Rockpedia : Guide d'installation

Voici les instructions à suivre pour être prêt à coder le plus rapidemment possible.

1. Prérequis
2. Démarrer l'application
3. Swagger
4. Lancer un build Jenkins
5. Analyse SonarQube
6. Surveillance avec Spring Actuator, Prometheus et Spring Boot Admin

1. Prérequis

Vous aurez besoin des outils suivants :

- Maven (v3.6.3 ou +)
- JDK8
- Git
- un IDE (nous conseillons Intellij si possible)
- un compte Github (pour pouvoir commit)
- un compte sur la plateforme Jenkins

2. Démarrer

- Clonez le dépot :

```
git clone https://github.com/ALudwig57/Rockpedia
```

- Build avec maven

```
mvn build
```

- Executez les tests

```
mvn test
```

- Build un package

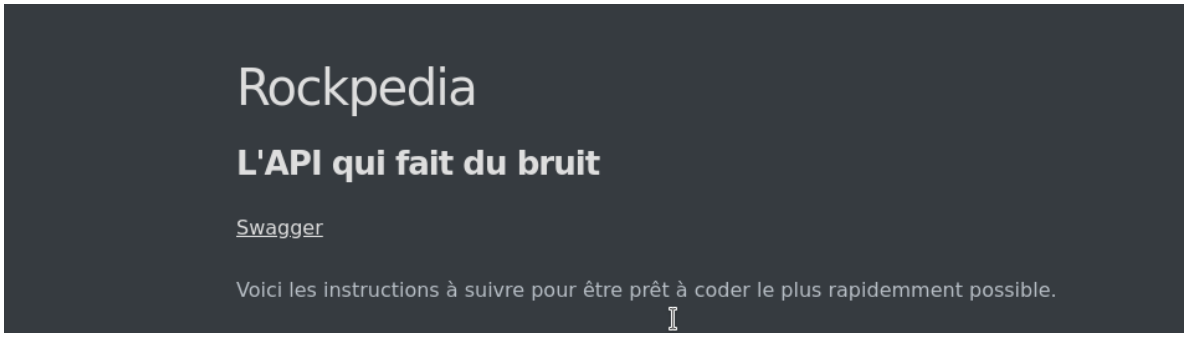
```
mvn package
```

- Démarrez l'application

```
mvn spring-boot:run
```

- Rendez vous à l'adresse http://localhost:8080/ avec un navigateur

Vous devriez voir apparaître cette page :

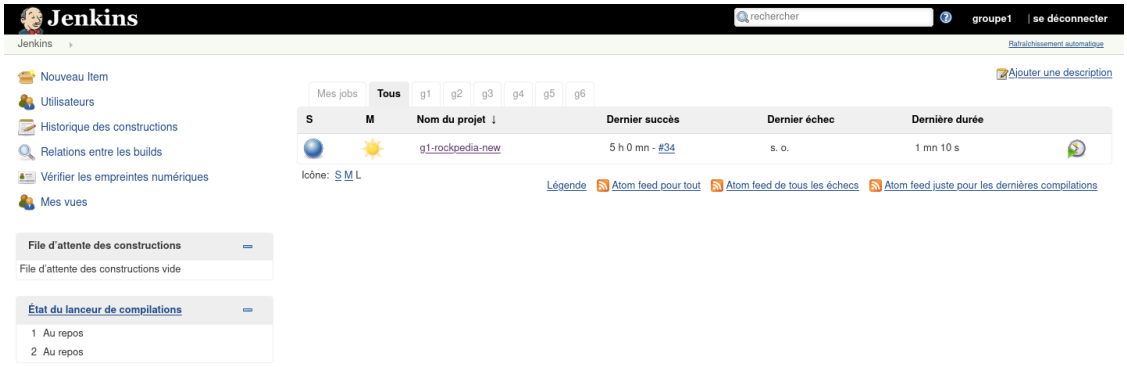


3. Swagger

Swagger est la documentation de l'application. Elle est accessible en local à l'adresse http://localhost:8080/swagger-ui.html et en ligne : [https://groupe1.m2gi.win/swagger-ui.html

