

Kubernetes Native Developer

Architecture Workshop

Working with Quarkus

<https://github.com/wpernath/quarkus-grumpycat>



linkedin.com/company/red-hat



facebook.com/redhatinc



youtube.com/user/RedHatVideos



twitter.com/RedHat

Self introduction

Name: Wanja Pernath

Email: wpernath@redhat.com

Base: Germany (very close to the Alps)

Role: EMEA Technical Partner Development Manager

- OpenShift and MW

Experience: Years of Consulting, Training, PreSales at
Red Hat and before

Twitter: <https://twitter.com/wpernath>

LinkedIn: <https://www.linkedin.com/in/wanjapernath/>

GitHub: <https://github.com/wpernath>



First book just published

Getting GitOps

A technical blueprint for developing with Kubernetes and OpenShift based on a REST microservice example written with Quarkus

Technologies discussed:

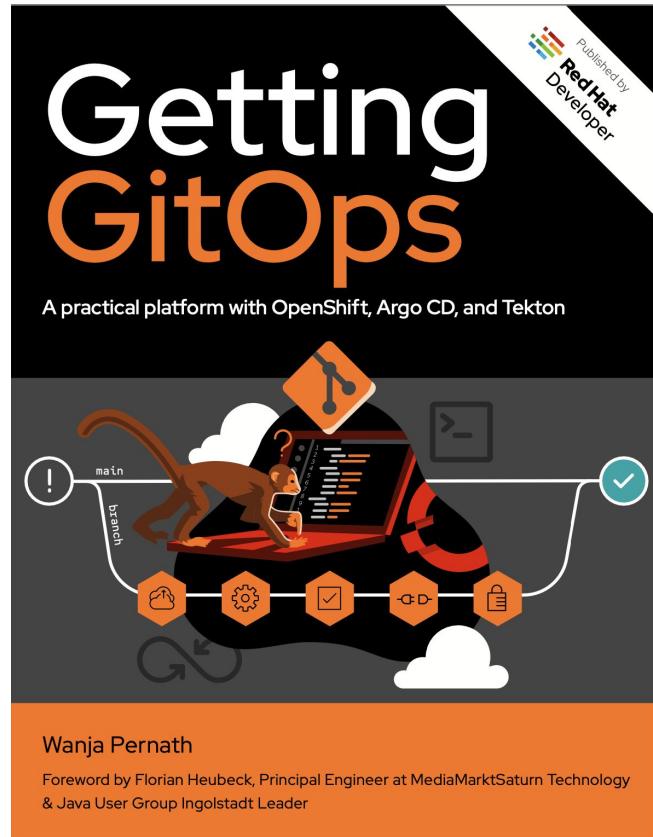
Quarkus, Helm Charts, Kustomize, Tekton Pipelines, Kubernetes Operators, OpenShift Templates, ArgoCD, CI/CD, GitOps....

Download for free at:

<https://developers.redhat.com/e-books/getting-gitops-practical-platform-openshift-argo-cd-and-tekton>

Interview with full GitOps Demo:

https://www.youtube.com/watch?v=znMfVqAIRzY&ab_channel=OpenShift



Agenda

Agenda

- **Quarkus Concepts**
 - Basics
 - CDI
 - Dev Services
 - Extensions
- **Database / CRUD REST services**
- **Messaging (smallrye.io)**
 - WebSockets
 - Kafka
- **Quie Templating Engine**
- **Container Images**
 - Kubernetes
 - Health
 - Metrics
- **Security**

Quarkus Concepts

Concepts

- **Runtime & CDI Framework**
- Runtime based on Eclipse Vert.x & Netty
 - Vert.x is a reactive, resource-efficient, concurrent & asynchronous runtime environment (think about NodeJS for Java)
- Eclipse MicroProfile based Framework
 - With use of SmallRye, Mutiny components for Reactive and Messaging
- Extension based!
 - You don't need no Servlets? – Don't use them
 - No JPA required? – Do not include jpa extension
- Can be used for CLI & “web” apps

Context & Dependency Injection (CDI)

```
import javax.inject.Inject;
import javax.enterprise.context.ApplicationScoped;
import org.eclipse.microprofile.metrics.annotation.Counted;

@ApplicationScoped ①
public class Translator {

    @Inject
    Dictionary dictionary; ②

    @Counted ③
    String translate(String sentence) {
        // ...
    }
}
```

- ① This is a scope annotation. It tells the container which context to associate the bean instance with. In this particular case, a **single bean instance** is created for the application and used by all other beans that inject `Translator`.
- ② This is a field injection point. It tells the container that `Translator` depends on the `Dictionary` bean. If there is no matching bean the build fails.
- ③ This is an interceptor binding annotation. In this case, the annotation comes from the MicroProfile Metrics. The relevant interceptor intercepts the invocation and updates the relevant metrics. We will talk about [interceptors](#) later.

Context & Dependency Injection (CDI)

- Quarkus is based on CDI 2.0 programming model
 - With some limitations
 - Read <https://quarkus.io/guides/cdi>
- Quarkus has its own CDI Reference Guide, which you can find here: <https://quarkus.io/guides/cdi-reference>

Runtime - Vert.x

- Quarkus is using Eclipse Vert.x underneath, a toolkit for building reactive applications (or NodeJS concepts for Java)
- Vert.x was originally invented by some former JBoss'ians and is now an Eclipse project
- **Important:** Although Quarkus is using Vert.x as it's runtime, you do NOT have to be reactive in your code (but you always could).

