEPT;
import static javax.ws.rs.core.HttpHeaders.CONTENT_TYPE;
import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static javax.ws.rs.core.Response.Status.*;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.jupiter.api.Assertions.*;

@QuarkusTest
@QuarkusTestResource(DatabaseResource.class)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class HeroResourceTest {

    private static final String DEFAULT_NAME = "Super Baguette";
    private static final String UPDATED_NAME = "Super Baguette (updated)";
    private static final String DEFAULT_OTHER_NAME = "Super Baguette Tradition";
    private static final String UPDATED_OTHER_NAME = "Super Baguette Tradition (updated)";
    private static final String DEFAULT_PICTURE = "super_baguette.png";
    private static final String UPDATED_PICTURE = "super_baguette_updated.png";
    private static final String DEFAULT_POWERS = "eats baguette really quickly";
    private static final String UPDATED_POWERS = "eats baguette really quickly (updated)";
    private static final int DEFAULT_LEVEL = 42;
    private static final int UPDATED_LEVEL = 43;

    private static final int NB_HEROES = 951;
    private static String heroId;




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

    @Test
    void shouldNotGetUnknownHero() {
        Long randomId = new Random().nextLong();
        given()
            .pathParam("id", randomId)
            .when().get("/api/heroes/{id}")
            .then()
            .statusCode(NO_CONTENT.getStatusCode());
```

```
    }

    @Test
    void shouldGetRandomHero() {
        given()
            .when().get("/api/heroes/random")
            .then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON);
    }

    @Test
    void shouldNotAddInvalidItem() {
        Hero hero = new Hero();
        hero.name = null;
        hero.otherName = DEFAULT_OTHER_NAME;
        hero.picture = DEFAULT_PICTURE;
        hero.powers = DEFAULT_POWERS;
        hero.level = 0;

        given()
            .body(hero)
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .header(ACCEPT, APPLICATION_JSON)
            .when()
            .post("/api/heroes")
            .then()
            .statusCode(BAD_REQUEST.getStatusCode());
    }

    @Test
    @Order(1)
    void shouldGetInitialItems() {
        List<Hero> heroes = get("/api/heroes").then()
            .statusCode(OK.getStatusCode())
            .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON)
            .extract().body().as(getHeroTypeRef());
        assertEquals(NB_HEROES, heroes.size());
    }

    @Test
    @Order(2)
    void shouldAddAnItem() {
        Hero hero = new Hero();
        hero.name = DEFAULT_NAME;
        hero.otherName = DEFAULT_OTHER_NAME;
        hero.picture = DEFAULT_PICTURE;
        hero.powers = DEFAULT_POWERS;
        hero.level = DEFAULT_LEVEL;

        String location = given()
            .body(hero)
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .header(ACCEPT, APPLICATION_JSON)
            .when()
            .post("/api/heroes")
            .then()
            .statusCode(CREATED.getStatusCode())
            .extract().header("Location");
        assertTrue(location.contains("/api/heroes"));

        // Stores the id
        String[] segments = location.split("/");
        heroId = segments[segments.length - 1];
        assertNotNull(heroId);

        given()
            .pathParam("id", heroId)
            .when().get("/api/heroes/{id}")
            .then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .body("name", Is.is(DEFAULT_NAME))
            .body("otherName", Is.is(DEFAULT_OTHER_NAME))
            .body("level", Is.is(DEFAULT_LEVEL))
            .body("picture", Is.is(DEFAULT_PICTURE))
            .body("powers", Is.is(DEFAULT_POWERS));

        List<Hero> heroes = get("/api/heroes").then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
```

```
            .extract().body().as(getHeroTypeRef());
        assertEquals(NB_HEROES + 1, heroes.size());
    }

    @Test
    @Order(3)
    void shouldUpdateAnItem() {
        Hero hero = new Hero();
        hero.id = Long.valueOf(heroId);
        hero.name = UPDATED_NAME;
        hero.otherName = UPDATED_OTHER_NAME;
        hero.picture = UPDATED_PICTURE;
        hero.powers = UPDATED_POWERS;
        hero.level = UPDATED_LEVEL;

        given()
            .body(hero)
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .header(ACCEPT, APPLICATION_JSON)
            .when()
            .put("/api/heroes")
            .then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .body("name", Is.is(UPDATED_NAME))
            .body("otherName", Is.is(UPDATED_OTHER_NAME))
            .body("level", Is.is(UPDATED_LEVEL))
            .body("picture", Is.is(UPDATED_PICTURE))
            .body("powers", Is.is(UPDATED_POWERS));

        List<Hero> heroes = get("/api/heroes").then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .extract().body().as(getHeroTypeRef());
        assertEquals(NB_HEROES + 1, heroes.size());
    }

    @Test
    @Order(4)
    void shouldRemoveAnItem() {
        given()
            .pathParam("id", heroId)
            .when().delete("/api/heroes/{id}")
            .then()
            .statusCode(NO_CONTENT.getStatusCode());

        List<Hero> heroes = get("/api/heroes").then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .extract().body().as(getHeroTypeRef());
        assertEquals(NB_HEROES, heroes.size());
    }

    private TypeRef<List<Hero>> getHeroTypeRef() {
        return new TypeRef<List<Hero>>() {
            // Kept empty on purpose
        };
    }
}
```

Configuring the Hero Microservice

```
quarkus.log.console.enable=true
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
quarkus.log.console.level=INFO
quarkus.log.console.color=true
```

```
quarkus.http.port=8083
```

```
# Business configuration
level.multiplier = 3
```

## Open API

```
./mvnw quarkus:add-extension -Dextensions="smallrye-openapi"

http://localhost:8083/openapi
```

HeroResource.java

```
package io.quarkus.workshop.superheroes.hero;

import org.eclipse.microprofile.openapi.annotations.Operation;
import org.eclipse.microprofile.openapi.annotations.enums.SchemaType;
import org.eclipse.microprofile.openapi.annotations.media.Content;
import org.eclipse.microprofile.openapi.annotations.media.Schema;
import org.eclipse.microprofile.openapi.annotations.parameters.Parameter;
import org.eclipse.microprofile.openapi.annotations.parameters.RequestBody;
import org.eclipse.microprofile.openapi.annotations.responses.APIResponse;
import org.jboss.logging.Logger;

import javax.inject.Inject;
import javax.validation.Valid;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.net.URI;
import java.util.List;

import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static javax.ws.rs.core.MediaType.TEXT_PLAIN;

@Path("/api/heroes")
@Produces(APPLICATION_JSON)
public class HeroResource {

    private static final Logger LOGGER = Logger.getLogger(HeroResource.class);

    @Inject
    HeroService service;

    @Operation(summary = "Returns a random hero")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Hero.class
    @GET
    @Path("/random")
    public Response getRandomHero() {
        Hero hero = service.findRandomHero();
        LOGGER.debug("Found random hero " + hero);
        return Response.ok(hero).build();
    }

    @Operation(summary = "Returns all the heroes from the database")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Hero.class
    @APIResponse(responseCode = "204", description = "No heroes")
    @GET
    public Response getAllHeroes() {
        List<Hero> heroes = service.findAllHeroes();
```

```
        LOGGER.debug("Total number of heroes " + heroes);
        return Response.ok(heroes).build();
    }

    @Operation(summary = "Returns a hero for a given identifier")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Hero.class
    @APIResponse(responseCode = "204", description = "The hero is not found for a given identifier")
    @GET
    @Path("/{id}")
    public Response getHero(
        @Parameter(description = "Hero identifier", required = true)
        @PathParam("id") Long id) {
        Hero hero = service.findHeroById(id);
        if (hero != null) {
            LOGGER.debug("Found hero " + hero);
            return Response.ok(hero).build();
        } else {
            LOGGER.debug("No hero found with id " + id);
            return Response.noContent().build();
        }
    }

    @Operation(summary = "Creates a valid hero")
    @APIResponse(responseCode = "201", description = "The URI of the created hero", content = @Content(mediaType = APPLICATION_JSON,
    @POST
    public Response createHero(
        @RequestBody(required = true, content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Hero.class)
        @Valid Hero hero, @Context UriInfo uriInfo) {
        hero = service.persistHero(hero);
        UriBuilder builder = uriInfo.getAbsolutePathBuilder().path(Long.toString(hero.id));
        LOGGER.debug("New hero created with URI " + builder.build().toString());
        return Response.created(builder.build()).build();
    }

    @Operation(summary = "Updates an exiting  hero")
    @APIResponse(responseCode = "200", description = "The updated hero", content = @Content(mediaType = APPLICATION_JSON, schema = @
    @PUT
    public Response updateHero(
        @RequestBody(required = true, content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Hero.class)
        @Valid Hero hero) {
        hero = service.updateHero(hero);
        LOGGER.debug("Hero updated with new valued " + hero);
        return Response.ok(hero).build();
    }

    @Operation(summary = "Deletes an exiting hero")
    @APIResponse(responseCode = "204")
    @DELETE
    @Path("/{id}")
    public Response deleteHero(
        @Parameter(description = "Hero identifier", required = true)
        @PathParam("id") Long id) {
        service.deleteHero(id);
        LOGGER.debug("Hero deleted with " + id);
        return Response.noContent().build();
    }

    @GET
    @Produces(TEXT_PLAIN)
    @Path("/hello")
    public String hello() {
        return "hello";
    }
}
```

## Customizing the Application

HeroApplication.java

```
package io.quarkus.workshop.superheroes.hero;

import org.eclipse.microprofile.openapi.annotations.ExternalDocumentation;
```

```
import org.eclipse.microprofile.openapi.annotations.OpenAPIDefinition;
import org.eclipse.microprofile.openapi.annotations.info.Contact;
import org.eclipse.microprofile.openapi.annotations.info.Info;
import org.eclipse.microprofile.openapi.annotations.servers.Server;
import org.eclipse.microprofile.openapi.annotations.tags.Tag;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/")
@OpenAPIDefinition(
    info = @Info(title = "Hero API",
        description = "This API allows CRUD operations on a hero",
        version = "1.0"
    servers = {
        @Server(url = "http://localhost:8083")
    },
    tags = {
        @Tag(name = "api", description = "Public that can be used by anybody"),
        @Tag(name = "heroes", description = "Anybody interested in heroes")
    }
)
public class HeroApplication extends Application {
}
```