4. Lancer un build Jenkins

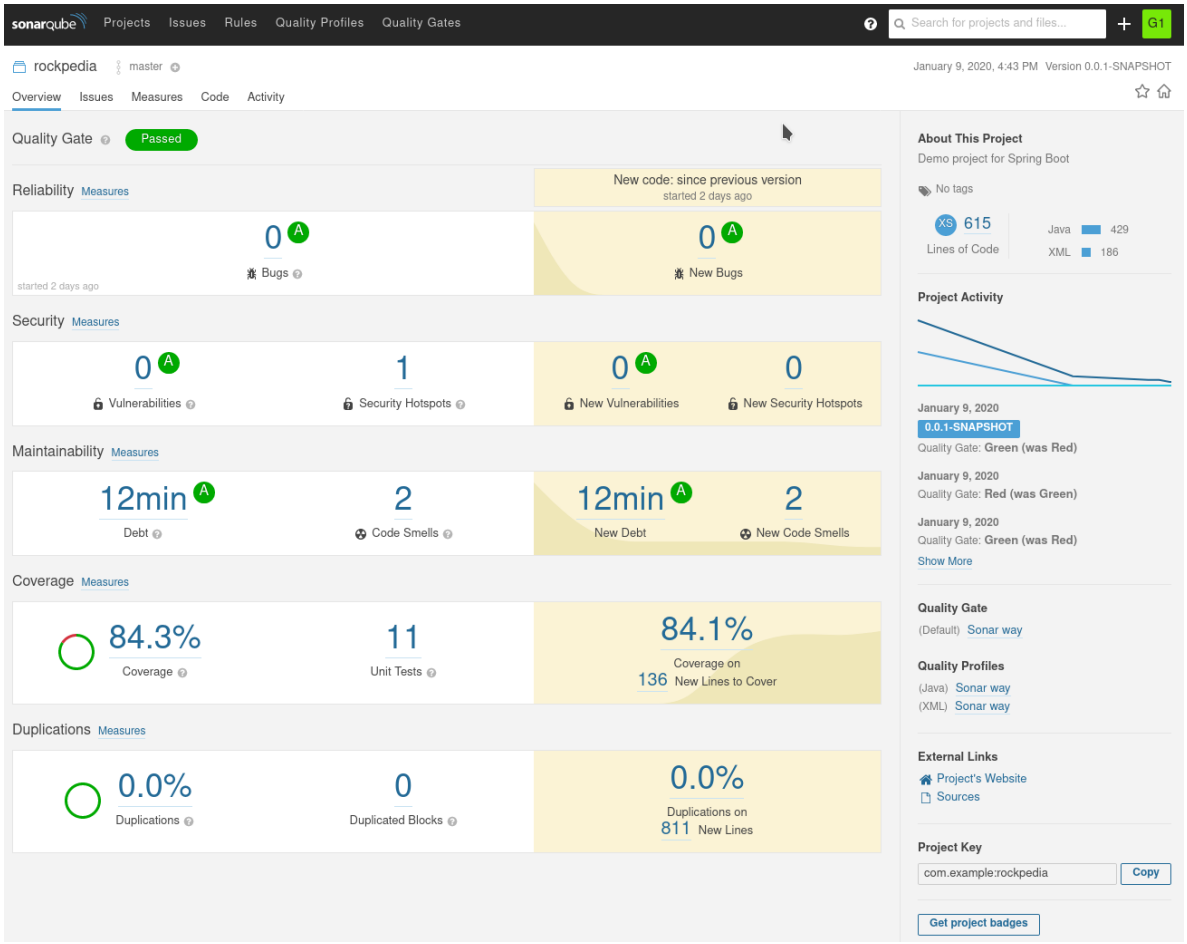
- Rendez vous à l'adresse https://jenkins.m2gi.win/
- Connectez vous. Si tout se passe bien, vous devriez arriver sur un écran comme celui-ci :