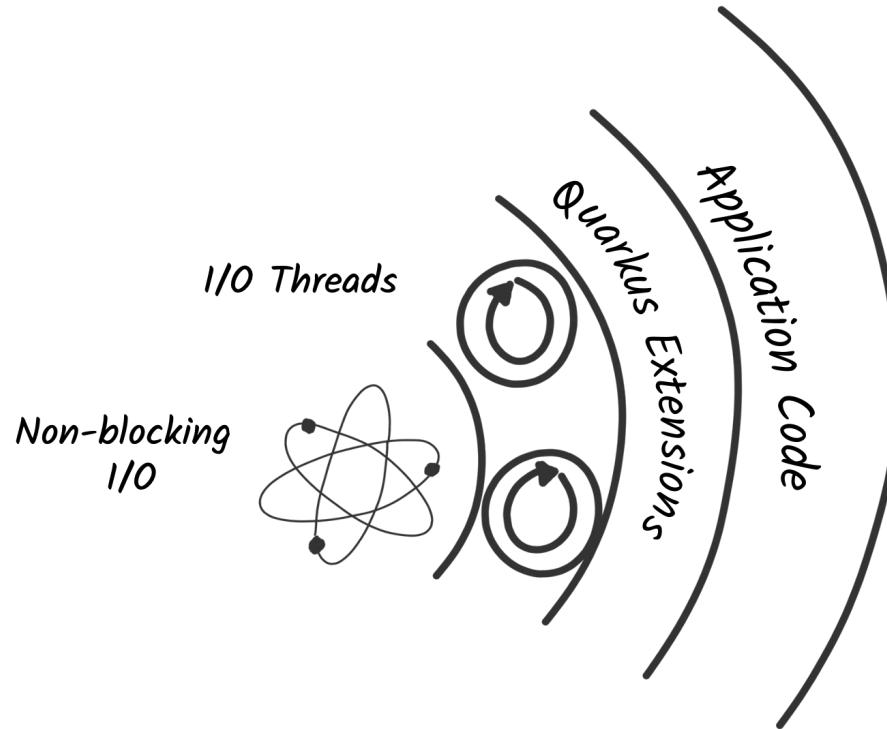
Runtime - Vert.x

Quarkus provides a managed Vertx instance, which you could always directly use, if you have the need for it. – But you typically don't need to

Quarkus is hiding the complexity for you.

<https://quarkus.io/guides/vertx>

<https://quarkus.io/guides/vertx-reference>



Extensions based

- Quarkus is organized in extensions. There is a core one (which you already have by using the project initialization of Quarkus). And then there are extensions sorted by use case.
 - REST
 - ORM
 - Messaging
 - Security (quickly integrate Keycloak?)
 - Kubernetes, Container-Images, Helm-Charts
 - Etc.
- All public extensions have an own how-to guide

The screenshot shows the Quarkus extension catalog interface. At the top, it displays the Quarkus logo and version 2.13. It includes a back button, a link to quarkus.io, and an 'Available with Enterprise Support' badge. Below this is a search bar with the placeholder 'origin:platform'. The main area is titled 'CONFIGURE YOUR APPLICATION' and shows a list of extensions grouped under 'Web'. The extensions listed are:

- RESTEasy Reactive (quarkus-resteasy-reactive) [STARTER CODE]
- RESTEasy Jackson (quarkus-resteasy-reactive-jackson) [STARTER CODE]
- RESTEasy JSON-B (quarkus-resteasy-reactive-json) [STARTER CODE]
- RESTEasy JAXB (quarkus-resteasy-reactive-jaxb) [STARTER CODE]
- RESTEasy Reactive Kotlin Serialization (quarkus-resteasy-reactive-kotlin-serialization) [STARTER CODE]
- RESTEasy Reactive Quarkus (quarkus-resteasy-reactive-quarkus) [STARTER CODE]
- RESTEasy Reactive Links (quarkus-resteasy-reactive-links) [STARTER CODE]
- REST Client Reactive (quarkus-rest-client-reactive) [STARTER CODE]
- REST Client Reactive Jackson (quarkus-rest-client-reactive-jackson) [STARTER CODE]
- REST Client Reactive JSON-B (quarkus-rest-client-reactive-json) [STARTER CODE]
- REST Client Reactive JAXB (quarkus-rest-client-reactive-jaxb) [STARTER CODE]
- REST Client Reactive Kotlin Serialization (quarkus-rest-client-reactive-kotlin-serialization) [STARTER CODE]
- RESTEasy Classic (quarkus-resteasy) [STARTER CODE]
- RESTEasy Classic Jackson (quarkus-resteasy-jackson) [STARTER CODE]
- RESTEasy Classic JSON-B (quarkus-resteasy-json) [STARTER CODE]
- RESTEasy Classic JAXB (quarkus-resteasy-jaxb) [STARTER CODE]

Each extension entry includes a 'STARTER CODE' button and a brief description of its compatibility with other extensions.

Extensions based

- Want to create a REST CRUD service with PostgreSQL?
 - Create your Quarkus skeleton
 - Add extensions to your code
 - Hibernate ORM
 - PostgreSQL JDBC driver
 - Add jackson / json-b
 - Create the Entity class
 - Create the REST service
 - Configure your service
 - Run quarkus dev
- You're done within 5 minutes!

```
[wanja@minimac ~/Devel/tmp]$ quarkus create app com.redhat.demo:demo-service:0.0.1
-----
applying codestarts...
└── java
    ├── maven
    └── quarkus
        ├── config-properties
        ├── dockerfiles
        ├── maven-wrapper
        └── resteasy-reactive-codestart

-----
[SUCCESS] ✅ quarkus project has been successfully generated in:
--> /Users/wanja/Devel/tmp/demo-service

-----
Navigate into this directory and get started: quarkus dev
[wanja@minimac ~/Devel/tmp]$ cd demo-service
[wanja@minimac ~/Devel/tmp/demo-service]$ quarkus ext add resteasy-reactive-jackson hibernate-orm-panache jdbc-postgresql
[SUCCESS] ✅ Extension io.quarkus:quarkus-resteasy-reactive-jackson has been installed
[SUCCESS] ✅ Extension io.quarkus:quarkus-hibernate-orm-panache has been installed
[SUCCESS] ✅ Extension io.quarkus:quarkus-jdbc-postgresql has been installed
[wanja@minimac ~/Devel/tmp/demo-service]$ Listening for transport dt_socket at address: 5005
2022-10-05 20:13:28,007 WARN [io.qua.hib.orm.dep.HibernateOrmProcessor] (build-27) Hibernate ORM is disabled because no JPA entities were found
2022-10-05 20:13:29,516 INFO [io.qua.dat.dep.dev.DevServicesDatasourceProcessor] (build-20) Dev Services for the default datasource (postgresql) started.
2022-10-05 20:13:29,517 INFO [io.qua.hib.orm.dep.HibernateOrmProcessor] (build-48) Setting quarkus.hibernate-orm.database.generation=drop-and-create to initialize Dev Services managed database
2022-10-05 20:13:29,583 WARN [io.net.res.dns.DnsServerAddressStreamProviders] (build-34) Can not find io.netty.resolver.dns.macos.MacOSDnsServerAddressStreamProvider in the classpath, fallback to system defaults. This may result in incorrect DNS resolutions on MacOS.
```
```
2022-10-05 20:13:29,962 INFO [io.quarkus] (Quarkus Main Thread) demo-service 0.0.1 on JVM (powered by Quarkus 2.13.0.Final) started in 2.350s. Listening on: http://localhost:8080
```

