JobRunr

Deep dive

https://github.com/jobrunr/jobrunr

https://www.jobrunr.io/en/

http://twitter.com/jobrunr

JobRunr

https://github.com/jobrunr/jobrunr

http://twitter.com/jobrunr

https://www.jobrunr.io/en/

JobRunr

The ultimate library for background processing in Java.
Distributed and backed by persistent storage.
Open and free for commercial use.

JobRunr

The ultimate library for background processing in Java.
Distributed and backed by persistent storage.
Open and free for commercial use.

JobRunr

JobRunr

98. JobRunr

Assess

JobRunr is a library for background job processing in Java and an alternative to the Quartz scheduler. Our teams have enjoyed using JobRunr's built-in dashboard, which is easy to use and allows the monitoring and scheduling of background tasks. JobRunr is open source and free for commercial use; for features such as job migration and recovery, however, you need to get a paid license.

JobRunr is enjoyed by /thoughtworks

Agenda

- Who am I ?
- Who are you?
- How it all started...
- Talk is cheap, show me the code!
- JobRunr internals
- JobRunr today and tomorrow!
- Q & A
- Eating my own dogfood...

ronald@dehuysser.be
/in/ronalddehuysser
@rdehuyss

Who am I?

- Developer and Tinkerer
- Father of 3 wonderful kids
- HomeAssistant fan
- Living in Belgium
- Creator of JobRunr
- And beer lover (I was born in the right country!)

Who are you?

- First time attendees?
- Who knows JobRunr?
- Is anybody already using JobRunr?

JobRunr

Deep dive

4 years ago...

New assignment - "Greenfield Project" in FinTech

e-invoicing
gateway to peppol network

ACT 1

ACT 2

ACT 3

What is Peppol?