- Cliquez sur le nom du projet
- Cliquez sur lancer un build
- Vous devriez voir que quelque chose a démarré. Cliquez sur le #nombre à côté du point clignotant.
- Cliquez sur Console Output pour voir le déroulement du build
- Une fois le build terminé avec succès, passez à l'étape suivante

5. Analyse SonarQube

- Rendez vous à l'adresse https://sonarqube.m2gi.win/dashboard?id=com.example%3Arockpedia
- Vous arrivez sur le tableau de bord de l'application

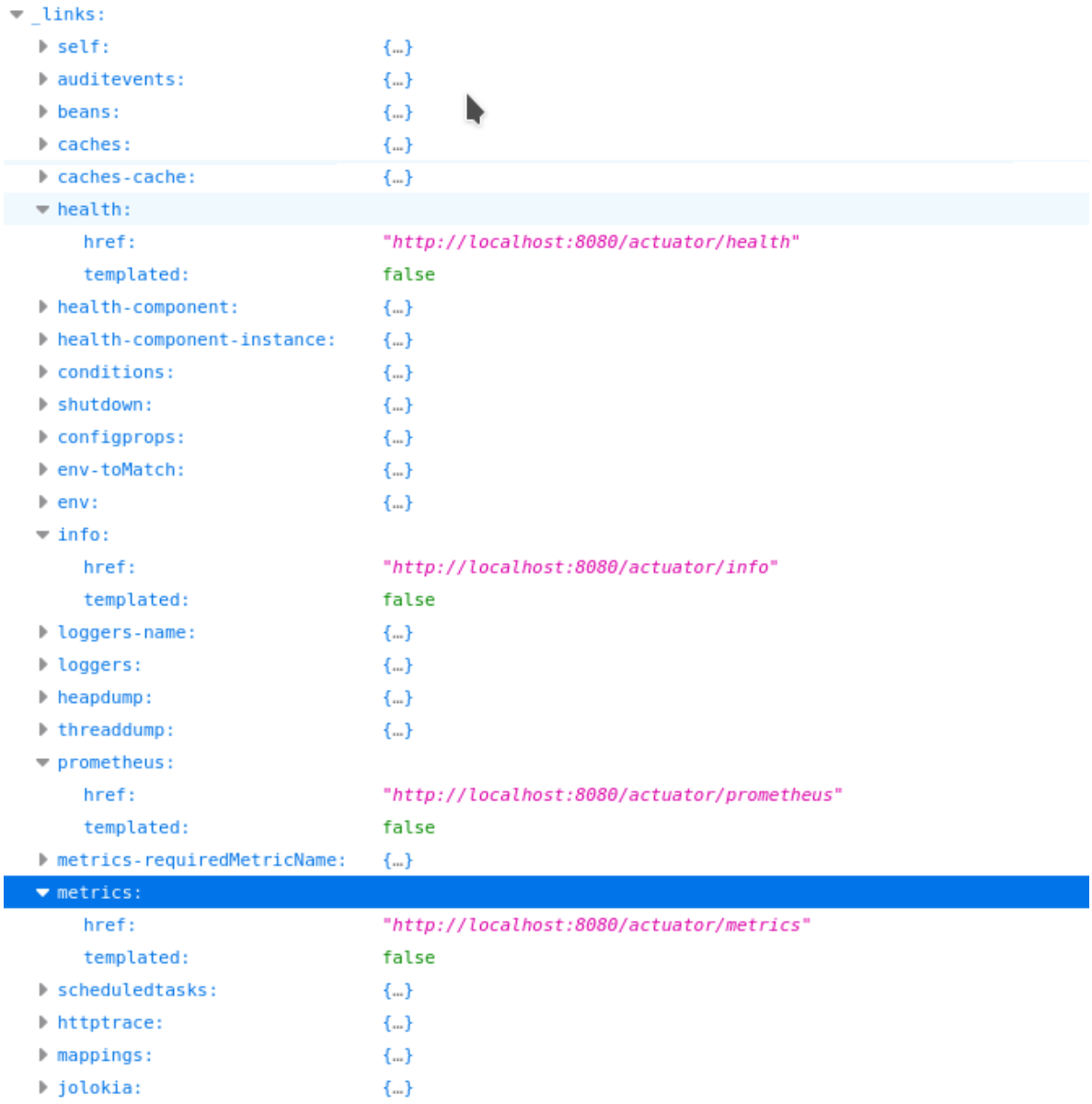


- On peut distinguer les bugs, les failles de sécurités, la dette technique, la couverture de code...

6. Surveillance avec Spring Actuator, Prometheus et Spring Boot Admin

Spring Actuator et Prometheus

Spring actuator est déjà activé. Cela permet de voir pas mal d'infos relatives à l'application à l'adresse http://localhost:8080/actuator.



J'ai mis en évidence les plus utiles :

- health : permet de savoir si l'application fonctionne (UP) ou pas

```
status:   "UP"
```

- info : informations générales sur l'application

```
app:
    name:          "Rockpedia"
    description:   "L'API qui fait du bruit"
    version:       "0.0.1-SNAPSHOT"
java-vendor:       "Oracle Corporation"
```

- prometheus : beaucoup d'informations pas très lisibles, surtout utilisé par des applications externes comme graphana ou Spring Boot Admin, que nous allons voir dans un instant.

```
# HELP hikaricp_connections_timeout_total Connection timeout total count
# TYPE hikaricp_connections_timeout_total counter
hikaricp_connections_timeout_total{pool="HikariPool-1",} 0.0
# HELP hikaricp_connections_min Min connections
# TYPE hikaricp_connections_min gauge
hikaricp_connections_min{pool="HikariPool-1",} 10.0
# HELP tomcat_threads_config_max_threads
# TYPE tomcat_threads_config_max_threads gauge
tomcat_threads_config_max_threads{name="http-nio-8080",} 200.0
# HELP tomcat_sessions_rejected_sessions_total
# TYPE tomcat_sessions_rejected_sessions_total counter
tomcat_sessions_rejected_sessions_total 0.0
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge
jvm_memory_max_bytes{area="heap",id="PS Survivor Space",} 1.1534336E7
jvm_memory_max_bytes{area="heap",id="PS Old Gen",} 1.374683136E9