```
http://localhost:8083/openapi
http://localhost:8083/swagger-ui
```

HeroResourceTest.java

```
@Test
void shouldPingOpenAPI() {
    given()
        .header(ACCEPT, APPLICATION_JSON)
        .when().get("/openapi")
        .then()
        .statusCode(OK.getStatusCode());
}

@Test
void shouldPingSwaggerUI() {
    given()
        .when().get("/swagger-ui")
        .then()
        .statusCode(OK.getStatusCode());
}
```

```
./mvnw test
```

---

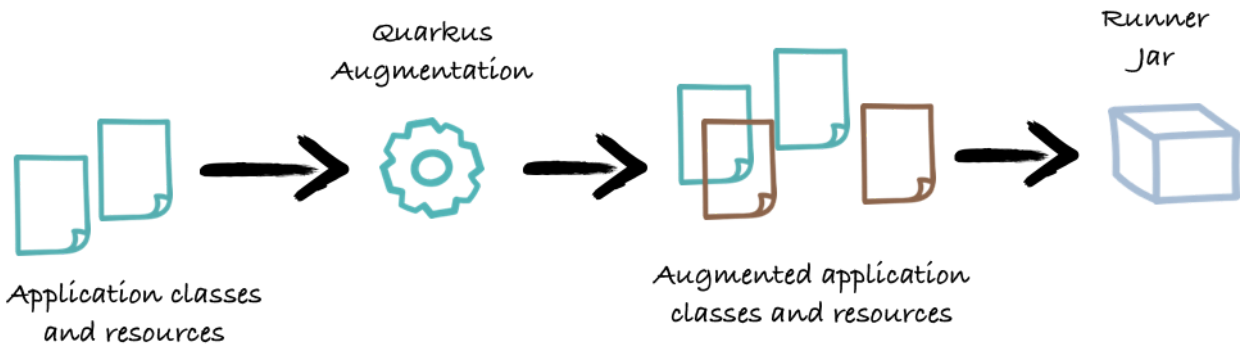
# What’s Quarkus?

Java was born more than 20 years ago. The world 20 years ago was quite different. The software industry has gone through several revolutions over these two decades. Java has always been able to reinvent itself to stay relevant.

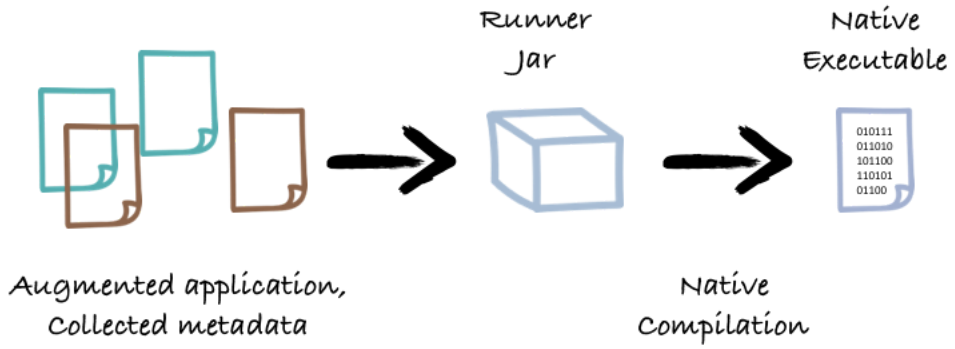
But a new revolution is happening right now. While for years, most applications were running on huge machines, with lots of CPU and memory, they are now running on the Cloud, in constrained environments, in containers, where the resources are shared. Density is the new optimization: crank as many mini-apps (or microservices) as possible per node. And scale by adding more instances of an app instead of a more powerful single instance.

The Java ergonomics, designed 20 years ago, do not fit well in this new environment. Java applications were designed to run 24/7 for months, even years. The JIT is optimizing the execution over time; the GC manages the memory efficiently… But all these features have a cost, and the memory required to run Java applications and startup times are showstoppers when instead of one application, you deploy 20 or 50 microservices. The issue is not the JVM itself; it's also the Java ecosystem that needs to be reinvented.

That's where Quarkus, and other projects, enter the game. Quarkus proposes to generalize "Ahead of Time" techniques. When a Quarkus application is built, some work that usually happens at runtime is moved to the build time. Thus, when the application runs, everything has been pre-computed, and all the annotation scanning, XML parsing, and so on won't be executed anymore. It has two direct benefits: on the startup time (a lot faster) and on memory consumption (a lot lower).




So, as depicted on the figure above, Quarkus does bring an infrastructure for frameworks to embrace build time metadata discovery (like annotations), declare which classes need reflection at runtime, boot at build time, and generally offer a lot GraalVM optimization for free (or cheap at least). Indeed, thanks to all these metadata, Quarkus can configure native compilers such as the SubstrateVM compiler to generate a native executable for your Java application. Thanks to an aggressive dead-code elimination, the final executable is smaller, faster to start and use a ridiculously small amount of memory.



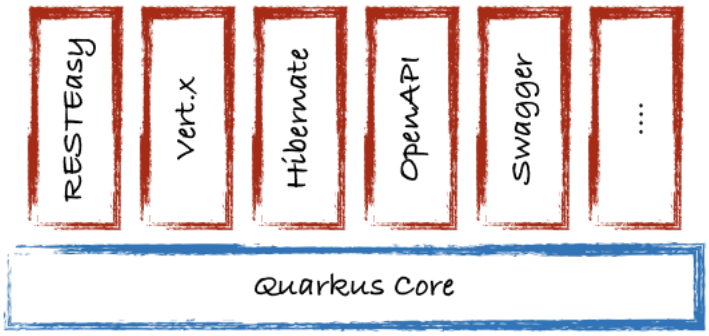

Quarkus does not stop there. As you have seen in the previous chapter, it proposes a stellar developer experience. It also unifies reactive and imperative so that you can mix regular JAX-RS and event-oriented code in the same application. Finally, Quarkus is based on many popular framework out there such as Eclipse Vert.x, Apache Camel, Undertow...You can already state that you have 5 years of experience with Quarkus.

Ok, but enough talking, time to see this in action.

---

Quarkus Augmentation

```
$ mvn package
```




Application Lifecycle

HeroApplicationLifeCycle.java

```
package io.quarkus.workshop.superheroes.hero;

import io.quarkus.runtime.ShutdownEvent;
import io.quarkus.runtime.StartupEvent;
import io.quarkus.runtime.configuration.ProfileManager;
import org.jboss.logging.Logger;

import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.event.Observes;

@ApplicationScoped
class HeroApplicationLifeCycle {

    private static final Logger LOGGER = Logger.getLogger(HeroApplicationLifeCycle.class);

    void onStart(@Observes StartupEvent ev) {
        LOGGER.info("  _   _                        _    ____ ___ ");
        LOGGER.info(" | | | | ___ _ __ ___         / \\  |  _ \\_ _|");
        LOGGER.info(" | |_| |/ _ \\ '__/ _ \\       / _ \\ | |_) | | ");
        LOGGER.info(" |  _  |  __/ | | (_) |    / ___ \\|  __/| | ");
        LOGGER.info(" |_| |_|\\___|_|  \\___/    /_/   \\_\\_|  |___|");
        LOGGER.info("                          Powered by Quarkus");
    }

    void onStop(@Observes ShutdownEvent ev) {
        LOGGER.info("The application HERO is stopping...");
    }
}
```

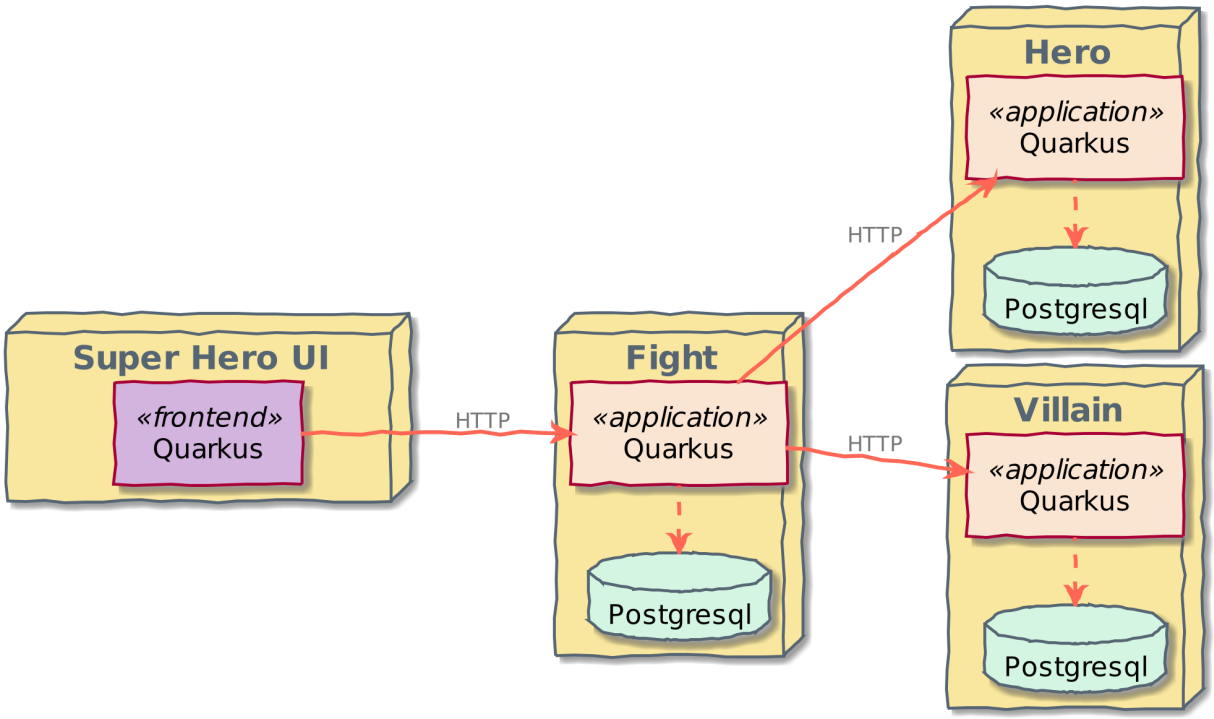
### Configuration Profiles

Quarkus supports the notion of configuration profiles. These allow you to have multiple configuration in the same file and select between them via a profile name.

By default Quarkus has three profiles, although it is possible to use as many as you like. The default profiles are:

- `dev` - Activated when in development mode (i.e. `quarkus:dev`)
- `test` - Activated when running tests
- `prod` - The default profile when not running in development or test mode

## One Microservice is no Microservices




Ex : Villain Microservice

Fight Microservice

Bootstrapping the Fight REST Endpoint