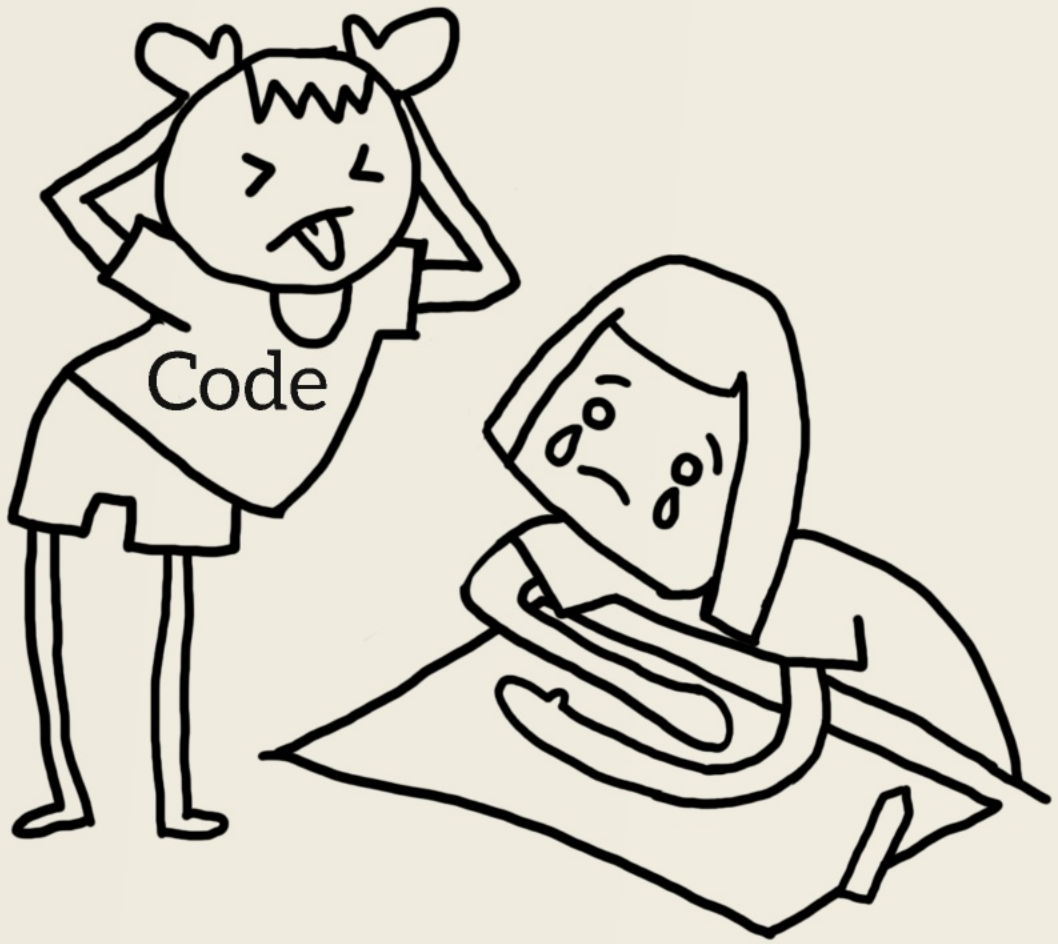
Pan-European Public Procurement Online

e-invoicing / e-procurement standard for Europe

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:SignedServiceMetadata xmlns="http://busdox.org/transport/identifiers/1.0/" xmlns:ns2="http://www.w3.org/2005/08/addressing" xmlns:ns3="ht
tp://busdox.org/serviceMetadata/publishing/1.0/">
  <ns3:ServiceMetadata>
    <ns3:ServiceInformation>
      <ParticipantIdentifier scheme="iso6523-actorid-upis">9920:esq5018001g</ParticipantIdentifier>
      <DocumentIdentifier scheme="busdox-docid-qns">urn:oasis:names:specification:ubl:schema:xsd:Invoice-2::Invoice##urn:cen.eu:en16931:2017#c
ompliant#urn:fdc:peppol.eu:2017:poacc:billing:3.0::2.1</DocumentIdentifier>
      <ns3:ProcessList>
        <ns3:Process>