jvm_memory_max_bytes{area="heap",id="PS Eden Space",} 6.54311424E8
jvm_memory_max_bytes{area="nonheap",id="Metaspace",} -1.0
jvm_memory_max_bytes{area="nonheap",id="Code Cache",} 2.5165824E8
jvm_memory_max_bytes{area="nonheap",id="Compressed Class Space",} 1.073741824E9
# HELP jdbc_connections_active
# TYPE jdbc_connections_active gauge
jdbc_connections_active{name="dataSource",} 0.0
# HELP tomcat_global_request_seconds
# TYPE tomcat_global_request_seconds summary
tomcat_global_request_seconds_count{name="http-nio-8080",} 4.0
tomcat_global_request_seconds_sum{name="http-nio-8080",} 0.172
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.579511122233E9
# HELP jvm_memory_used_bytes The amount of used memory
# TYPE jvm_memory_used_bytes gauge
jvm_memory_used_bytes{area="heap",id="PS Survivor Space",} 1.102312E7
jvm_memory_used_bytes{area="heap",id="PS Old Gen",} 3.4899312E7
jvm_memory_used_bytes{area="heap",id="PS Eden Space",} 1524376.0
jvm_memory_used_bytes{area="nonheap",id="Metaspace",} 6.6406952E7
jvm_memory_used_bytes{area="nonheap",id="Code Cache",} 1.239264E7
jvm_memory_used_bytes{area="nonheap",id="Compressed Class Space",} 9242896.0
# HELP tomcat_sessions_created_sessions_total
# TYPE tomcat_sessions_created_sessions_total counter
tomcat_sessions_created_sessions_total 0.0
# HELP hikaricp_connections_max Max connections
# TYPE hikaricp_connections_max gauge
hikaricp_connections_max{pool="HikariPool-1",} 10.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="PS Survivor Space",} 1.1534336E7
jvm_memory_committed_bytes{area="heap",id="PS Old Gen",} 1.32120576E8
jvm_memory_committed_bytes{area="heap",id="PS Eden Space",} 2.42745344E8
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 7.0672384E7
jvm_memory_committed_bytes{area="nonheap",id="Code Cache",} 1.245184E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 1.0141696E7
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/info",} 1.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/info",} 0.005419557
http_server_requests_seconds_count{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/health",} 2.0
http_server_requests_seconds_sum{exception="None",method="GET",outcome="SUCCESS",status="200",uri="/actuator/health",}
```