```
mvn io.quarkus:quarkus-maven-plugin:1.9.1.Final:create \
    -DprojectGroupId=io.quarkus.workshop.super-heroes \
    -DprojectArtifactId=rest-fight \
    -DclassName="io.quarkus.workshop.superheroes.fight.FightResource" \
    -Dpath="api/fights"
cd rest-fight
./mvnw quarkus:add-extension -Dextensions="jdbc-h2,jdbc-postgresql,hibernate-orm-panache,hibernate-validator,quarkus-resteasy-jsonb,
```

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <scope>test</scope>
</dependency>
```

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>1.12.2</version>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>postgresql</artifactId>
    <version>1.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>kafka</artifactId>
    <version>1.12.2</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <scope>test</scope>
</dependency>
```

### Fight Entity

A fight is between a hero and a villain. Each time there is a fight, there is a winner and a loser. So the `Fight` entity is there to store all these fights.

Fight.java

```
package io.quarkus.workshop.superheroes.fight;

import io.quarkus.hibernate.orm.panache.PanacheEntity;
import org.eclipse.microprofile.openapi.annotations.media.Schema;

import javax.persistence.Entity;
import javax.validation.constraints.NotNull;
import java.time.Instant;

@Entity
@Schema(description="Each fight has a winner and a loser")
public class Fight extends PanacheEntity {

    @NotNull
    public Instant fightDate;
    @NotNull
    public String winnerName;
    @NotNull
    public int winnerLevel;
    @NotNull
    public String winnerPicture;
    @NotNull
    public String loserName;
    @NotNull
    public int loserLevel;
    @NotNull
    public String loserPicture;
    @NotNull
    public String winnerTeam;
    @NotNull
    public String loserTeam;

    // toString method
}
```

### Fighters Bean

Now comes a trick. The Fight REST API will ultimatelly invoke the Hero and Villain APIs (next sections) to get two random fighters. The `Fighters` class has one `Hero` and one `Villain`. Notice that `Fighters` is not an entity, it is not persisted in the database, just marshalled and unmarshalled to JSon.

Fighters.java

```
package io.quarkus.workshop.superheroes.fight;

import io.quarkus.workshop.superheroes.fight.client.Hero;
import io.quarkus.workshop.superheroes.fight.client.Villain;
import org.eclipse.microprofile.openapi.annotations.media.Schema;

import javax.validation.constraints.NotNull;

@Schema(description="A fight between one hero and one villain")
public class Fighters {

    @NotNull
    public Hero hero;
    @NotNull
    public Villain villain;

}
```

Hero.java

```
package io.quarkus.workshop.superheroes.fight.client;

import org.eclipse.microprofile.openapi.annotations.media.Schema;

import javax.validation.constraints.NotNull;

@Schema(description="The hero fighting against the villain")
public class Hero {

    @NotNull
    public String name;
    @NotNull
    public int level;
    @NotNull
    public String picture;
    public String powers;

}
```

Villain.java

```
package io.quarkus.workshop.superheroes.fight.client;

import org.eclipse.microprofile.openapi.annotations.media.Schema;

import javax.validation.constraints.NotNull;

@Schema(description="The villain fighting against the hero")
public class Villain {
```

```
    @NotNull
    public String name;
    @NotNull
    public int level;
    @NotNull
    public String picture;
    public String powers;

}
```

So, these classes are just used to map the results from the Hero and Villain microservices.

### FightService Transactional Service

To *transactionnally* manipulate the `Fight` entity we need a `FightService`. Notice the `persistFight` method. This method is the one creating a fight between a hero and a villain. As you can see the algorithm to determine the winner is a bit random (even though it uses the levels). If you are not happy about the way the fight operates, choose your own winning algorithm ;o)

```
package io.quarkus.workshop.superheroes.fight;

import io.quarkus.workshop.superheroes.fight.client.Hero;
import io.quarkus.workshop.superheroes.fight.client.Villain;
import org.jboss.logging.Logger;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.transaction.Transactional;
import java.time.Instant;
import java.util.List;
import java.util.Random;

import static javax.transaction.Transactional.TxType.REQUIRED;
import static javax.transaction.Transactional.TxType.SUPPORTS;

@ApplicationScoped
@Transactional(SUPPORTS)
public class FightService {

    private static final Logger LOGGER = Logger.getLogger(FightService.class);

    private final Random random = new Random();

    public List<Fight> findAllFights() {
        return Fight.listAll();
    }

    public Fight findFightById(Long id) {
        return Fight.findById(id);
    }

    @Transactional(REQUIRED)
    public Fight persistFight(Fighters fighters) {
        // Amazingly fancy logic to determine the winner...
        Fight fight;

        int heroAdjust = random.nextInt(20);
        int villainAdjust = random.nextInt(20);

        if ((fighters.hero.level + heroAdjust)
            > (fighters.villain.level + villainAdjust)) {
            fight = heroWon(fighters);
        } else if (fighters.hero.level < fighters.villain.level) {
            fight = villainWon(fighters);
        } else {
            fight = random.nextBoolean() ? heroWon(fighters) : villainWon(fighters);
        }
```

```
        fight.fightDate = Instant.now();
        fight.persist(fight);
        return fight;
    }

    private Fight heroWon(Fighters fighters) {
        LOGGER.info("Yes, Hero won :o)");
        Fight fight = new Fight();
        fight.winnerName = fighters.hero.name;
        fight.winnerPicture = fighters.hero.picture;
        fight.winnerLevel = fighters.hero.level;
        fight.loserName = fighters.villain.name;
        fight.loserPicture = fighters.villain.picture;
        fight.loserLevel = fighters.villain.level;
        fight.winnerTeam = "heroes";
        fight.loserTeam = "villains";
        return fight;
    }

    private Fight villainWon(Fighters fighters) {
        LOGGER.info("Gee, Villain won :o(");
        Fight fight = new Fight();
        fight.winnerName = fighters.villain.name;
        fight.winnerPicture = fighters.villain.picture;
        fight.winnerLevel = fighters.villain.level;
        fight.loserName = fighters.hero.name;
        fight.loserPicture = fighters.hero.picture;
        fight.loserLevel = fighters.hero.level;
        fight.winnerTeam = "villains";
        fight.loserTeam = "heroes";
        return fight;
    }


}
```

```
public Fighters findRandomFighters() {
    // Will be implemented later
    return null;
}
```

FightResource.java

```
package io.quarkus.workshop.superheroes.fight;

import org.eclipse.microprofile.openapi.annotations.Operation;
import org.eclipse.microprofile.openapi.annotations.enums.SchemaType;
import org.eclipse.microprofile.openapi.annotations.media.Content;
import org.eclipse.microprofile.openapi.annotations.media.Schema;
import org.eclipse.microprofile.openapi.annotations.parameters.Parameter;
import org.eclipse.microprofile.openapi.annotations.parameters.RequestBody;
import org.eclipse.microprofile.openapi.annotations.responses.APIResponse;
import org.jboss.logging.Logger;

import javax.inject.Inject;
import javax.validation.Valid;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import java.util.List;

import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
```

```
import static javax.ws.rs.core.MediaType.TEXT_PLAIN;

@Path("/api/fights")
@Produces(APPLICATION_JSON)
public class FightResource {

    private static final Logger LOGGER = Logger.getLogger(FightResource.class);

    @Inject
    FightService service;

    @Operation(summary = "Returns two random fighters")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Fighters.c
    @GET
    @Path("/randomfighters")
    public Response getRandomFighters() throws InterruptedException {
        Fighters fighters = service.findRandomFighters();
        LOGGER.debug("Get random fighters " + fighters);
        return Response.ok(fighters).build();
    }

    @Operation(summary = "Returns all the fights from the database")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Fight.clas
    @APIResponse(responseCode = "204", description = "No fights")
    @GET
    public Response getAllFights() {
        List<Fight> fights = service.findAllFights();
        LOGGER.debug("Total number of fights " + fights);
        return Response.ok(fights).build();
    }

    @Operation(summary = "Returns a fight for a given identifier")
    @APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Fight.clas
    @APIResponse(responseCode = "204", description = "The fight is not found for a given identifier")
    @GET
    @Path("/{id}")
    public Response getFight(@Parameter(description = "Fight identifier", required = true) @PathParam("id") Long id) {
        Fight fight = service.findFightById(id);
        if (fight != null) {
            LOGGER.debug("Found fight " + fight);
            return Response.ok(fight).build();
        } else {
            LOGGER.debug("No fight found with id " + id);
            return Response.noContent().build();
        }
    }

    @Operation(summary = "Trigger a fight between two fighters")
    @APIResponse(responseCode = "200", description = "The result of the fight", content = @Content(mediaType = APPLICATION_JSON, sch
    @POST
    public Fight fight(@RequestBody(description = "The two fighters fighting", required = true, content = @Content(mediaType = APPLI
        return service.persistFight(fighters);
    }

    @GET
    @Produces(TEXT_PLAIN)
    @Path("/hello")
    public String hello() {
        return "hello";
    }
}
```

FightApplication.java