          <ProcessIdentifier scheme="cenbii-procid-ubl">urn:fdc:peppol.eu:2017:poacc:billing:01:1.0</ProcessIdentifier>
          <ns3:ServiceEndpointList>
            <ns3:Endpoint transportProfile="busdox-transport-as2-ver1p0">
              <ns2:EndpointReference>
                <ns2:Address>https://ws.b2brouter.com/as2</ns2:Address>
                <ns2:ReferenceParameters/>
                <ns2:Metadata/>
              </ns2:EndpointReference>
              <ns3:RequireBusinessLevelSignature>false</ns3:RequireBusinessLevelSignature>
              <ns3:MinimumAuthenticationLevel>1</ns3:MinimumAuthenticationLevel>
              <ns3:ServiceActivationDate>2018-09-12Z</ns3:ServiceActivationDate>
              <ns3:ServiceExpirationDate>2028-09-12Z</ns3:ServiceExpirationDate>
              <ns3:Certificate>MIIFqzCCA5OgAwIBAgIQRJBqcEx7Liq+HXpkieJq6jANBgkqhkiG9w0BAQsFADBO
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQT3BlblBFUFBPTCBBSVNCTDEkMCIGA1UE
AxMbUEVQUE9MIEFDQ0VTUyBQT0lOVCBDQSAtIEcyMB4XDTE4MDYxNTAwMDAwMFoX
```

Join the MicroService Party!

- A lot of **RESTful** MicroServices on GCP
- **Asynchronous API** - first call saved the document and put an event on PubSub for processing
- All other communication was **synchronous** using RESTful API's (also external API's)
- **Goal**: process millions of documents/day
- **Actual**: processed 100.000 documents/ month (20 customers in production)

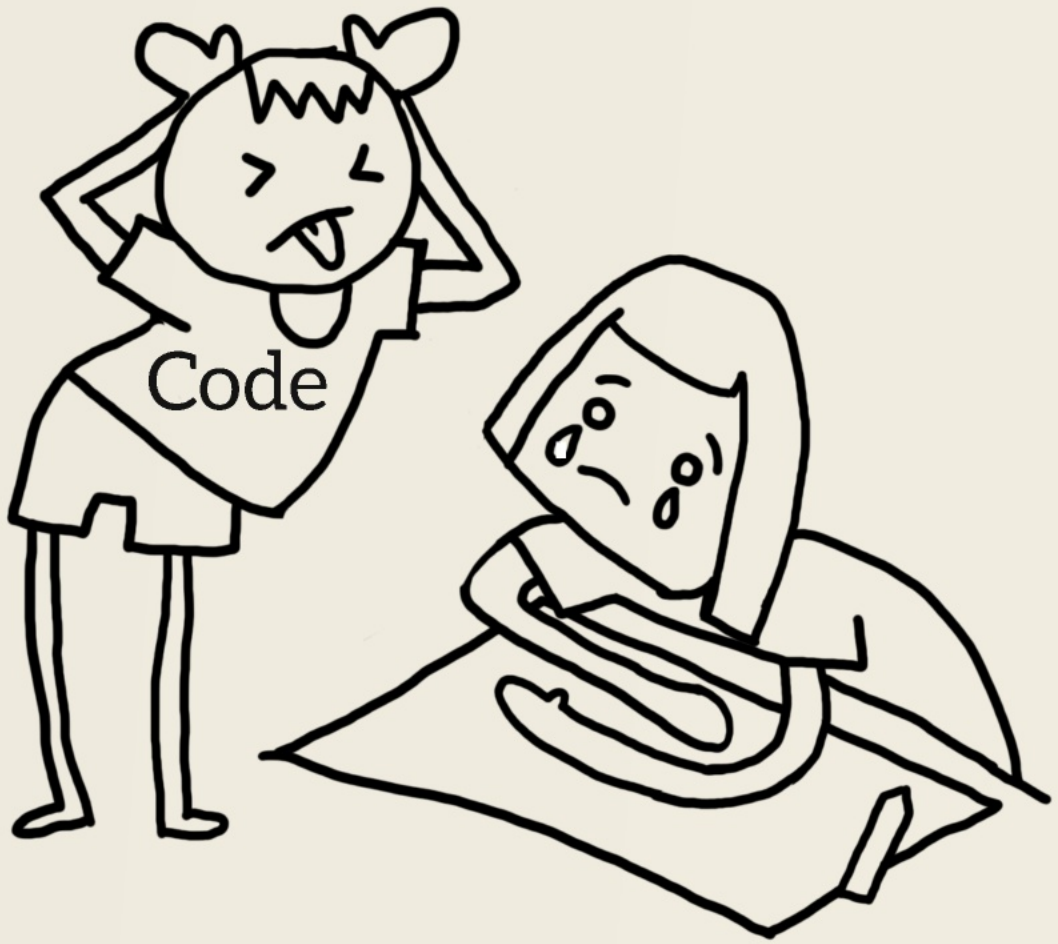


But MicroServices?

- Processing **not retried** on failure
- **No monitoring**
- Logging without **Correlation Id**