Quarkus Dev Services

- Most services you require during development provide a Dev Service environment via the extension
- Example services are:
 - Databases
 - AMQP
 - Kafka
 - Keycloak
 - Redis, MongoDB
 - RabbitMQ
 - Vault
 - Infinispan
 - Elasticsearch



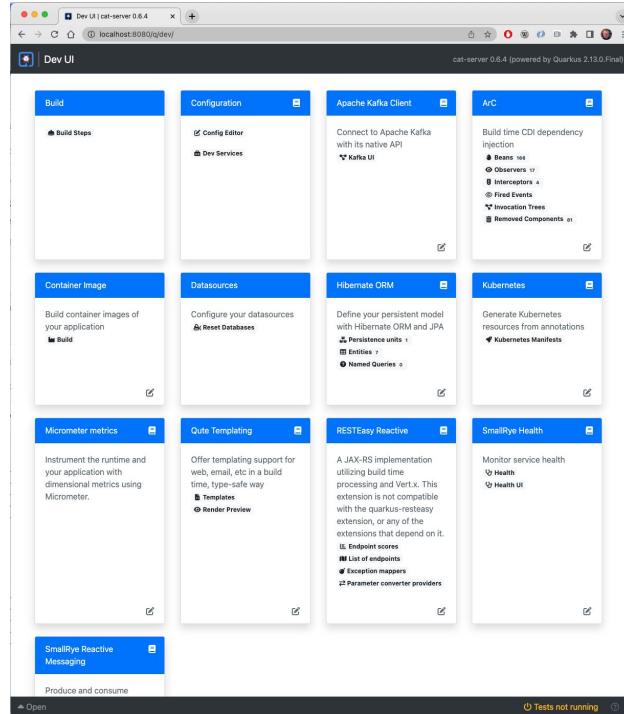
Quarkus Dev Services

- DevServices require a working Docker Desktop environment (or a running Podman Desktop)
- Benefits?
 - Focus on what you need and implement it
 - Don't worry about local installation of PostgreSQL, Kafka... this comes for free
 - Speed up in Dev
- Configuration via application.properties
 - Use **%prod.quarkus...** to define the location of the service in Quarkus PROD environments
 - Use **\$(ENV_NAME)** to build your value

```
22 #  
23 # Datasource options  
24 #  
25 You, 4 months ago  
26 quarkus.datasource.db-kind=postgresql  
wpernath, 2 months ago  
27 quarkus.hibernate-orm.log.sql=false  
quarkus.hibernate-orm.log.format-sql=true  
28  
29 # the following props only for production  
You, 2 months ago  
30 %prod.quarkus.hibernate-orm.log.sql=false  
31 %prod.quarkus.hibernate-orm.log.format-sql=false  
You, 4 months ago  
32 %prod.quarkus.hibernate-orm.database.generation=update  
You, 4 months ago  
33 %prod.quarkus.datasource.username=${DB_user:cat}  
34 %prod.quarkus.datasource.password=${DB_password:grumpy}  
You, 4 months ago  
35 %prod.quarkus.datasource.jdbc.url=jdbc:postgresql://${DB_host:catserver}/${DB_dbname:catdb} You, 4 months ago • changed app
```

Quarkus Dev UI

- Quarkus provides a Dev UI, available on /q/dev if the app runs in development mode
- This UI is extendable via extensions
- An extension might provide monitoring here (Kafka UI, Health UI etc.) or some configuration stuff (ConfigEditor)
- But NOTE: The UI is only available in Dev mode, i.e. if the app is started via “quarkus dev”



<https://quarkus.io/guides/dev-services>

Working with a Database

```
26      <dependency>
27          <groupId>io.quarkus</groupId>
28          <artifactId>quarkus-hibernate-orm-panache</artifactId>
29      </dependency>
30
```

```
31      <dependency>
32          <groupId>io.quarkus</groupId>
33          <artifactId>quarkus-jdbc-postgresql</artifactId>
34      </dependency>
```

Using Hibernate ORM / JPA

- Quarkus makes it easy to create a CRUD service, all you need to do is to add the **hibernate-orm** or **hibernate-orm-panache** extension to your app
- And you of course need to add the extension with the jdbc driver of your database
- If you have to, you can use hibernate-orm extension with full flexibility and control of the JPA process
- If you don't require full access, use simplified ORM via panache
- Most common databases are part of the Dev Services of Quarkus

```
10  @Entity
11  public class Score extends PanacheEntity {
12      @Column(name = "player_id")
13      public Long playerId;
14
15      @Column(name = "game_id")
16      public Long gameId;
17      public long score;
18      public int level;
19
20      @Column(name = "placed_barriers")
21      public int placedBarriers;
22
23      @Column(name = "used_bombs")
24      public int usedBombs;
25
26      @Column(name = "bitten_by_spiders")
27      public int bittenBySpiders;
28
29      @Column(name = "catched_by_cats")
30      public int catchedByCats;
31
```

Using Hibernate ORM / JPA

- Panache makes it possible that you don't require getters / setters for your entities, they're implicitly being created during Quarkus augmentation process
- The REST service looks clean and lean as well
- You have two different ways of using the Entity:
 - **Active Record Pattern:** Here we are using the Active Record Pattern, all access methods are defined as static methods in the entity itself
 - **Repository Pattern:** You can also define an EntityRepository, which encapsulates all accessors

```
14 @Path("/highscore")
15 public class HighScoreResource {
16
17     @GET
18     @Path("/{highestX}")
19     http://localhost:8080/highscore/{highestX}
20     public List<Score> readHighscore(int highestX) {
21         List<Score> list = Score.list(query: "order by score desc, time");
22         if( highestX > 0 && list.size() > highestX) {
23             list = list.subList(fromIndex: 0, highestX);
24         }
25         return list;
26     }
27
28     @POST
29     @Transactional
30     public List<Score> addScore(Score score) {
31         score.id = null;
32         if( score.time == null ) score.time = new Date();
33         score.persist();
34         return readHighscore(highestX: 10);
35     }
36
37     @DELETE
38     @Transactional
39     @Path("/{id}")
40     public List<Score> deleteScore(Long id) {
41         Score.deleteById(id);
42         return readHighscore(highestX: 10);
43     }
}
```