```
package io.quarkus.workshop.superheroes.fight;

import org.eclipse.microprofile.openapi.annotations.ExternalDocumentation;
import org.eclipse.microprofile.openapi.annotations.OpenAPIDefinition;
import org.eclipse.microprofile.openapi.annotations.info.Contact;
import org.eclipse.microprofile.openapi.annotations.info.Info;
import org.eclipse.microprofile.openapi.annotations.servers.Server;
import org.eclipse.microprofile.openapi.annotations.tags.Tag;

import javax.ws.rs.ApplicationPath;
```

```
import javax.ws.rs.core.Application;

@ApplicationPath("/")
@OpenAPIDefinition(
    info = @Info(title = "Fight API",
        description = "This API allows a hero and a villain to fight",
        version = "1.0"),
    servers = {
        @Server(url = "http://localhost:8082")
    },

    tags = {
        @Tag(name = "api", description = "Public that can be used by anybody"),
        @Tag(name = "fight", description = "Anybody interested in fights"),
        @Tag(name = "superheroes", description = "Well, superhero fights")
    }
)
public class FightApplication extends Application {
}
```

Adding Data

```
import.sql
```

Configuration

application.properties

```
quarkus.http.port=8082



## Database configuration
quarkus.datasource.url=jdbc:postgresql://localhost:5432/fights_database
quarkus.datasource.driver=org.postgresql.Driver
quarkus.datasource.username=superfight
quarkus.datasource.password=superfight
quarkus.datasource.max-size=8
quarkus.datasource.min-size=2
quarkus.hibernate-orm.database.generation=drop-and-create
quarkus.hibernate-orm.log.sql=true

## Logging configuration
quarkus.log.console.enable=true
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
quarkus.log.console.level=DEBUG
quarkus.log.console.color=true

## Production configuration
%prod.quarkus.hibernate-orm.log.sql=false
%prod.quarkus.log.console.level=INFO
%prod.quarkus.hibernate-orm.database.generation=update

process.milliseconds=0
```

FightResourceTest.java

```
package io.quarkus.workshop.superheroes.fight;

import io.quarkus.test.common.QuarkusTestResource;
import io.quarkus.test.junit.QuarkusTest;
import io.quarkus.workshop.superheroes.fight.client.Hero;
import io.quarkus.workshop.superheroes.fight.client.Villain;
import io.restassured.common.mapper.TypeRef;
import org.hamcrest.core.Is;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;

import java.util.List;

import java.util.Random;
import static io.restassured.RestAssured.get;
import static io.restassured.RestAssured.given;
import static javax.ws.rs.core.HttpHeaders.ACCEPT;
import static javax.ws.rs.core.HttpHeaders.CONTENT_TYPE;
import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static javax.ws.rs.core.Response.Status.*;
import static org.hamcrest.CoreMatchers.*;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;



@QuarkusTest
@QuarkusTestResource(DatabaseResource.class)
@QuarkusTestResource(KafkaResource.class)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class FightResourceTest {

    private static final String DEFAULT_WINNER_NAME = "Super Baguette";
    private static final String DEFAULT_WINNER_PICTURE = "super_baguette.png";
    private static final int DEFAULT_WINNER_LEVEL = 42;
    private static final String DEFAULT_LOSER_NAME = "Super Chocolatine";
    private static final String DEFAULT_LOSER_PICTURE = "super_chocolatine.png";
    private static final int DEFAULT_LOSER_LEVEL = 6;

    private static final int NB_FIGHTS = 10;
    private static String fightId;

    @Test
    void shouldPingOpenAPI() {
        given()
            .header(ACCEPT, APPLICATION_JSON)
            .when().get("/openapi")
            .then()
            .statusCode(OK.getStatusCode());
    }

    @Test
    void shouldPingSwaggerUI() {
        given()
            .when().get("/swagger-ui")
            .then()
            .statusCode(OK.getStatusCode());
    }



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

    @Test
    void shouldNotGetUnknownFight() {
        Long randomId = new Random().nextLong();
        given()
            .pathParam("id", randomId)
            .when().get("/api/fights/{id}")
```

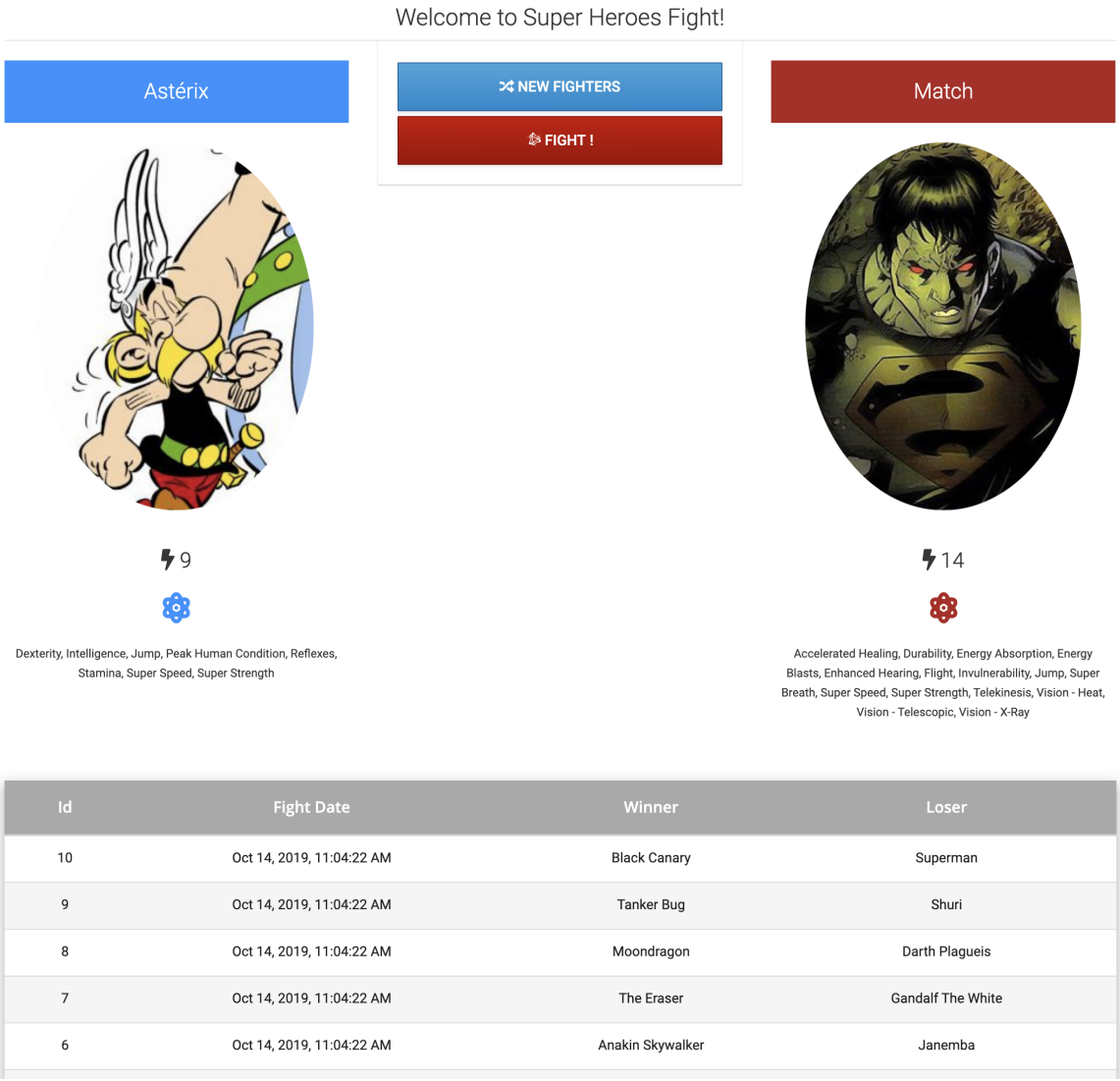
```
            .then()
            .statusCode(NO_CONTENT.getStatusCode());
    }


    @Test
    void shouldNotAddInvalidItem() {
        Fighters fighters = new Fighters();
        fighters.hero = null;
        fighters.villain = null;

        given()
            .body(fighters)
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .header(ACCEPT, APPLICATION_JSON)
            .when()
            .post("/api/fights")
            .then()
            .statusCode(BAD_REQUEST.getStatusCode());
    }

    @Test
    @Order(1)
    void shouldGetInitialItems() {
        List<Fight> fights = get("/api/fights").then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .extract().body().as(getFightTypeRef());
        assertEquals(NB_FIGHTS, fights.size());
    }

    @Test
    @Order(2)
    void shouldAddAnItem() {
        Hero hero = new Hero();
        hero.name = DEFAULT_WINNER_NAME;
        hero.picture = DEFAULT_WINNER_PICTURE;
        hero.level = DEFAULT_WINNER_LEVEL;
        Villain villain = new Villain();
        villain.name = DEFAULT_LOSER_NAME;
        villain.picture = DEFAULT_LOSER_PICTURE;
        villain.level = DEFAULT_LOSER_LEVEL;
        Fighters fighters = new Fighters();
        fighters.hero = hero;
        fighters.villain = villain;

        fightId = given()
            .body(fighters)
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .header(ACCEPT, APPLICATION_JSON)
            .when()
            .post("/api/fights")
            .then()
            .statusCode(OK.getStatusCode())
            .body(containsString("winner"), containsString("loser"))
            .extract().body().jsonPath().getString("id");

        assertNotNull(fightId);

        given()
            .pathParam("id", fightId)
            .when().get("/api/fights/{id}")
            .then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .body("winnerName", Is.is(DEFAULT_WINNER_NAME))
            .body("winnerPicture", Is.is(DEFAULT_WINNER_PICTURE))
            .body("winnerLevel", Is.is(DEFAULT_WINNER_LEVEL))
            .body("loserName", Is.is(DEFAULT_LOSER_NAME))
            .body("loserPicture", Is.is(DEFAULT_LOSER_PICTURE))
            .body("loserLevel", Is.is(DEFAULT_LOSER_LEVEL))
            .body("fightDate", Is.is(notNullValue()));