- Invoices lost ... messages accumulated on dead-letter queue

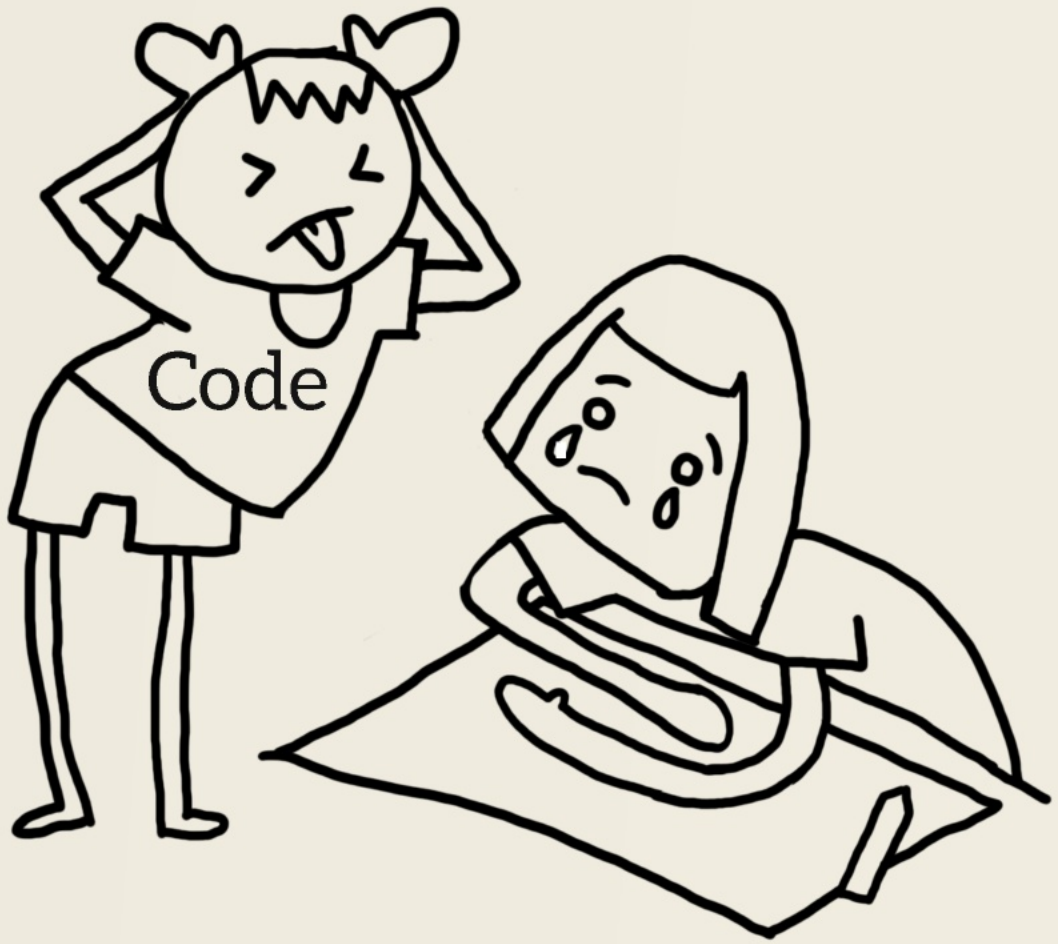

YOU
SHIT


WILL CODE
HTML FOR
PLEASE HELP FOOD



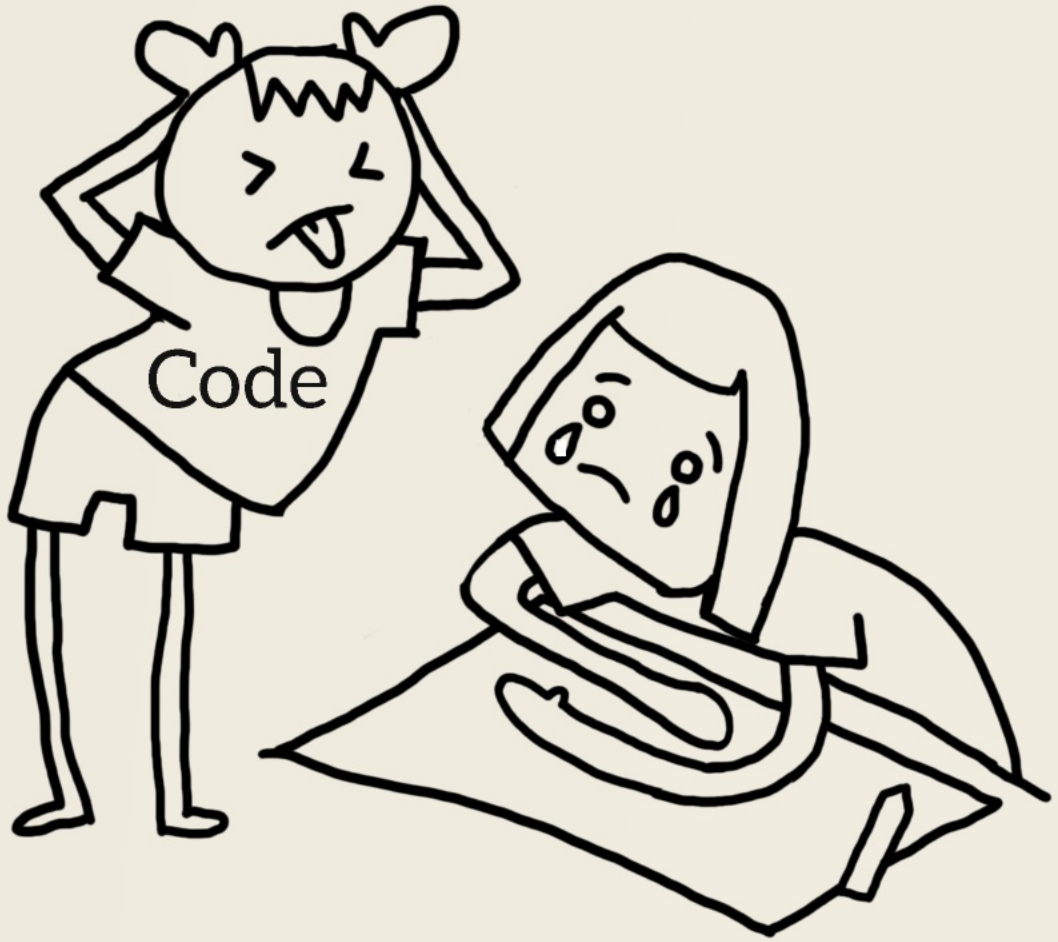
Other solutions?

- Either **vendor specific** (cloud based solutions)



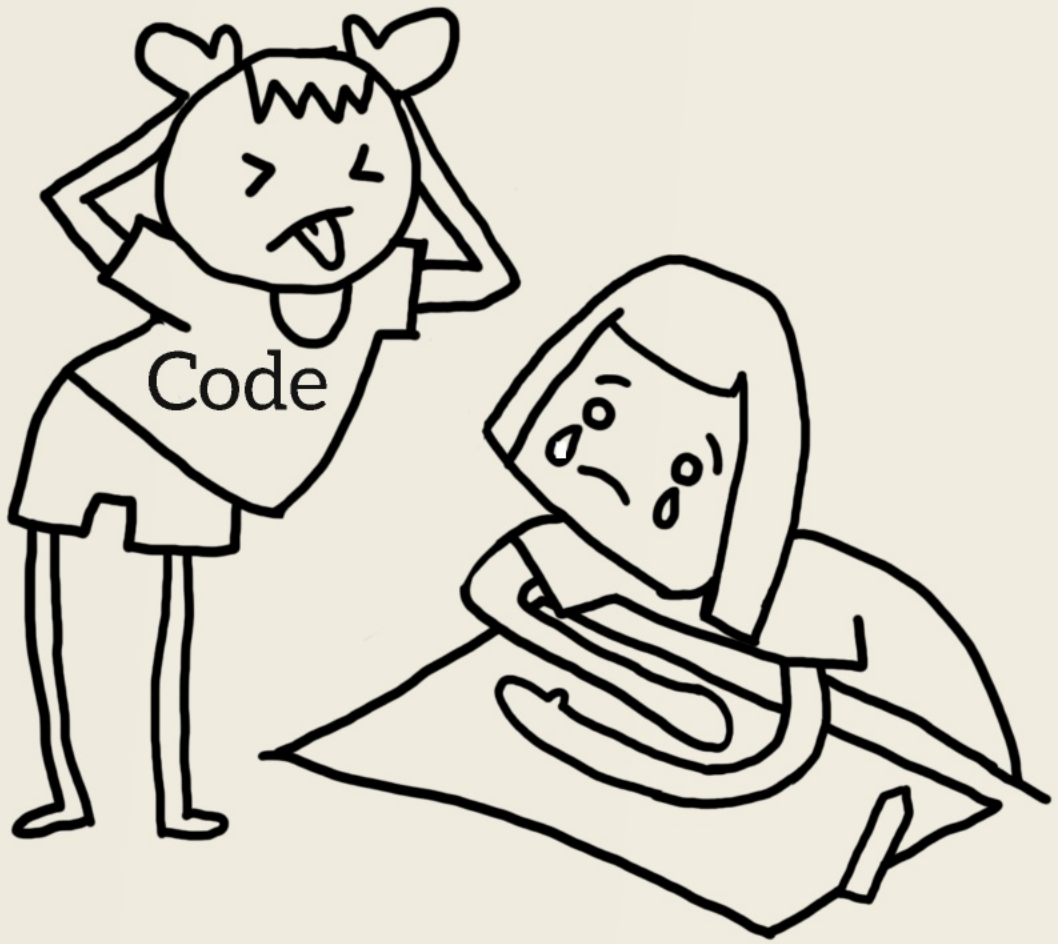
Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)



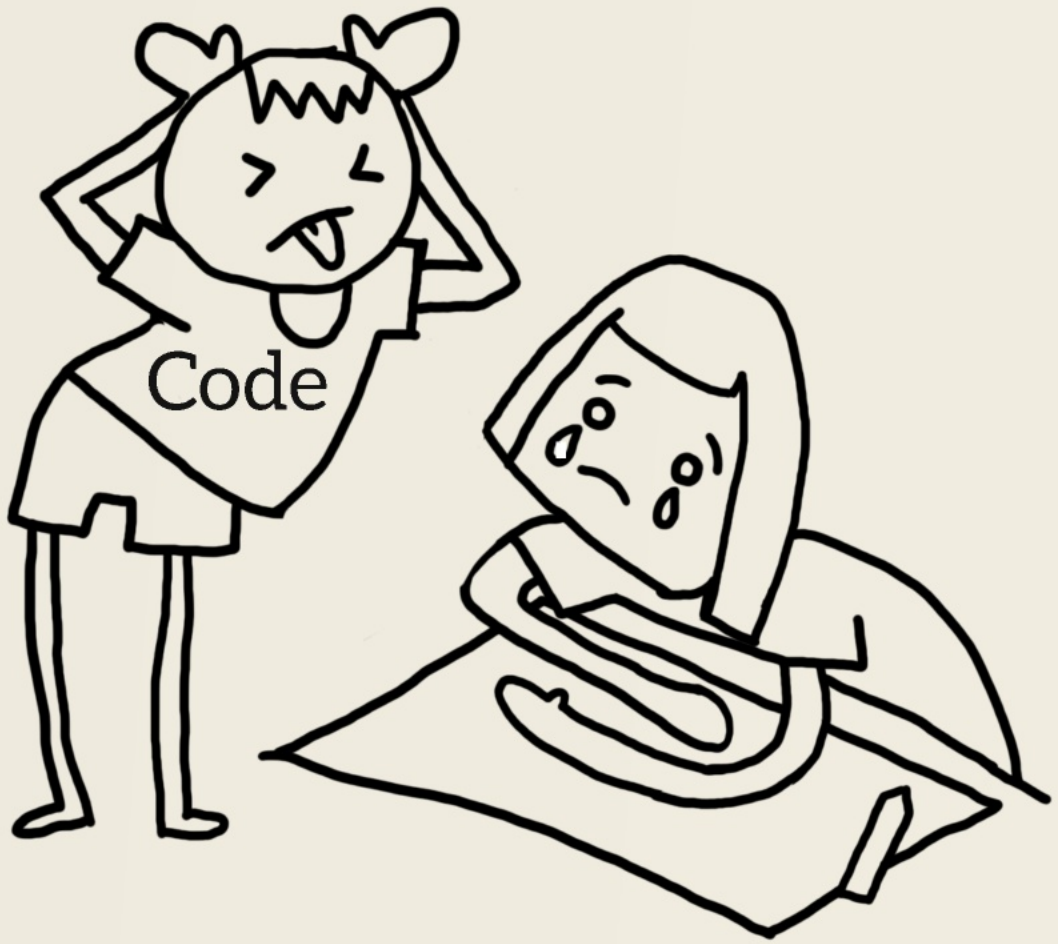
Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)



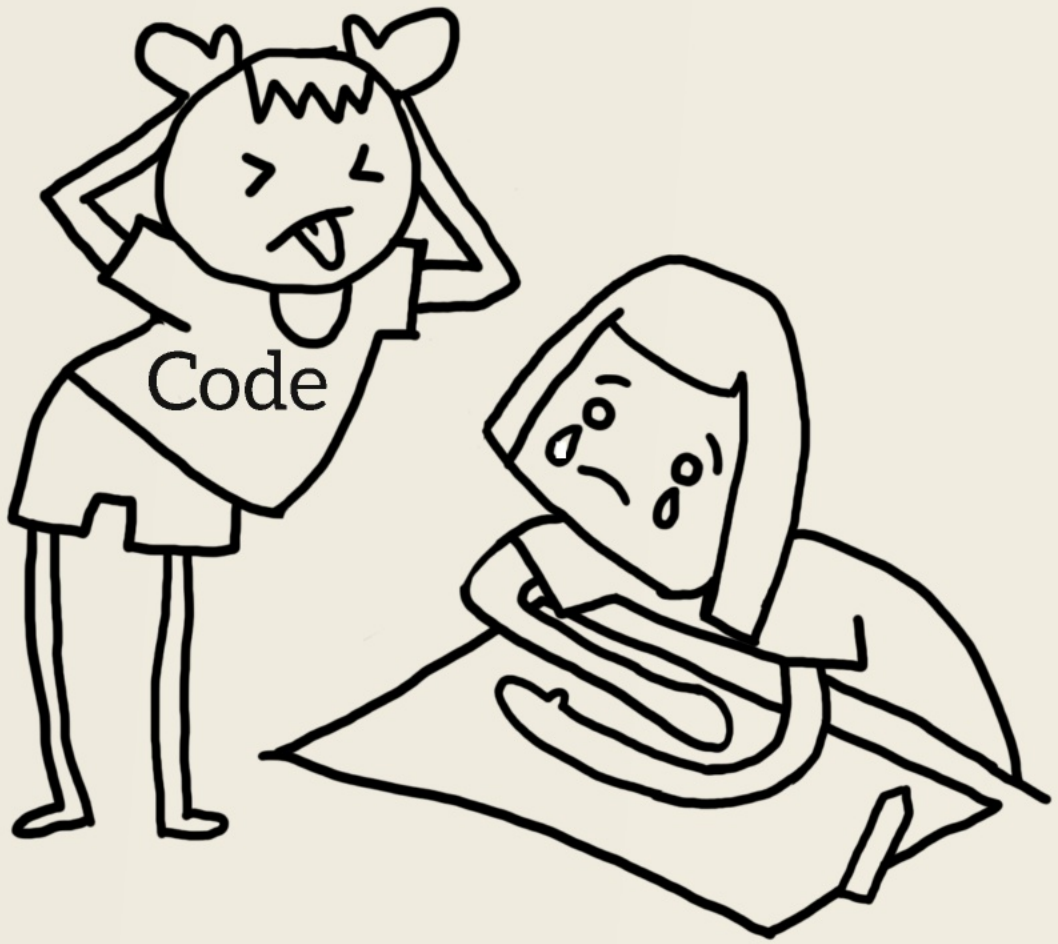
Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)



Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)
- No (open-source) one-stop shop that is:
    - **easy** to use (developer friendly)
    - **distributed** by default
    - uses **existing** infrastructure