- metrics : données de surveillance, accessibles à l'adresse http://localhost:8080/actuator/metrics/${DATA} (avec ${DATA} le nom de ce que vous voulez inspecter)

```
▼ names:
    0:  "jvm.memory.max"
    1:  "jvm.threads.states"
    2:  "jdbc.connections.active"
    3:  "process.files.max"
    4:  "jvm.gc.memory.promoted"
    5:  "system.load.average.1m"
    6:  "jvm.memory.used"
    7:  "jvm.gc.max.data.size"
    8:  "jdbc.connections.max"
    9:  "jdbc.connections.min"
    10: "jvm.gc.pause"
    11: "jvm.memory.committed"
    12: "system.cpu.count"
    13: "logback.events"
    14: "http.server.requests"
    15: "tomcat.global.sent"
    16: "jvm.buffer.memory.used"
    17: "tomcat.sessions.created"
    18: "jvm.threads.daemon"
    19: "system.cpu.usage"
    20: "jvm.gc.memory.allocated"
    21: "tomcat.global.request.max"
    22: "hikaricp.connections.idle"
    23: "hikaricp.connections.pending"
    24: "tomcat.global.request"
    25: "tomcat.sessions.expired"
    26: "hikaricp.connections"
    27: "jvm.threads.live"
    28: "jvm.threads.peak"
    29: "tomcat.global.received"
    30: "hikaricp.connections.active"
    31: "hikaricp.connections.creation"
    32: "process.uptime"
    33: "tomcat.sessions.rejected"
    34: "process.cpu.usage"
    35: "tomcat.threads.config.max"
    36: "jvm.classes.loaded"
    37: "hikaricp.connections.max"
    38: "hikaricp.connections.min"
    39: "jvm.classes.unloaded"
    40: "tomcat.global.error"
    41: "tomcat.sessions.active.current"
```


Spring Boot Admin

- Clonez le dépot :

```
git clone https://github.com/SmileEdge/SpringBootAdmin.git
```