Using Hibernate ORM / JPA

- The only thing you really have to do is, configuring your datasource by specifying properties in **application.properties**
- If you want to use the database dev service, you need to use a property profile (%prod.) as prefix here
- By using \${ENV_NAME:default} you are able to configure the configuration
 - This is useful if you want to deploy your app to Kubernetes and you're using an environment mapped Secret

```
22 #  
23 # Datasource options  
24 #  
25 You, 4 months ago  
26 quarkus.datasource.db-kind=postgresql  
wpernath, 2 months ago  
27 quarkus.hibernate-orm.log.sql=false  
quarkus.hibernate-orm.log.format-sql=true  
28  
29 # the following props only for production  
You, 2 months ago  
30 %prod.quarkus.hibernate-orm.log.sql=false  
31 %prod.quarkus.hibernate-orm.log.format-sql=false  
You, 4 months ago  
32 %prod.quarkus.hibernate-orm.database.generation=update  
You, 4 months ago  
33 %prod.quarkus.datasource.username=${DB_user:cat}  
34 %prod.quarkus.datasource.password=${DB_password:grumpy} You, 4 months ago * changed ap  
35 %prod.quarkus.datasource.jdbc.url=jdbc:postgresql://${DB_host:catserver}/${DB_dbname:catdb}
```

Qute Templating Engine

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-reactive-qute</artifactId>
</dependency>
```

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-qute</artifactId>
</dependency>
```

Qute Templating Engine

- Qute is an easy to use templating engine for Quarkus. It's designed especially for Quarkus:
 - Minimizes reflection (for native compilation)
 - Contains imperative and non-blocking APIs
 - Understands Dev Services (templates are being watched for changes during development)
 - Syntax & runtime checks during build time
- Qute has two parts
 - The template file itself, located under `src/main/resources/templates`
 - The Java part

```
13  @Path("/reports")
14  public class ReportResource {
15
16      Open `src/main/resources/templates/openGames.txt`
17      @Inject
18      Template openGames;
19
20      @Inject
21      MultiPlayerResource multiPlayerResource;
22
23      @Path("/open-games")
24      @GET
25      @Produces(MediaType.TEXT_PLAIN)
26      @Blocking
27      http://localhost:8080/reports/open-games
28      public TemplateInstance listOpenGames() {
29          return openGames.data(
30              key: "games",
31              multiPlayerResource.listOpenGames()
32          );
33      }
```

```
ReportResource#openGames | games : List<MultiPlayerGame>
1  This report lists all currently open games
2
3  ID,Time,player1name,player2name,player3name,player4name
4  {#for game:MultiPlayerGame in games}
5      {game.id},{game.timeStarted},{game.player1.name},{game.player2 ? game.player2.name : "Unused"}
6  {/for}
```

Qute Templating Engine - The Java part

- You could use it through a REST service (other use cases are possible, for example via Scheduler)
- All you need to do is:
 - @Inject a Template object
 - The object's name is the name of the template file (so openGames needs to have a corresponding openGames.txt file)
 - The GET method of the service is returning a TemplateInstance, which you can prefill with parameters.
 - Parameters are key/value pairs

```
13  @Path("/reports")
14  public class ReportResource {
15
16      Open `src/main/resources/templates/openGames.txt`
17      @Inject
18      Template openGames;
19
20      @Inject
21      MultiPlayerResource multiPlayerResource;
22
23      @Path("/open-games")
24      @GET
25      @Produces(MediaType.TEXT_PLAIN)
26      @Blocking
27      http://localhost:8080/reports/open-games
28      public TemplateInstance listOpenGames() {
29          return openGames.data(
30              key: "games",
31              multiPlayerResource.listOpenGames()
32          );
33      }
34  }
```

Quie Templating Engine - The template file

- By default, all templates are in `/src/main/resources/templates` (could be overwritten via `application.properties`).
- All templates need to have a suffix like `".txt"`, `".html"` (could be overwritten via `application.properties`).
- A template has access to all parameters provided via key/value
- A template has also access to all CDI beans available globally in the quarkus application
- Comprehensive syntax to render the output

```
ReportResource#openGames | games : List<MultiPlayerGame>
1 This report lists all currently open games
2
3 ID,Time,player1name,player2name,player3name,player4name
4 {#for game:MultiPlayerGame in games}
5   {game.id},{game.timeStarted},{game.player1.name},{game.player2 ? game.player2.name : "Unused"}
6 {/for}
```

Quie Templating Engine - The template file

- Quie language is easy and contains the following parts
 - **Comments:** {! this is comment !}
 - **Expressions:** {foo.name} outputs the evaluated value
 - **Sections:** A section may contain static text, expressions and more sections. {#if foo.isActive } {foo.name} {/if}
 - **Unparsed data:** Marks content that should be rendered but not parsed. {! alert("hi"); !}

```
ReportResource#openGames | games : List<MultiPlayerGame>
1  {! this is a comment !}
2  Report for: {config:['quarkus.application.name']} {config:['application.version']}
3
4  This report lists all currently open games
5  {! the same as !}
6  {!This section is not parsed by quie!}
7
8  {#if !games.isEmpty }
9  ID,Time,player1name,player2name,player3name,player4name
10
11 {#each games}
12   {#let game:MultiPlayerGame =it}
13   {game.id},{game.timeStarted},{game.player1.name},{game.player2 ? game.player2.name : "Unused"}
14   {/let}
15 {/each}
16
17
18 {#for game:MultiPlayerGame in games}
19   this could be done as well, instead of #each ..
20 {/for}
21
22 {#else}
23 There are NO open games available
24 {/if}
25
```

