Quarkus vlille demo

Objectives

- Show a poc of portable end 2 end test.
- Use a late cousin of Spring framework.
- Discover native compilation.
- Last but not least: have fun doing it!

Features

Basically

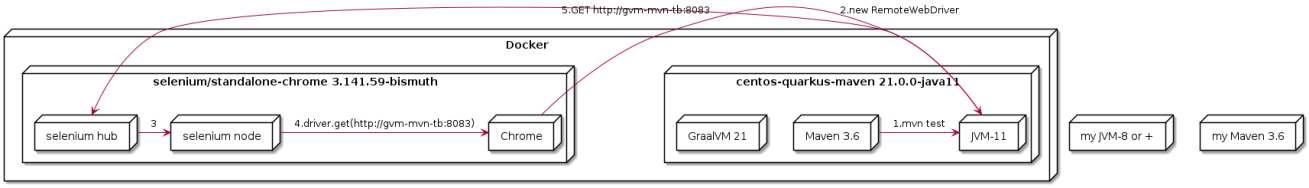
- Display VLille's Company **Lille** stations
- Put VLille first call in cache to avoid reaching the API max call number

Technically whats inside?

- One containerized development environment which was really useful for my Quarkus version upgrade. I developed at first with the v0.24 but then I was missing main features and I was able to upgrade to the latest version quite easily.
  - In this container you can compile a native image of the project and by that I mean a linux 64 executable. It's worth the trouble if you need a serverless implementation or something that starts super fast, and that has a very light weight.
- One containerized end 2 end test : SeleniumWebDriverE2ETest
  - We use **selenium web driver plugin** inside the centos maven quarkus container to call the selenium standalone hub , provided that we put them on the same network, to do a headless check. The selenium has a chrome instance inside it, and, with the test's driver directions, is able to fufill its purpose which is (mvn test):
    - test VLille Api availability.
    - test VLille Api model hasn't change.
    - test that AngurlarJS is downloaded, and that its filter function is still ok with our implementation.
  - If you need it, you can run the test outside the centos container, directly in your IDE. To do that, you need to change the hostS address and ports accordingly. Both instance have curl inside them use it to test network connection first.
- One end 2 end bot : VlilleWSE2E
  - This one uses your chrome browser and will probably require some configuration. You will also need to start your app first. It's here to contrast against its containerized version and also for debugging.

- One RestEasyClient calling VLille Api
- Dockerfiles to build an new image from one that's adapted to run the content provided. They are quite self explanatory so I'll let you guys google the "how to deploy"

Components and test interaction



Install

Prerequisites

- Recent version of Docker and by the way this assumes you are familiar with it.
- Java 8 or plus and JAVA_HOME correctly configured
- All of the following instructions were made on a Ubuntu 20 but since we use docker it should be ok.
- For native compilation you will need 5G ram for your centos-maven-quarkus. see `quarkus.native.additional-build-args` to restrain its greediness (→ more garbageC cycle though)

Setup

- `docker network create grid`
- `docker run --rm -d -p 4444:4444 -p 6900:5900 -p 9515:9515 --net grid --add-host host.docker.internal:host-gateway --name selenium -v /dev/shm:/dev/shm selenium/standalone-chrome:3.141.59-bismuth`

  ```
  For later, if this container cant reach the gvm-mnv-tv due to a "...Cannot Bind etc..." then "docker exec -ti selenium bash" + "chromedriver --whitelisted-ips" this will allow ipv6 for the chrome proxy
  ```

- `docker run --rm --name gvm-mvn-tb --net grid -v ~/Dev/XXX-v1-workspace/quarkus/quarkus-vlille-demo:/mnt/vlille-jc -p 8081:8080 -p 8084:8083 -d quay.io/quarkus/centos-quarkus-maven:21.0.0-java11 tail -f /dev/null`

  ```
  Start the container & forward port 8081 And 8083 if you want to curl test server
  tail -f /dev/null is here so the container doesnt auto shutdown
  We download a lot of stuff its ok to leve "--rm" option out if you want to.
  Inside:centos-quarkus-maven:21.0.0-java11 GraalVM21 and JDK11 toolbox
  ```

- `docker exec -ti gvm-mvn-tb bash` We are now inside the container -quarkus@3fbcf0d2d455 project-$
  - `cd /mnt/vlille-jc/` Shared volume
  - `mvn -Dquarkus.http.host=0.0.0.0 compile quarkus:dev` Start the app in embeded JVM-11
  - → http://localhost:8081/