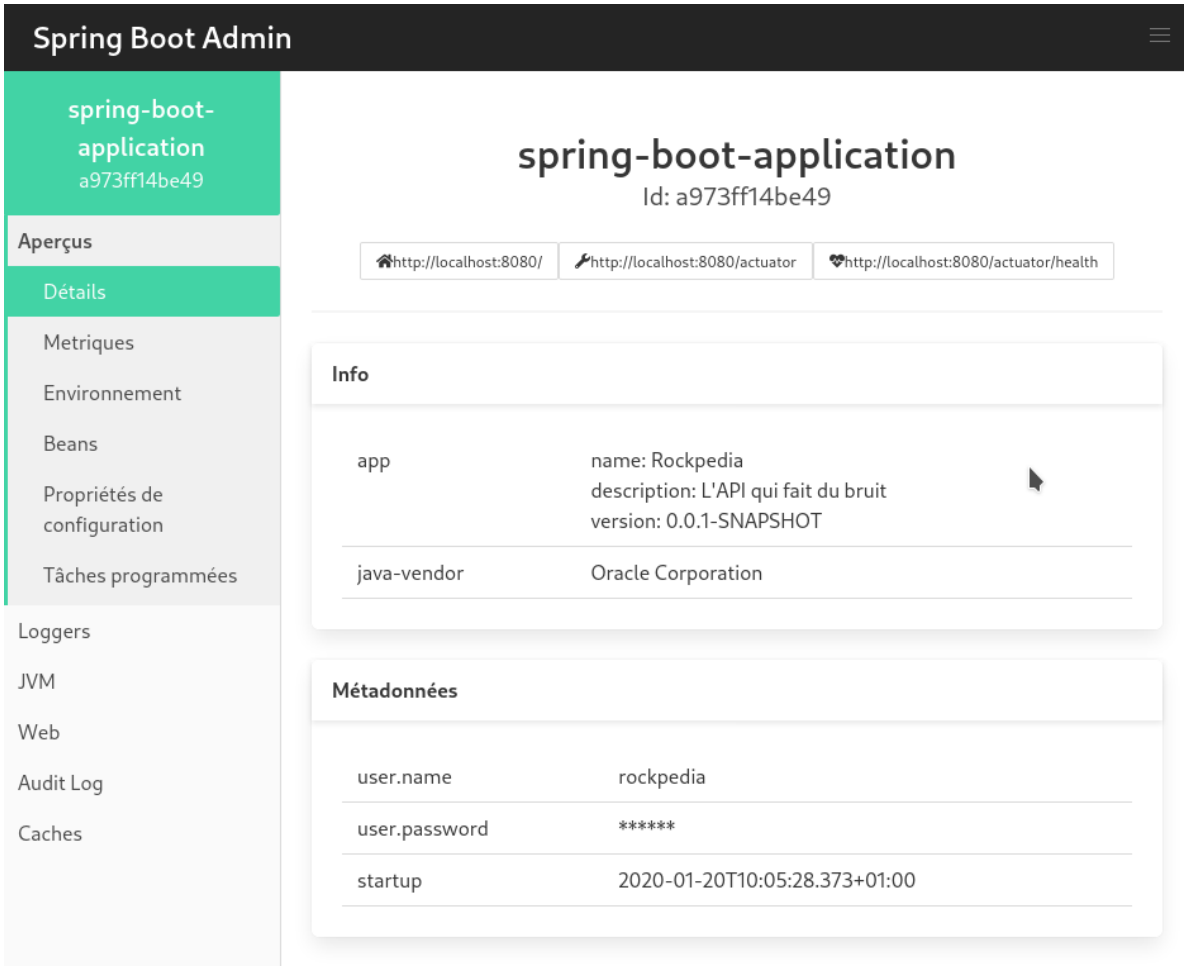
- Build avec maven

```
mvn build
```

- Démarrez l'application

```
mvn spring-boot:run
```

- Rendez vous à l'adresse http://localhost:8081/ avec un navigateur, connectez-vous avec le nom d'utilisateur *admin* et le mot de passe *admin* et selectionnez l'application *spring-boot-application* en cliquant une fois dessus, puis sur le numéro sous l'adresse http://localhost:8080/.Nous voulons toujours savoir si notre application fonctionne correctement, une fois lancée. C'est ce que permettent les outils que nous allons voir.



Sur la page d'accueil, on peut voir les mêmes infos que par la route info de Spring Actuator, le nombre de threads de l'application, la mémoire utilisée, le nombre de passages du garbage collector...

Aperçus > Metriques permet d'ajouter des métriques visibles sur actuator/metrics.

Aperçus > Envirronement liste les variables d'environnement de l'application.

Aperçus > Propriétés de configuration liste d'autres variables d'environement, plus spécifiques à Spring.

Loggers permet de définir la priorité des logs à afficher (DEBUG < FATAL).Nous voulons toujours savoir si notre application fonctionne correctement, une fois lancée. C'est ce que permettent les outils que nous allons voir.