Qute Templating Engine - Advanced features

- **Template Extension Methods:** A data class can be extended with new functionality, w/o changing the original Java class
- **Template Global** variables: A class annotated with @TemplateGlobal contains variables which ALL TemplateInstances can use out of the box
- **Modularize your templates** by using includes and user defined tags

```
package org.acme;

class Item {

    public final BigDecimal price;

    public Item(BigDecimal price) {
        this.price = price;
    }
}

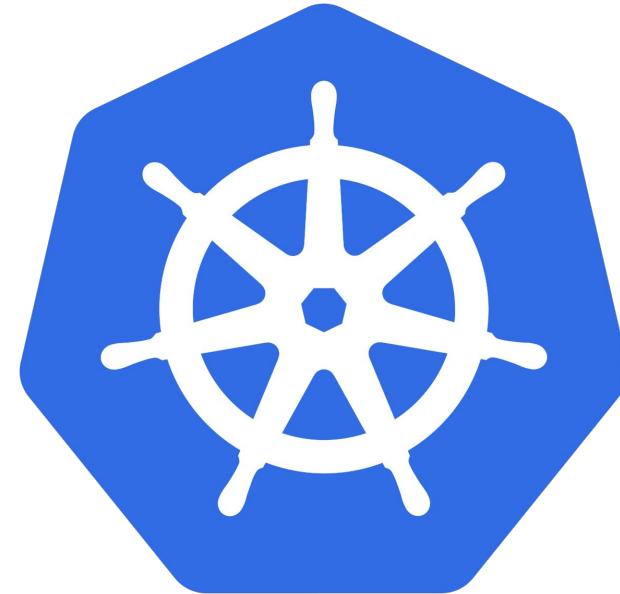
@TemplateExtension
class MyExtensions {

    static BigDecimal discountedPrice(Item item) { ①
        return item.getPrice().multiply(new BigDecimal("0.9"));
    }
}
```

5 Steps to Kubernetes & Container Images

Kubernetes Native Coding

- One of the main goals of Quarkus is to make it easy and user friendly to develop for Kubernetes or Docker / Podman
- So there are several extensions available to reach this goal
 - quarkus-container-image-*
 - quarkus-kubernetes
 - quarkus-kubernetes-config
 - quarkus-openshift
 - quarkus-smallrye-health
 - Smallrye-metrics / micrometer-metrics
 - Centralized log management



Step #1 - Creating container images

- There are several ways to create a container image. The most obvious way is to manually call “docker build...” on one of the predefined Dockerfiles
- But there are better ways via maven by including one of the quarkus-container-image-* extension
 - Jib (my preferred choice)
 - Docker (does NOT work inside Kubernetes)
 - S2i (OpenShift only)
 - Buildpack (standardized s2i)
- Creating and pushing a container is just a matter of calling maven now → This can even be done inside Kubernetes

```
13 #
14 # container image properties
15 #
16 You, 4 months ago
16 quarkus.container-image.image=quay.io/wpernath/quarkus-grumpycat:v${quarkus.application.version}
17 quarkus.container-image.builder=jib
18 quarkus.container-image.build=false
19 quarkus.container-image.push=false
```

```
[wanja@minimac ~/Devel/grumpycat/quarkus-grumpycat/quarkus-server]$ mvn clean package -Dquarkus.container-image.push=true
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building cat-server 0.6.4
[INFO] ------------------------------------------------------------------------
[INFO]
```

```
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Starting (local) container image build for jar using jib.
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] No container image registry was set, so 'docker.io' will be used
[WARNING] [io.quarkus.container.image.jib.deployment.JibProcessor] Base image 'registry.access.redhat.com/ubi8/openshift-11-runtime:1.11' does not use a specific image digest - build may not be reproducible
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] LogEvent [level=INFO, message=trying docker-credential-desktop for quay.io]
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] LogEvent [level=LIFECYCLE, message=Using credentials from Docker config (/Users/wanja/.docker/config.json) for quay.io/wpernath/quarkus-grumpycat:v0.6.4]
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Using base image with digest: sha256:0aca47bf03430b5e033d67ba9b38871470a700fa7d80a3cc0ae34270935
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Container entrypoint set to [java, -Djava.util.logging.manager=org.jboss.logmanager, -jar, quarkus-run.jar]
[INFO] [io.quarkus.container.image.jib.deployment.JibProcessor] Pushed container image quay.io/wpernath/quarkus-grumpycat:v0.6.4 (sha256:378a5ca73390e9b67e3fdcb53f5623d17bb9e2f2e83e20aaef9bff8cd6eb37e2)
```

Step #2 - Creating Kubernetes files

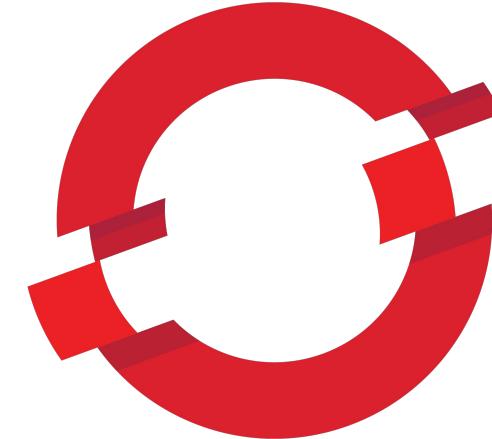
- In order to properly deploy a container image on Kubernetes, you need to define some manifest files
 - Deployment, Service, Route, Ingress, ConfigMap... etc.
- Those files can easily be generated during build by using the **quarkus-kubernetes** extension
- If you're placing some additional files in `/src/main/kubernetes`, they'll be merged with quarkus own generations and placed in `/target/kubernetes/` folder
- You can configure the extension by specifying properties `quarkus.kubernetes.*` in `application.properties` file

```
1  ---
2  apiVersion: route.openshift.io/v1
3  kind: Route
4  metadata:
5    annotations:
6      app.openshift.io/connects-to: >-
7        [{"apiVersion": "apps/v1", "kind": "Deployment", "name": "cat-server"}]
8  labels:
9    app: cat-server
10   app.kubernetes.io/component: cat-server
11   app.kubernetes.io/instance: cat-server
12   app.kubernetes.io/name: cat-server
13   app.kubernetes.io/part-of: grumpycat-app
14   name: cat-server
15 spec:
16   port:
17     targetPort: 8080-tcp
18   to:
19     kind: Service
20     name: cat-server
21     weight: 100
22   wildcardPolicy: None
23   ...
24   apiVersion: apps/v1
25   kind: Deployment
```

```
[INFO] Adding existing Service with name: cat-server.
[INFO] Adding existing Deployment with name: cat-server.
[INFO] Adding existing PostgresCluster with name: cat.
[INFO] Adding existing Kafka with name: grumpy-kafka.
[INFO] Adding existing KafkaTopic with name: player-actions.
[INFO] Adding existing Route with name: cat-server.
```

Step #2 - Creating OpenShift files

- If you really require OpenShift specific things, you can use the **quarkus-openshift** extension instead of the **quarkus-kubernetes** one.
- However, this is not required when you're targeting OpenShift as Kubernetes distribution
- This extension makes it easy to directly deploy to OpenShift and to make use of s2i inside the cluster
- However, by specifying a Route in src/main/kubernetes, you can also easily use the kubernetes extension