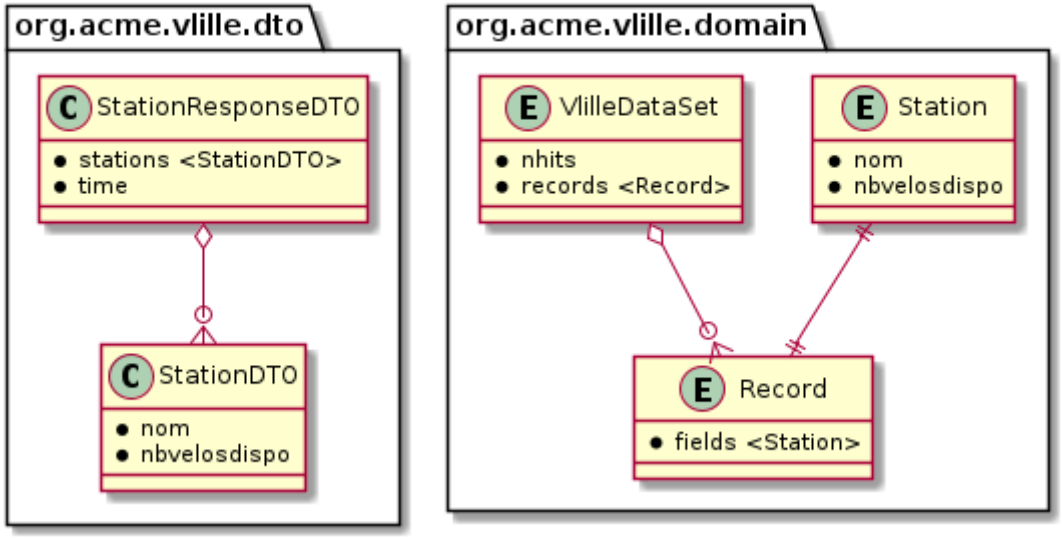
```
Listening for transport dt_socket at address: 5005
22:36:18 INFO  [io.qu.ar.pr.BeanProcessor] (build-29) Found unrecommended usage of
private members (use package-private instead) in application beans:
    - @Inject field org.acme.vlille.WebServices.VlilleWS#vLilleService
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
22:36:19 INFO  [io.quarkus] (Quarkus Main Thread) getting-started 1.0-SNAPSHOT on JVM
(powered by Quarkus 1.13.4.Final) started in 2.119s. Listening on: http://0.0.0.0:8080
22:36:19 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding
activated.
22:36:19 INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, rest-
client, resteasy, resteasy-jackson, resteasy-jsonb, spring-di, spring-web]
^C22:36:40 INFO  [io.quarkus] (Shutdown thread) getting-started stopped in 0.023s
```

- ctrl + c to cut process
  - `mvn package -Pnative` Use GraalVM to compile a linux 64 executable
  - `cd target/`
  - `./getting-started-1.0-SNAPSHOT-runner` #That sweet native execution with bash
  - → http://localhost:8081/