        List<Fight> fights = get("/api/fights").then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .extract().body().as(getFightTypeRef());
        assertEquals(NB_FIGHTS + 1, fights.size());
    }

    private TypeRef<List<Fight>> getFightTypeRef() {
```

```
        return new TypeRef<List<Fight>>() {
            // Kept empty on purpose
        };
    }
}
```

Running, Testing and Packaging the Application

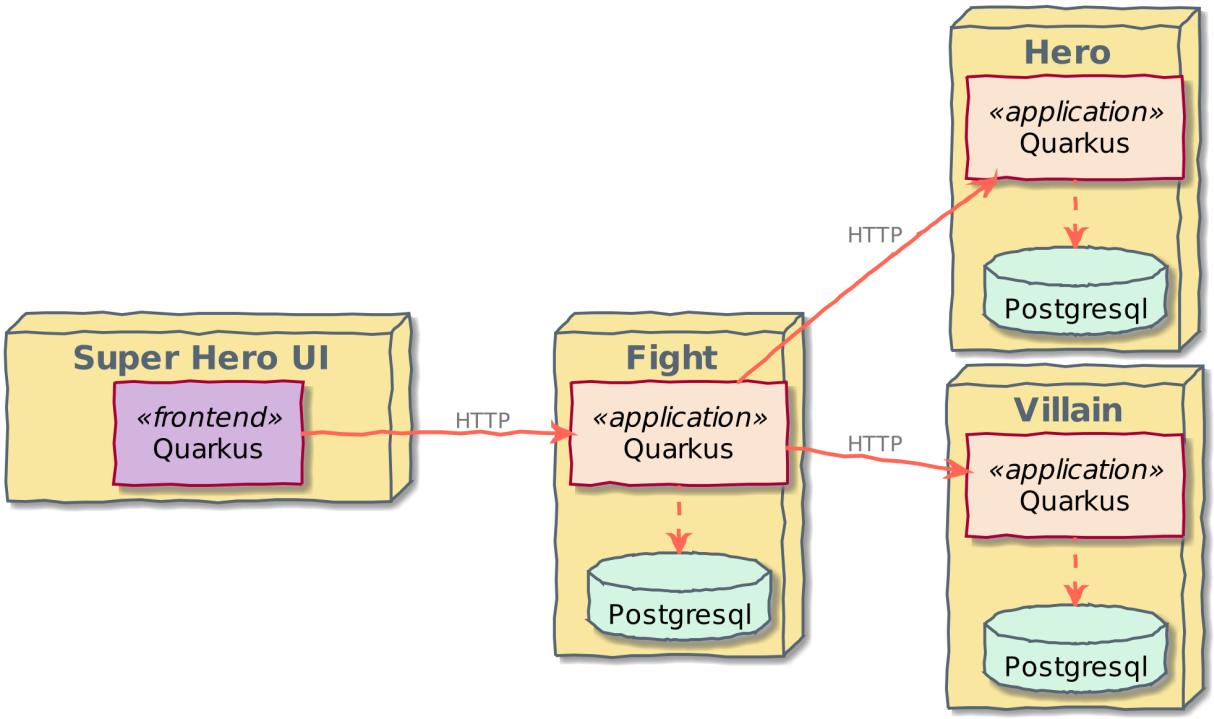
```
$ curl http://localhost:8082/api/fights
```

User Interface




| Id | Fight Date | Winner | Loser |
|---|---|---|---|
| 10 | Oct 14, 2019, 11:04:22 AM | Black Canary | Superman |
| 9 | Oct 14, 2019, 11:04:22 AM | Tanker Bug | Shuri |
| 8 | Oct 14, 2019, 11:04:22 AM | Moondragon | Darth Plagueis |
| 7 | Oct 14, 2019, 11:04:22 AM | The Eraser | Gandalf The White |
| 6 | Oct 14, 2019, 11:04:22 AM | Anakin Skywalker | Janemba |

HTTP communication & Fault Tolerance




Installing the REST Client Dependency

```
./mvnw quarkus:add-extension -Dextensions="rest-client"
```

FightService Invoking External Microservices

FightService.java

```
@Inject
@RestClient
HeroService heroService;

@Inject
@RestClient
VillainService villainService;

Fighters findRandomFighters() {
    Hero hero = findRandomHero();
    Villain villain = findRandomVillain();
    Fighters fighters = new Fighters();
    fighters.hero = hero;
    fighters.villain = villain;
    return fighters;
}

Hero findRandomHero() {
    return heroService.findRandomHero();
}
```

```
Villain findRandomVillain() {
    return villainService.findRandomVillain();
}
```

Creating the Interfaces

HeroService.java

```
package io.quarkus.workshop.superheroes.fight.client;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/api/heroes")
@Produces(MediaType.APPLICATION_JSON)
@RegisterRestClient
public interface HeroService {

    @GET
    @Path("/random")
    Hero findRandomHero();
}
```

VillainService.java

```
package io.quarkus.workshop.superheroes.fight.client;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/api/villains")
@Produces(MediaType.APPLICATION_JSON)
@RegisterRestClient
public interface VillainService {

    @GET
    @Path("/random")
    Villain findRandomVillain();
}
```

Configuring REST Client Invocation

```
io.quarkus.workshop.superheroes.fight.client.HeroService/mp-rest/url=http://localhost:8083
io.quarkus.workshop.superheroes.fight.client.HeroService/mp-rest/scope=javax.inject.Singleton
io.quarkus.workshop.superheroes.fight.client.VillainService/mp-rest/url=http://localhost:8084
io.quarkus.workshop.superheroes.fight.client.VillainService/mp-rest/scope=javax.inject.Singleton
```

Updating the Test with Mock Support

MockHeroService.java

```
package io.quarkus.workshop.superheroes.fight.client;

import io.quarkus.test.Mock;
import org.eclipse.microprofile.rest.client.inject.RestClient;

import javax.enterprise.context.ApplicationScoped;

@Mock
@ApplicationScoped
@RestClient
public class MockHeroService implements HeroService {

    public static final String DEFAULT_HERO_NAME = "Super Baguette";
    public static final String DEFAULT_HERO_PICTURE = "super_baguette.png";
    public static final String DEFAULT_HERO_POWERS = "eats baguette really quickly";
    public static final int DEFAULT_HERO_LEVEL = 42;

    @Override
    public Hero findRandomHero() {
        Hero hero = new Hero();
        hero.name = DEFAULT_HERO_NAME;
        hero.picture = DEFAULT_HERO_PICTURE;
        hero.powers = DEFAULT_HERO_POWERS;
        hero.level = DEFAULT_HERO_LEVEL;
        return hero;
    }
}
```

MockVillainService.java

```
package io.quarkus.workshop.superheroes.fight.client;

import io.quarkus.test.Mock;
import org.eclipse.microprofile.rest.client.inject.RestClient;

import javax.enterprise.context.ApplicationScoped;

@Mock
@ApplicationScoped
@RestClient
public class MockVillainService implements VillainService {

    public static final String DEFAULT_VILLAIN_NAME = "Super Chocolatine";
    public static final String DEFAULT_VILLAIN_PICTURE = "super_chocolatine.png";
    public static final String DEFAULT_VILLAIN_POWERS = "does not eat pain au chocolat";
    public static final int DEFAULT_VILLAIN_LEVEL = 42;

    @Override
    public Villain findRandomVillain() {
        Villain villain = new Villain();
        villain.name = DEFAULT_VILLAIN_NAME;
        villain.picture = DEFAULT_VILLAIN_PICTURE;
        villain.powers = DEFAULT_VILLAIN_POWERS;
        villain.level = DEFAULT_VILLAIN_LEVEL;
        return villain;
    }
}
```

FightResourceTest.java

```
import io.quarkus.workshop.superheroes.fight.client.MockHeroService;
import io.quarkus.workshop.superheroes.fight.client.MockVillainService;