Optional step #2 - Directly use Kubernetes Secrets / ConfigMaps

- By default, you can mount any ConfigMap / Secret to your Pod's environment, either as ENV or as volume.
- If you want to directly use any Secret or ConfigMap as Config Source in your Quarkus Application, you can use the **quarkus-kubernetes-config** extension
- This extension allows you to directly use any key of the Secret / ConfigMap directly in application.properties (and thus of course also in your code, via @ConfigProperty)

One possible way to make Quarkus use these entries to connect the database is to use the following configuration:

```
%prod.quarkus.kubernetes-config.secrets.enabled=true  
quarkus.kubernetes-config.secrets=postgres  
  
%prod.quarkus.datasource.jdbc.url=postgresql://somehost:5432/${database-name}  
%prod.quarkus.datasource.username=${database-user}  
%prod.quarkus.datasource.password=${database-password}
```

- 1 Enable reading of secrets. Note the use of **%prod** profile as we only want this setting applied when the application is running in production.
- 2 Configure the name of the secret that will be used. This doesn't need to be prefixed with the **%prod** profile as it won't have any effect if secret reading is disabled.
Quarkus will substitute **\${database-name}** with the value obtained from the entry with name **database-name** of the **postgres** Secret. **somehost** is the name of the Kubernetes **Service** that was created when PostgreSQL was deployed to Kubernetes.
- 3 Quarkus will substitute **\${database-user}** with the value obtained from the entry with name **database-user** of the **postgres** Secret.
- 4 Quarkus will substitute **\${database-password}** with the value obtained from the entry with name **database-password** of the **postgres** Secret.

Step #3 - Adding Health Checks

- In a Kubernetes Native environment, it's essential to have Health checks implemented accordingly to make it easy for the Kubernetes scheduler to see if your app is ready to be used or needs to be killed / restarted.
- For this there is the **smallrye-health** extension available. In combination with the kubernetes extension, the created Deployment will already contain the corresponding checks automatically
- Other extensions (like messaging and databases) will automatically add checks here

<https://quarkus.io/guides/smallrye-health>

Importing the **smallrye-health** extension directly exposes three REST endpoints:

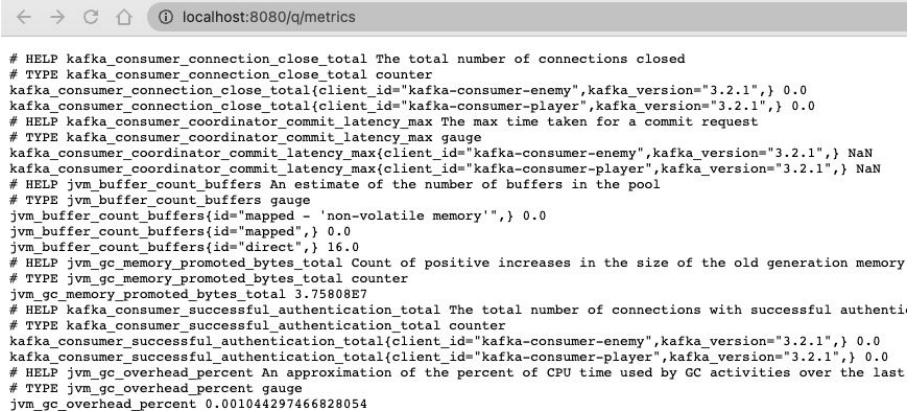
- `/q/health/live` - The application is up and running.
- `/q/health/ready` - The application is ready to serve requests.
- `/q/health/started` - The application is started.
- `/q/health` - Accumulating all health check procedures in the application.

```
74      readinessProbe:
75        httpGet:
76          path: /q/health/ready
77          port: 8080
78          scheme: HTTP
79          initialDelaySeconds: 5
80          timeoutSeconds: 1
81          periodSeconds: 10
82          successThreshold: 1
83          failureThreshold: 5
84      livenessProbe:
85        httpGet:
86          path: /q/health/live
87          port: 8080
88          scheme: HTTP
89          initialDelaySeconds: 10
90          timeoutSeconds: 1
91          periodSeconds: 10
92          successThreshold: 1
93          failureThreshold: 3
94      startupProbe:
95        httpGet:
96          path: /q/health/started
97          port: 8080
98          scheme: HTTP
99          initialDelaySeconds: 10
100         timeoutSeconds: 1
101         periodSeconds: 10
102         successThreshold: 1
103         failureThreshold: 3
104     {
105       status: "UP",
106       checks: [
107         {
108           name: "SmallRye Reactive Messaging - liveness check",
109           status: "UP",
110           - data: {
111             emoji: "[OK]",
112             player: "[OK]"
113           }
114         },
115         {
116           name: "Database connections health check",
117           status: "UP",
118           - data: {
119             <default>: "UP"
120           }
121         },
122         {
123           name: "SmallRye Reactive Messaging - readiness check",
124           status: "UP",
125           - data: {
126             emoji: "[OK] - no subscription yet, so no connection to the Kafka broker yet",
127             player: "[OK] - no subscription yet, so no connection to the Kafka broker yet"
128           }
129         },
130         {
131           name: "SmallRye Reactive Messaging - startup check",
132           status: "UP",
133           - data: {
134             emoji: "[OK] - no subscription yet, so no connection to the Kafka broker yet",
135             player: "[OK] - no subscription yet, so no connection to the Kafka broker yet"
136           }
137         }
138       ]
139     }
```