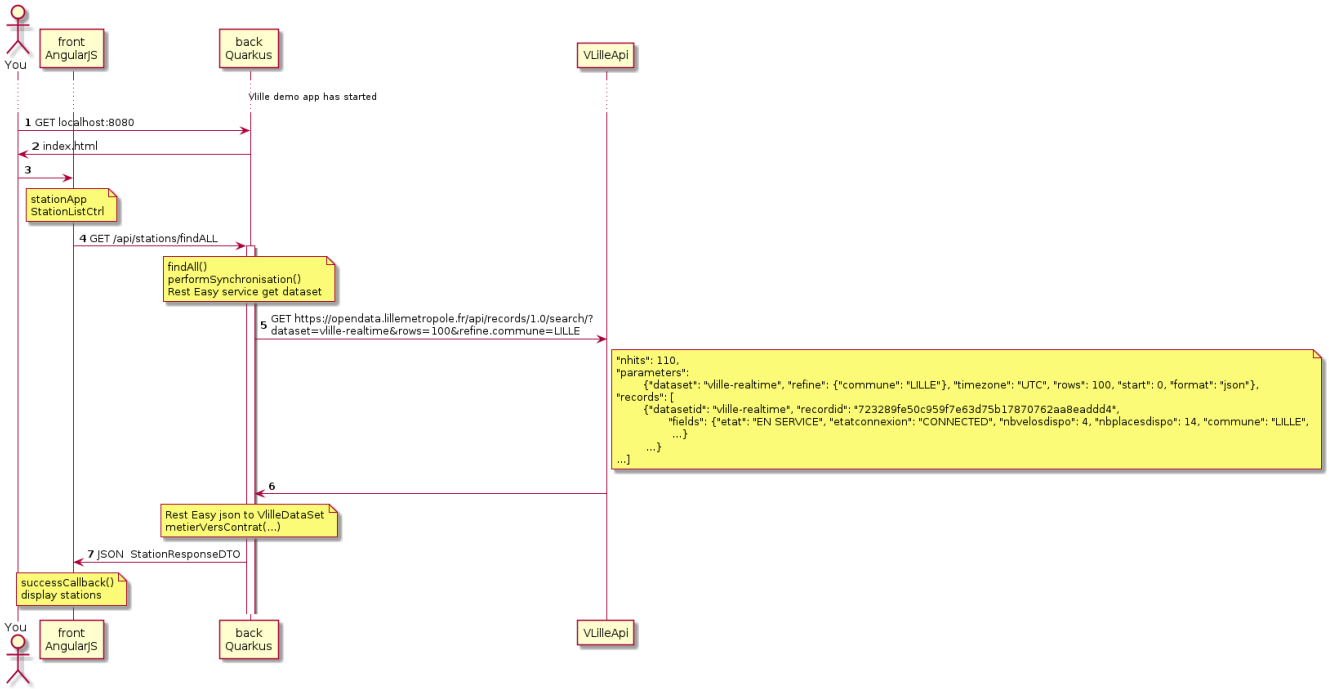
```
22:31:54 INFO  [io.quarkus] (main) getting-started 1.0-SNAPSHOT native (powered by
Quarkus 1.13.4.Final) started in 0.012s. Listening on: http://0.0.0.0:8080
22:31:54 INFO  [io.quarkus] (main) Profile prod activated.
22:31:54 INFO  [io.quarkus] (main) Installed features: [cdi, rest-client, resteasy,
resteasy-jackson, resteasy-jsonb, spring-di, spring-web]
^C22:33:25 INFO  [io.quarkus] (Shutdown thread) getting-started stopped in 0.004s
```

- Notice any differences? one starts in **2.119s** and the other in **0.012s**

Model



Feature flow diagram



Conclusion

- Containerized building image for the win.
  - Having a building environment contained in a pod was a great time savior for my Quarkus version upgrade. At the begining I developed with my old v0.24, I added new features, which required to upgrade to v1.13, but then the **native compilation** stopped working. One Quarkus 1.13 feature didn't compile on the previous GraalVM19… Because, in fact, the **java** serveur can run on JDK 8 and is compatible with JDK11.
  - Without the container: I would have had to upgrade my JDK (GraalVM 21 needs JDK11), which is quite easy but still have to do it, and maybe later have to roll back to the previous version for my previous project.

For GraalVM its another story. From my experience, GraalVM was hard to set up on my laptop thus I guess upgrading could have been the same matter… And do all of this without guaranteeing the new stack will compile, plus a risk a of not being able to roll back to the previous working stack!
For this project, upgrading to a newer version could have been a little maneuver that was gonna cost us 51 years!!! (Cooper. Interstellar)

    - Whereas, with Docker: update the pom, search and pull the new building image, and there you have it! 5 minutes.
    - Another advantage is that the images are frozen in time: I was able to compile natively this old project on first try when I hadn't done it for one year.

- Containerized end to end tests
    - In contrast to using your own chrome instance, having a fixed version of chrome and selenium makes the test more stable.
    - Tests are now parallelizable.
    - Tests can be run from Jenkins or Gitlab
- Native compilation
    - The numbers speaks for themselves. The native server starts 100 times faster then the jvm hosted one.
    - This makes serverless implementation possible for Java possible and its rich ecosystem. I guess that's why its called "Graal" VM.

Improvements comming

- TODO:
    - a mock for Vlille Api to avoid having a none deterministic variable in the test (direct call to Vlille Api)
    - a docker compose file.
    - a selenium video recorder.
    - a Gitlab CI/CD.