    //....
    @Test
    void shouldGetRandomFighters() {
        given()
            .when().get("/api/fights/randomfighters")
            .then()
            .statusCode(OK.getStatusCode())
            .header(CONTENT_TYPE, APPLICATION_JSON)
            .body("hero.name", Is.is(MockHeroService.DEFAULT_HERO_NAME))
            .body("hero.picture", Is.is(MockHeroService.DEFAULT_HERO_PICTURE))
            .body("hero.level", Is.is(MockHeroService.DEFAULT_HERO_LEVEL))
            .body("villain.name", Is.is(MockVillainService.DEFAULT_VILLAIN_NAME))
            .body("villain.picture", Is.is(MockVillainService.DEFAULT_VILLAIN_PICTURE))
            .body("villain.level", Is.is(MockVillainService.DEFAULT_VILLAIN_LEVEL));
    }
```

Fallbacks (optional)

Installing the Fault Tolerance Dependency

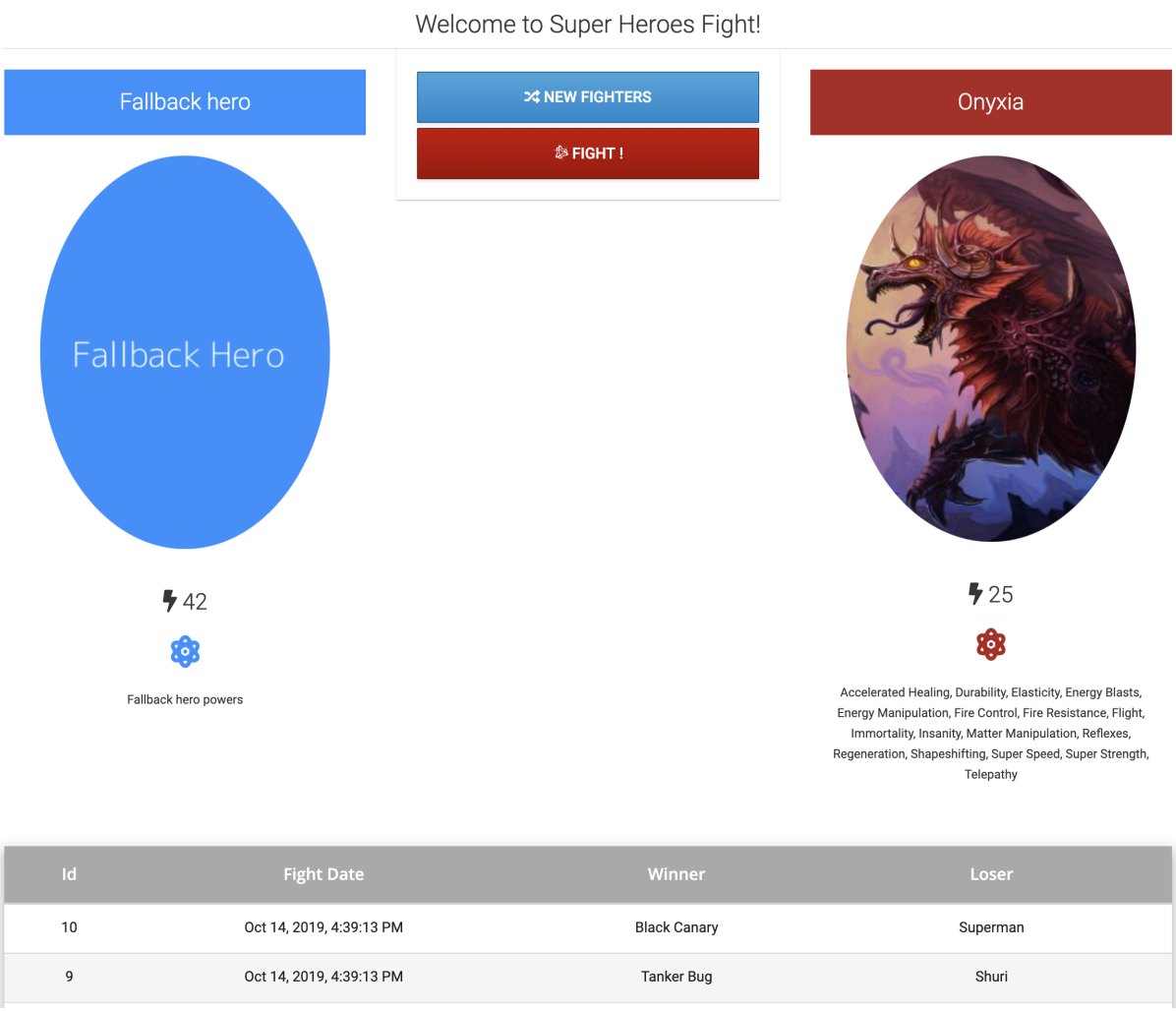
```
./mvnw quarkus:add-extension -Dextensions="smallrye-fault-tolerance"
```

Adding Fallbacks

FightService.java

```
@Inject
@RestClient
HeroService heroService;

@Inject
@RestClient
VillainService villainService;

Fighters findRandomFighters() {
    Hero hero = findRandomHero();
    Villain villain = findRandomVillain();
    Fighters fighters = new Fighters();
    fighters.hero = hero;
    fighters.villain = villain;
    return fighters;
}

@Fallback(fallbackMethod = "fallbackRandomHero")
Hero findRandomHero() {
    return heroService.findRandomHero();
}

@Fallback(fallbackMethod = "fallbackRandomVillain")
Villain findRandomVillain() {
    return villainService.findRandomVillain();
}

public Hero fallbackRandomHero() {
    LOGGER.warn("Falling back on Hero");
    Hero hero = new Hero();
    hero.name = "Fallback hero";
    hero.picture = "https://dummyimage.com/280x380/1e8fff/ffffff&text=Fallback+Hero";
```

```
    hero.powers = "Fallback hero powers";
    hero.level = 1;
    return hero;
}

public Villain fallbackRandomVillain() {
    LOGGER.warn("Falling back on Villain");
    Villain villain = new Villain();
    villain.name = "Fallback villain";
    villain.picture = "https://dummyimage.com/280x380/b22222/ffffff&text=Fallback+Villain";
    villain.powers = "Fallback villain powers";
    villain.level = 42;
    return villain;
}
```



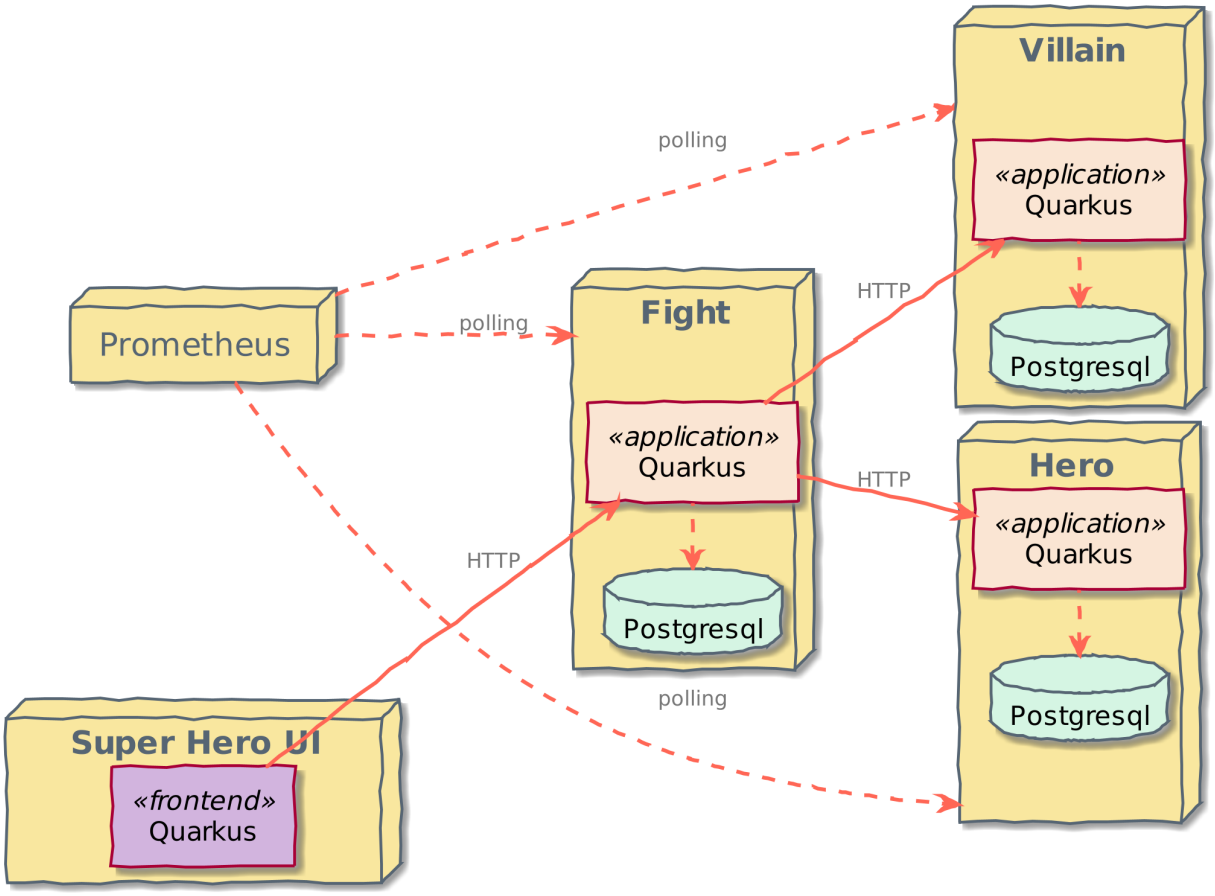

## Adding Timeouts

Getting random fighters can take longer than expected.

FightResource.java

```
@ConfigProperty(name = "process.milliseconds", defaultValue="0")
long tooManyMilliseconds;

private void veryLongProcess() throws InterruptedException {
    Thread.sleep(tooManyMilliseconds);
}

@Operation(summary = "Returns two random fighters")
@APIResponse(responseCode = "200", content = @Content(mediaType = APPLICATION_JSON, schema = @Schema(implementation = Fighters.class
@Timeout(250)
@GET
@Path("/randomfighters")
public Response getRandomFighters() throws InterruptedException {
    veryLongProcess();
    Fighters fighters = service.findRandomFighters();
    LOGGER.debug("Get random fighters " + fighters);
    return Response.ok(fighters).build();
}
```

Observability (optional)




Health Check

Installing the Health Dependency

```
./mvnw quarkus:add-extension -Dextensions="health"
```

Adding Liveness

PingHeroResourceHealthCheck.java

```
package io.quarkus.workshop.superheroes.hero.health;

import io.quarkus.workshop.superheroes.hero.HeroResource;
import org.eclipse.microprofile.health.HealthCheck;
import org.eclipse.microprofile.health.HealthCheckResponse;
import org.eclipse.microprofile.health.Liveness;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
```

```
@Liveness
@ApplicationScoped
public class PingHeroResourceHealthCheck implements HealthCheck {

    @Inject
    HeroResource heroResource;

    @Override
    public HealthCheckResponse call() {
        heroResource.hello();
        return HealthCheckResponse.named("Ping Hero REST Endpoint").up().build();
    }
}
```

Adding Readiness

DatabaseConnectionHealthCheck.java

```
package io.quarkus.workshop.superheroes.hero.health;

import io.quarkus.workshop.superheroes.hero.Hero;
import io.quarkus.workshop.superheroes.hero.HeroService;
import org.eclipse.microprofile.health.HealthCheck;
import org.eclipse.microprofile.health.HealthCheckResponse;
import org.eclipse.microprofile.health.HealthCheckResponseBuilder;
import org.eclipse.microprofile.health.Readiness;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import java.util.List;

@Readiness
@ApplicationScoped
public class DatabaseConnectionHealthCheck implements HealthCheck {