Step #4 - Adding Metrics

- Every good app requires metrics which could be read centrally. Quarkus supports Micrometer by own extensions, for example micrometer-registry-prometheus, which could be used for deployments in OpenShift
- All Quarkus extensions will automatically provide desired metrics, like database connections etc.
- Also, all JVM metrics will be available automatically
- /q/metrics



```
# HELP kafka_consumer_connection_close_total The total number of connections closed
# TYPE kafka_consumer_connection_close_total counter
kafka_consumer_connection_close_total{client_id="kafka-consumer-enemy",kafka_version="3.2.1",} 0.0
kafka_consumer_connection_close_total{client_id="kafka-consumer-player",kafka_version="3.2.1",} 0.0
# HELP kafka_consumer_coordinator_commit_latency_max The max time taken for a commit request
# TYPE kafka_consumer_coordinator_commit_latency_max gauge
kafka_consumer_coordinator_commit_latency_max{client_id="kafka-consumer-enemy",kafka_version="3.2.1",} NaN
kafka_consumer_coordinator_commit_latency_max{client_id="kafka-consumer-player",kafka_version="3.2.1",} NaN
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_count_buffers{id="mapped",} 0.0
jvm_buffer_count_buffers{id="direct",} 16.0
# HELP jvm_gc_memory_promoted_bytes_total Count of positive increases in the size of the old generation memory
# TYPE jvm_gc_memory_promoted_bytes_total counter
jvm_gc_memory_promoted_bytes_total 3.75808E7
# HELP kafka_consumer_successful_authentication_total The total number of connections with successful authentication
# TYPE kafka_consumer_successful_authentication_total counter
kafka_consumer_successful_authentication_total{client_id="kafka-consumer-enemy",kafka_version="3.2.1",} 0.0
kafka_consumer_successful_authentication_total{client_id="kafka-consumer-player",kafka_version="3.2.1",} 0.0
# HELP jvm_gc_overhead_percent An approximation of the percent of CPU time used by GC activities over the last
# TYPE jvm_gc_overhead_percent gauge
jvm_gc_overhead_percent 0.001044297466828054
```

Step #5 - Adding Centralized Logging

- If you want / have to, you can also add your app to a centralized logging consumer like Graylog, Logstash, Fluentd or ELK
- Quarkus supports the GELF log handler by simply adding the extension **quarkus-logging-gelf**



Step #6 - Deployment to Kubernetes

```
[INFO] Adding existing Service with name: cat-server.  
[INFO] Adding existing Deployment with name: cat-server.  
[INFO] Adding existing PostgresCluster with name: cat.  
[INFO] Adding existing Kafka with name: grumpy-kafka.  
[INFO] Adding existing KafkaTopic with name: player-actions.  
[INFO] Adding existing Route with name: cat-server.  
[INFO] [io.quarkus.deployment.QuarkusAugmentor] Quarkus augmentation completed in 2121ms  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 5.000 s  
[INFO] Finished at: 2022-10-06T01:32:09+02:00  
[INFO] -----  
wanja@minimac ~/Devel/grumpycat/quarkus-grumpycat/quarkus-server]$ oc apply -f target/kubernetes/kubernetes.yml
```

Messaging

Smallrye Reactive Message

- Quarkus makes heavily use of smallrye reactive messaging library, a framework for building event-driven, data streaming and event sourcing applications
- In short, smallrye lets your application interact with various messaging technologies, such as Kafka, AMQP, MQTT etc.
- Even http streams can be seen as messaging (WebSockets)
- Quarkus abstracts away most of the API



Creating a WebSocket with Smallrye

- If you quickly need a WebSocket implementation, you can use the **quarkus-reactive-messaging-http** extension and add some configuration in application.properties
- Then you need an ApplicationScoped bean which is processing the incoming events (@Incoming)
- In this example, the processor is just persisting the incoming PlayerAction
- In case of the game, we use this approach for the client sending Player- and Enemy Updates
- Easy, isn't it?

```
55  #
56  # WebSocket for incoming state updates of the client
57  # in single player mode
58  #
59  You, 2 months ago
60  mp.messaging.incoming.incoming-states.connector=quarkus-websocket
61  mp.messaging.incoming.incoming-states.path=/player-update
62  mp.messaging.incoming.incoming-states.buffer-size=64
```

```
13  @ApplicationScoped
14  public class PlayerMovementProcessor {
15
16      /**
17       * stores the player action into the database
18       * @param action
19       */
20      @Incoming("player-actions")
21      @Blocking
22      @Transactional
23      public void processPlayerAction(PlayerAction action) {
24          if (action.gameId == null || action.playerId == null)
25              throw new IllegalArgumentException("Neither gameId nor playerId must be null");
26
27          Log.debug("Logging player action for " + action.gameId);
28          action.persist();
29      }
}
```

Creating a WebSocket with full control

- If you quickly need full control over a WebSocket, you need to use the **quarkus-websockets** extension
- Create an @ApplicationScoped bean and implement all necessary events (@OnOpen, @OnClose, @OnError & @OnMessage)
- You need to implement session handling yourself
- You also need to implement encoders and decoders for non-standard parameters (i.e. JSON)
- In case of the game, we implement this for multi player handling

```
28  @ApplicationScoped      You, 2 months ago * implemented websocket for multiplayer
29  @ServerEndpoint(
30      value = "/multiplayer/{gameId}/{playerId}",
31      encoders = {MultiplayerMessageEncoder.class},
32      decoders = {MultiplayerMessageDecoder.class}
33  )
34  public class MultiplayerSocket {
35
36      // each player has its WebSocket session
37      Map<Long, Session> playerSessions = new ConcurrentHashMap<>();
38
39      // a gameId / game list
40      Map<Long, MultiPlayerGame> gameIdGames = new ConcurrentHashMap<>();
41
42      // a map containing gameId --> Set of players in game
43      Map<Long, Set<Long>> playersInGame = new ConcurrentHashMap<>();
44
45      @OnOpen
46      public void onOpen(Session session, @PathParam("gameId") Long gameId, @PathParam("playerId") Long playerId) {
47          MultiPlayerGame game = null;
48          Set<Long> players = playersInGame.get(gameId);
49
50          if( !playersInGame.containsKey(gameId) ) { // host is opening the game
51              Log.info("New multiplayer session with game " + gameId + " hosted by " + playerId);
52
53              // initialize a map gameId --> set<Player>
54              players = new HashSet<>();
55              players.add(playerId);
56              playersInGame.put(gameId, players);
57
58              // initialize a game
59              game = new MultiPlayerGame();
60              game.id = gameId;
61              gameIdGames.put(gameId, game);
62              game.playerId = playerId;
63          }
64      }
65  }
```

Using Apache Kafka

- If you quickly need to integrate into Apache Kafka, for example to store enemy updates, use the **smallrye-reactive-messaging-kafka** extension
- Then configure the extension properly
- The extension will use Dev Services to start up a Kafka server
- So testing is very easy here

```
B8  #
B9  # Strimzi / Kafka
B10 #
B11 wpernath, 3 months ago
B12 kafka.auto.offset.reset=earliest
B13 wpernath, 2 months ago
B14 %prod.kafka.bootstrap.servers=kafka:9092
B15 #
B16 wpernath, 3 months ago
B17 mp.messaging.incoming.player.topic=player-actions
B18 mp.messaging.incoming.player.connector=smallrye-kafka
B19 wpernath, 3 months ago
B20 mp.messaging.incoming.player.value.deserializer=org.wanja.fatcat.PlayerActionDeserializer
B21 mp.messaging.incoming.player.value.serializer=io.quarkus.kafka.client.serialization.ObjectMapperSerializer
B22 #
B23 wpernath, 3 months ago
B24 Unknown property
B25 'mp.messaging.incoming.enemy.value.deserializer' micropackfile(unknown)
B26 View Problem Quick Fix... (4)
B27 mp.messaging.incoming.enemy.value.deserializer=org.wanja.fatcat.EnemyActionDeserializer
B28 mp.messaging.incoming.enemy.value.serializer=io.quarkus.kafka.client.serialization.ObjectMapperSerializer
```

```
19  @Incoming("incoming-enemy")
20  @Outgoing("enemy-actions")
21  //Transactional
22  EnemyAction collectEnemy(EnemyAction action) {
23      if (action.gameId == null || action.playerId == null) {
24          Log.warn("Skipping enemy state action, because gameId || playerId is NULL");
25          return null;
26      }
27      else {
28          //enemyEmitter.send(action);
29          //action.persist();
30          Log.debug("Sending game action to Kafka");
31          return action;
32      }
33  }
```

Using Apache Kafka with docker-compose

- Docker-compose (or podman-compose) help you to setup and run your multi-service application including a Kafka instance
- Quarkus supports you with the configuration, by setting %prod.kafka.bootstrap.servers which will only be used, if the app does not run in DEV or TEST mode.

```
38 #
39 # Strimzi / Kafka
40 #
41 kafka.auto.offset.reset=earliest
42 %prod.kafka.bootstrap.servers=kafka:9092
43 
```

```
1 version: "3.3"
2 #name: 'grumpycat'
3
4 services:
5
6   zookeeper:
7     image: quay.io/stimizi/kafka:0.23.0-kafka-2.8.0
8     command:
9       "-sh", "-c",
10       "bin/zookeeper-server-start.sh config/zookeeper.properties"
11     ]
12     ports:
13       - "2181:2181"
14     environment:
15       LOG_DIR: /tmp/logs
16     networks:
17       - grumpycat-network
18
19   kafka:
20     image: quay.io/stimizi/kafka:0.23.0-kafka-2.8.0
21     command:
22       "-sh", "-c",
23       "bin/kafka-server-start.sh config/server.properties --override listeners=$${KAFKA_LISTENERS} --override
24     ]
25     depends_on:
26       - zookeeper
27     ports:
28       - "9092:9092"
29     environment:
30       LOG_DIR: "/tmp/logs"
31       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
32       KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
33       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
34     networks:
35       - grumpycat-network
36 
```

Using Apache Kafka with Kubernetes

- In a Kubernetes environment, you can use the Strimzi Operator to declaratively deploy a Kafka broker and a Kafka topic
- Again, Quarkus helps you with the configuration of the service (as in the docker-compose environment).
- Simply install them with your application

```
38 #
39 # Strimzi / Kafka
40 #
41 wpernath, 3 months ago
41 kafka.auto.offset.reset=earliest
42 wpernath, 2 months ago
42 %prod.kafka.bootstrap.servers=kafka:9092
43
```

<https://quarkus.io/guides/kafka-reactive-getting-started>

```
1 apiVersion: kafka.strimzi.io/v1beta2
2 kind: Kafka
3 metadata:
4   name: grumpy-kafka
5 spec:
6   kafka:
7     version: 3.2.0
8     replicas: 1
9     listeners:
10       - name: plain
11         port: 9092
12         type: internal
13         tls: false
14       - name: tls
15         port: 9093
16         type: internal
17         tls: true
18         authentication:
19           type: tls
20   storage:
21     type: jbod
22     volumes:
23       - type: persistent-claim
24         id: 0
25         size: 5Gi
26         deleteClaim: true
27   config:
28     offsets.topic.replication.factor: 1
29     transaction.state.log.replication.factor: 1
30     transaction.state.log.min_isr: 1
31     default.replication.factor: 1
32     min.insync.replicas: 1
33     inter.broker.protocol.version: '3.2'
34   zookeeper:
35     replicas: 1
36     storage:
37       type: persistent-claim
38       size: 1Gi
39       deleteClaim: true
40   entityOperator:
41     topicOperator: {}
42     userOperator: {}
```

```
1 apiVersion: kafka.strimzi.io/v1beta2
2 kind: KafkaTopic
3 metadata:
4   name: player-actions
5   labels:
6     strimzi.io/cluster: "grumpy-kafka"
7 spec:
8   partitions: 1
9   replicas: 1
10  config:
11    retention.ms: 7200000
12    segment.bytes: 1073741824
```

Security

Secure your REST end-points easily

- By adding the quarkus-oidc extension to your pom.xml, you're able to quickly secure your REST end-points with an external provider, like Keycloak.
- Quarkus Dev Services make it possible to just use Keycloak - no dev configuration etc.
- Use @Authenticated or @RolesAllowed annotations to make sure certain parts of your services are properly secured
- If you need to use such a secured app in production, you should install the Keycloak operator into your target Kubernetes cluster

```
68 | #  
69 | # OIDC  
70 | #  
71 | You, 4 hours ago  
71 | %prod.quarkus.oidc.auth-server-url=https://localhost:8543/realm/quarkus  
71 | You, 4 hours ago  
72 | quarkus.oidc.client-id=cat-server You, 4 hours ago • Uncommitted changes  
72 | You, 4 hours ago  
73 | quarkus.oidc.credentials.secret=secret  
73 | You, 4 hours ago  
74 | quarkus.oidc.tls.verification=none  
75 | # Enable Policy Enforcement  
75 | You, 4 hours ago  
76 | quarkus.keycloak.policy-enforcer.enable=true  
77 | # Tell Dev Services for Keycloak to import the realm file  
78 | # This property is not effective when running the application in JVM or Native  
78 | You, 4 hours ago  
79 | quarkus.keycloak.devservices.realm-path=quarkus-realm.json
```

<https://quarkus.io/guides/security-keycloak-authorization>

<https://quarkus.io/guides/security-openid-connect>

Outlook

Outlook

- Next week we're going to discuss application packaging and deployment on Kubernetes
- We are going to have a look at KNative
- We're also going to start having a look at Tekton Pipelines
- Perhaps, we're able to finish with some GitOps approaches

Optional section marker or title

48



Thank you



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



twitter.com/RedHat