    @Inject
    HeroService heroService;

    @Override
    public HealthCheckResponse call() {
        HealthCheckResponseBuilder responseBuilder = HealthCheckResponse
            .named("Hero Datasource connection health check");

        try {
            List<Hero> heroes = heroService.findAllHeroes();
            responseBuilder.withData("Number of heroes in the database", heroes.size()).up();
        } catch (IllegalStateException e) {
            responseBuilder.down();
        }

        return responseBuilder.build();
    }
}
```

Health Check Tests in HeroResourceTest

```
@Test
void shouldPingLiveness() {
    given()
        .when().get("/health/live")
        .then()
        .statusCode(OK.getStatusCode());
}

@Test
void shouldPingReadiness() {
```

```
    given()
        .when().get("/health/ready")
        .then()
        .statusCode(OK.getStatusCode());
}
```

Metrics

MicroProfile Metrics allows applications to gather various metrics and statistics that provide insights into what is happening inside the application. The metrics can be read remotely using JSON format or the OpenMetrics format, so that they can be processed by additional tools such as Prometheus, and stored for analysis and visualisation.

Installing the Metrics Dependency

```
./mvnw quarkus:add-extension -Dextensions="metrics"
```

HeroResource.java

```
@Counted(name = "countGetRandomHero", description = "Counts how many times the getRandomHero method has been invoked")
@Timed(name = "timeGetRandomHero", description = "Times how long it takes to invoke the getRandomHero method", unit = MetricUnits.MI
@GET
@Path("/random")
public Response getRandomHero() {
    Hero hero = service.findRandomHero();
    LOGGER.debug("Found random hero " + hero);
    return Response.ok(hero).build();
}
```

```
@Test
void shouldPingMetrics() {
    given()
        .header(ACCEPT, APPLICATION_JSON)
        .when().get("/metrics/application")
        .then()
        .statusCode(OK.getStatusCode());
}
```

```
curl -H "Accept: application/json" http://localhost:8083/metrics/application
```

Loading the Microservices

Running the Load Application

```
$ mvn compile
$ mvn exec:java
```

Displaying Metrics on Prometheus

To execute the application we now need Prometheus. Make sure the infrastructure is up and running. This means that you've executed

```
docker-compose -f docker-compose.yaml up -d.
```

### Adding Graphs to Prometheus

The Prometheus console is accessible at http://localhost:9090.

# Event-driven and Reactive microservices

So far, we have build 3 microservices, all using HTTP to interact. However, HTTP has significant flaws, such as temporal coupling between the actors. If the service is not there or is slow, the caller is directly impacted. Also, it's hard to guess the capacity of the service you call; maybe you should not call it right now because this service is under heavy load.

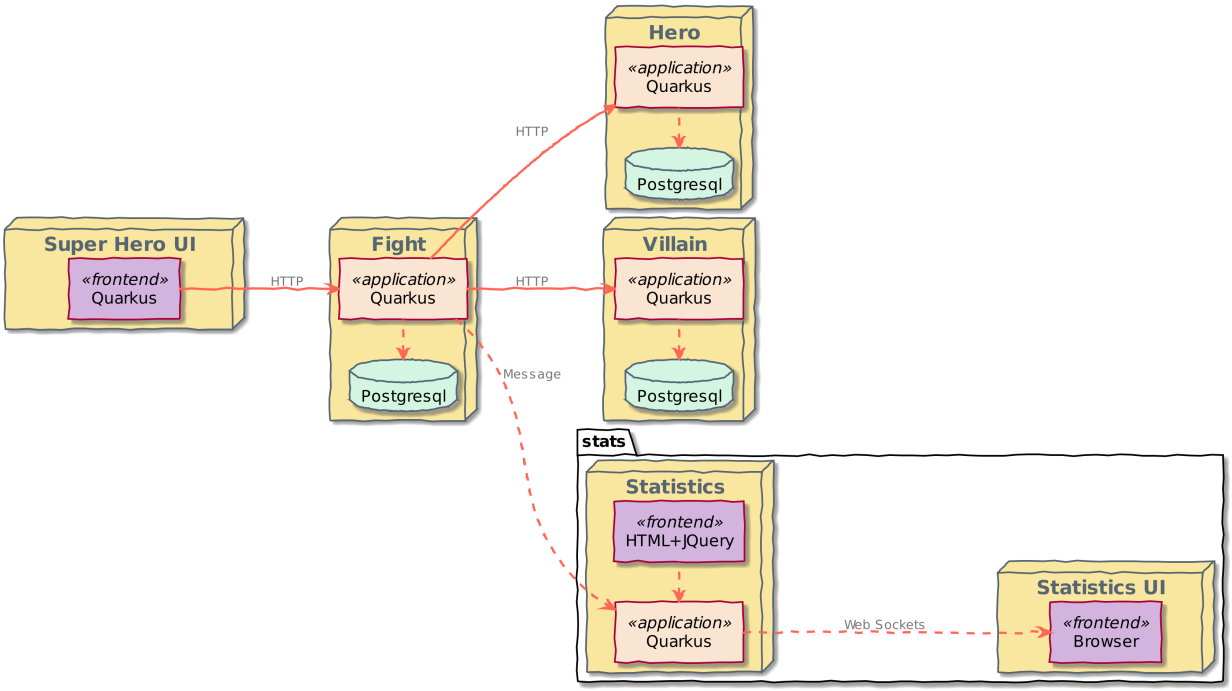
Fortunately, event-driven microservices are rising and avoid most of these issues. By using events (wrapped in messages), the different microservices enforce a looser coupling. Depending on the messaging protocol you use, it may handle durability (avoiding the temporal coupling) and back-pressure (avoiding the overload).

In this section, we are going to see how Quarkus let you build event-driven microservices. More specially, you are going to see how to:

- send messages and process them
- connect a Quarkus application to Apache Kafka
- write Kafka records and read them
- use reactive programming to compute statistics on the fly
- how to send messages to the browser using web sockets

Quarkus uses MicroProfile Reactive Messaging to interact with Kafka, and other messaging middleware (such as AMQP)

we are going to use events as a way for microservices to interact. You are going to extend the current system with the `stats` group depicted on the next figure:




Sending Messages to Kafka

Adding the Reactive Messaging Dependency

```
./mvnw quarkus:add-extension -Dextensions="kafka"
```

Connecting Imperative and Reactive Using an Emitter

```
@Inject
@Channel("fights") Emitter<Fight> emitter;
```

```
emitter.send(fight);
```

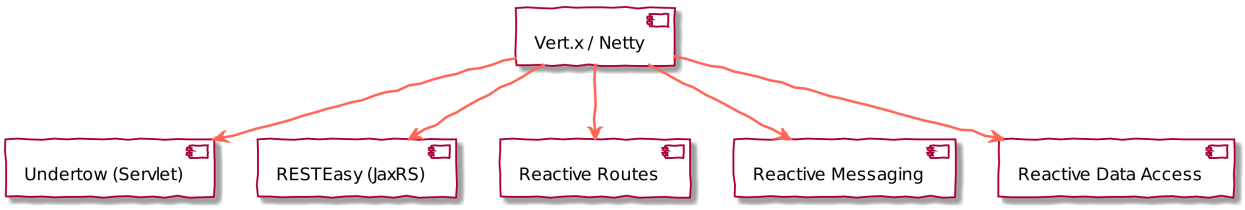
Connecting to Kafka

```
## Kafka configuration
mp.messaging.outgoing.fights.connector=smallrye-kafka
mp.messaging.outgoing.fights.value.serializer=io.quarkus.kafka.client.serialization.JsonbSerializer
```

```
mp.messaging.[incoming|outgoing].channel.attribute=value
```

Statistics service

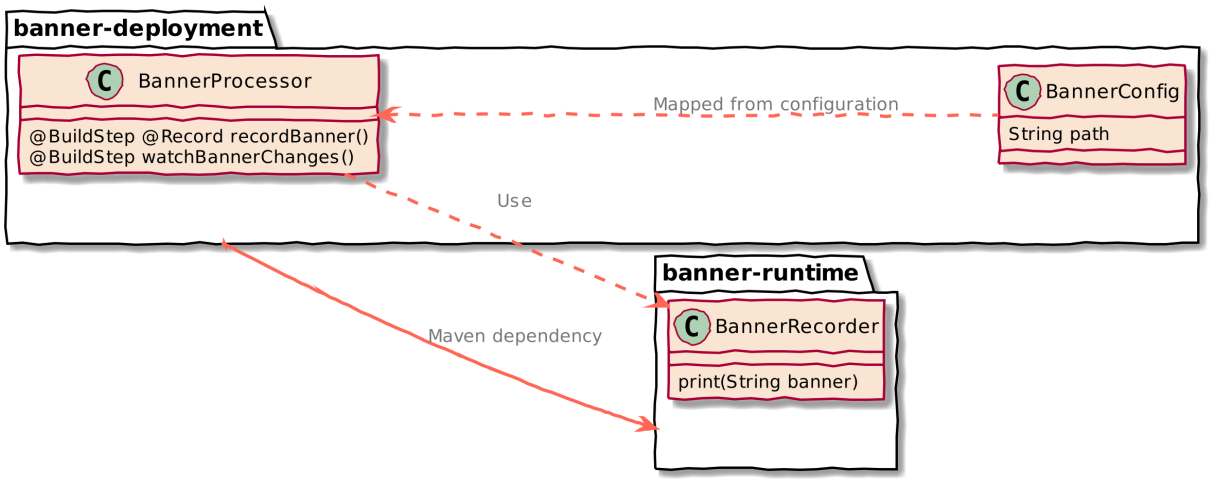
Unifying Imperative and Reactive Programming

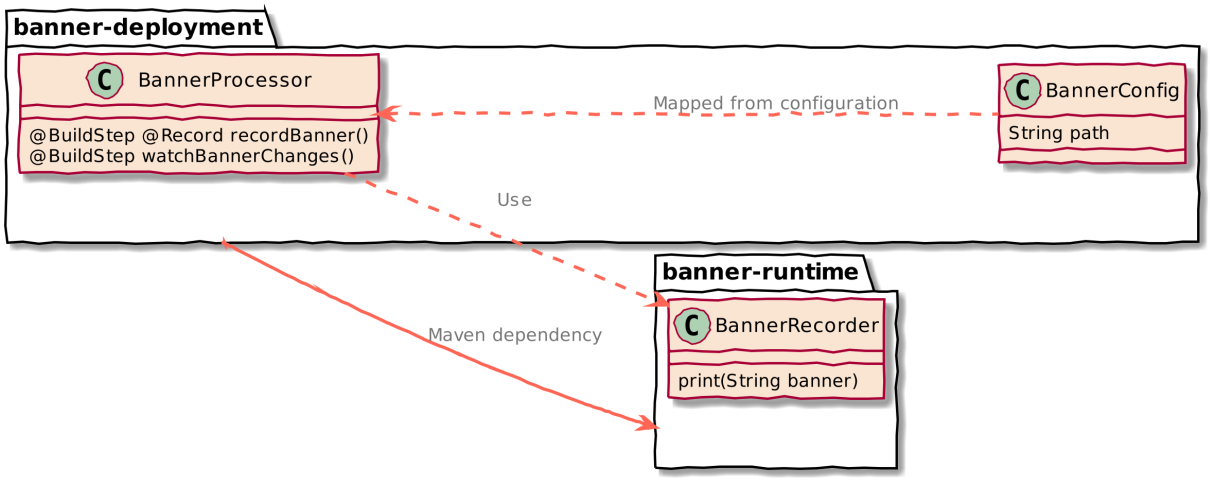



Writing a Quarkus Extension

# Structure of an extension

As stated above, an extension is divided into 2 parts, called `deployment` (augmentation) and `runtime`.




From the directory extensions/extension-banner execute the following commands:

```
mkdir -p deployment/src/main/java
mkdir -p deployment/src/main/resources
mkdir -p runtime/src/main/java
mkdir -p runtime/src/main/resources

echo "<project xmlns='http://maven.apache.org/POM/4.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
         xsi:schemaLocation='http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd'>
    <modelVersion>4.0.0</modelVersion>

    <groupId>io.quarkus.workshop.super-heroes</groupId>
    <artifactId>extension-banner-parent</artifactId>
    <version>1.0</version>
    <packaging>pom</packaging>
    <name>Quarkus Workshop :: Extensions :: Banner Extension</name>

    <modules>
        <module>runtime</module>
        <module>deployment</module>
    </modules>

     <dependencyManagement>
        <dependencies>
```

```
            <dependency>
                <groupId>io.quarkus</groupId>
                <artifactId>quarkus-bom</artifactId>
                <version>\${quarkus.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <properties>
        <quarkus.version>1.7.0.Final</quarkus.version>
        <surefire-plugin.version>2.22.0</surefire-plugin.version>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.build.timestamp.format>yyyy-MM-dd</maven.build.timestamp.format>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
" > pom.xml

echo "<project xmlns='http://maven.apache.org/POM/4.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
         xsi:schemaLocation='http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd'>
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>io.quarkus.workshop.super-heroes</groupId>
        <artifactId>extension-banner-parent</artifactId>
        <version>1.0</version>
    </parent>

    <artifactId>extension-banner-deployment</artifactId>
    <name>Quarkus Workshop :: Extensions :: Banner Extension :: Deployment</name>

    <dependencies>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-core-deployment</artifactId>
            <version>\${quarkus.version}</version>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-arc-deployment</artifactId>
            <version>\${quarkus.version}</version>
        </dependency>
        <dependency>
            <groupId>io.quarkus.workshop.super-heroes</groupId>
            <artifactId>extension-banner</artifactId>
            <version>\${project.version}</version>
        </dependency>

        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-junit5-internal</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>io.quarkus</groupId>
                            <artifactId>quarkus-extension-processor</artifactId>
                            <version>\${quarkus.version}</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M4</version>
                <configuration>
                    <systemProperties>
                        <java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
                    </systemProperties>
```

```
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>" > deployment/pom.xml

echo "<project xmlns='http://maven.apache.org/POM/4.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
         xsi:schemaLocation='http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd'>
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>io.quarkus.workshop.super-heroes</groupId>
        <artifactId>extension-banner-parent</artifactId>
        <version>1.0</version>
    </parent>

    <artifactId>extension-banner</artifactId>
    <name>Quarkus Workshop :: Extensions :: Banner Extension :: Runtime</name>

    <dependencies>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-core</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-arc</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>io.quarkus</groupId>
                <artifactId>quarkus-bootstrap-maven-plugin</artifactId>
                <version>\${quarkus.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>extension-descriptor</goal>
                        </goals>
                        <phase>compile</phase>
                        <configuration>
                            <deployment>\${project.groupId}:\${project.artifactId}-deployment:\${project.version}</deployment>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>io.quarkus</groupId>
                            <artifactId>quarkus-extension-processor</artifactId>
                            <version>\${quarkus.version}</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>" > runtime/pom.xml
```

The Runtime module

The *runtime* part of an extension contains only the classes and resources required at runtime. For the banner extension, it would be a single class that prints the banner.

In the runtime module, creates the `io.quarkus.workshop.superheroes.banner.runtime.BannerRecorder` class with the following content:

```
package io.quarkus.workshop.superheroes.banner.runtime;

import io.quarkus.runtime.annotations.Recorder;

@Recorder
public class BannerRecorder {

    public void print(String banner) {
        System.err.println(banner);
    }
}
```

## The deployment module

This module contains *build steps*, i.e., methods called during the augmentation phase and computing just enough bytecode to serve the services the application requires. For the banner extension, it consists of two build steps:

The first build step is going to read the banner file and use the `BannerRecorder`The second build step is related to the dev mode and triggers a hot-reload when the content of the banner file changes.

In the deployment module, create the io.quarkus.workshop.superheroes.banner.deployment.BannerConfig with the following content:

```
package io.quarkus.workshop.superheroes.banner.deployment;

import io.quarkus.runtime.annotations.ConfigItem;
import io.quarkus.runtime.annotations.ConfigPhase;
import io.quarkus.runtime.annotations.ConfigRoot;

@ConfigRoot(name = "banner", phase = ConfigPhase.BUILD_TIME)
public class BannerConfig {

    /**
     * The path of the banner.
     */
    @ConfigItem public String path;
}
```

Create the io.quarkus.workshop.superheroes.banner.deployment.BannerProcessor class with the following content:

```
package io.quarkus.workshop.superheroes.banner.deployment;

import io.quarkus.deployment.annotations.BuildStep;
```

```
import io.quarkus.deployment.annotations.ExecutionTime;
import io.quarkus.deployment.annotations.Record;
import io.quarkus.deployment.builditem.HotDeploymentWatchedFileBuildItem;
import io.quarkus.deployment.util.FileUtil;
import io.quarkus.workshop.superheroes.banner.runtime.BannerRecorder;

import java.io.IOException;
import java.io.InputStream;
import java.io.UncheckedIOException;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

public class BannerProcessor {

    @BuildStep
    @Record(ExecutionTime.RUNTIME_INIT)
    public void recordBanner(BannerRecorder recorder, BannerConfig config) {
        String content = readBannerFile(config.path);
        recorder.print(content);
    }

    @BuildStep
    List<HotDeploymentWatchedFileBuildItem> watchBannerChanges(BannerConfig config) {
        List<HotDeploymentWatchedFileBuildItem> watchedFiles = new ArrayList<>();
        watchedFiles.add(new HotDeploymentWatchedFileBuildItem((config.path)));
        return watchedFiles;
    }

    private String readBannerFile(String path) {
        URL resource = Thread.currentThread().getContextClassLoader().getResource(path);
        if (resource != null) {
            try (InputStream is = resource.openStream()) {
                byte[] content = FileUtil.readFileContents(is);
                return new String(content, StandardCharsets.UTF_8);
            } catch (IOException e) {
                throw new UncheckedIOException(e);
            }
        } else {
            throw new IllegalArgumentException("Cannot find the banner file: " + path);
        }
    }

}
```

```
mvn clean install
```

Using the extension

Go back to the fight microservice, and add the following dependency to the pom.xml file:

```
<dependency>
    <groupId>io.quarkus.workshop.super-heroes</groupId>
    <artifactId>extension-banner</artifactId>
    <version>1.0</version>
</dependency>
```

Containers & Cloud

create openshift cluster

```
cd openshift-install-mac
./openshift-install create install-config --dir=<dir-name>
./openshift-install create cluster --dir=<dir-name>



INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/Users/nag/Downloads/openshift-install-m
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.quarkusworkshop.nagcloudlab.com
INFO Login to the console with user: "kubeadmin", and password: "B5B8o-bIwzH-BCQbs-5a7DV"
```

From bare metal to containers

```
cd rest-hero
mvn clean package -Pnative -Dnative-image.docker-build=true -DskipTests
cd ..
cd rest-villain
mvn clean package -Pnative -Dnative-image.docker-build=true -DskipTests
cd ..
cd rest-fight
cp -R ../ui-super-heroes/dist/* src/main/resources/META-INF/resources
mvn clean package -Pnative -Dnative-image.docker-build=true -DskipTests
cd ..
cd event-statistics
mvn clean package -Pnative -Dnative-image.docker-build=true -DskipTests
cd ..
```

Building containers

```
export ORG=xxxx
cd rest-hero
docker build -f src/main/docker/Dockerfile.native -t $ORG/quarkus-workshop-hero .
cd ..
cd rest-villain
docker build -f src/main/docker/Dockerfile.native -t $ORG/quarkus-workshop-villain .
cd ..
cd rest-fight
docker build -f src/main/docker/Dockerfile.native -t $ORG/quarkus-workshop-fight .
cd ..
cd event-statistics
docker build -f src/main/docker/Dockerfile.native -t $ORG/quarkus-workshop-stats .
cd ..
```