    - includes **monitoring**
    - has **resilience** built-in



Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)
- No (open-source) one-stop shop that is:
    - **easy** to use (developer friendly)
    - **distributed** by default
    - uses **existing** infrastructure
    - includes **monitoring**
    - has **resilience** built-in

I believe...

we're not all wo

ve...

we're not all working for Facebook, LinkedIn or Netflix and process petabytes of data per day (or is it hour?).

solve

We often need to solve
complex business processes
with a moderate amount of
data.
(gigabytes, maybe terabytes).

We don't need ex

ybe terabytes).

We don't need extra infrastructure just to run some Background Jobs. Let's KISS and reuse proven technology like SQL and NoSQL databases.

unning jobs

databases.

Creating and running jobs should be fast and easy!

Failing i

Failing jobs should be retried by default and only bug me if they don't succeed after a while

Failing jobs should be retried by default and only bug me if they don't succeed after a while

It should be Resilient

We should have instant insights in how our jobs are doing...

Such a tool should be self-maintaining and be plug & play

I believe...

we're not all working for Facebook, LinkedIn or Netflix and process petabytes of data per day (or is it hour?).

We often need to solve
complex business processes
with a moderate amount of
data.
(gigabytes, maybe terabytes).

We don't need extra infrastructure just to run some Background Jobs. Let's KISS and reuse proven technology like SQL and NoSQL databases.

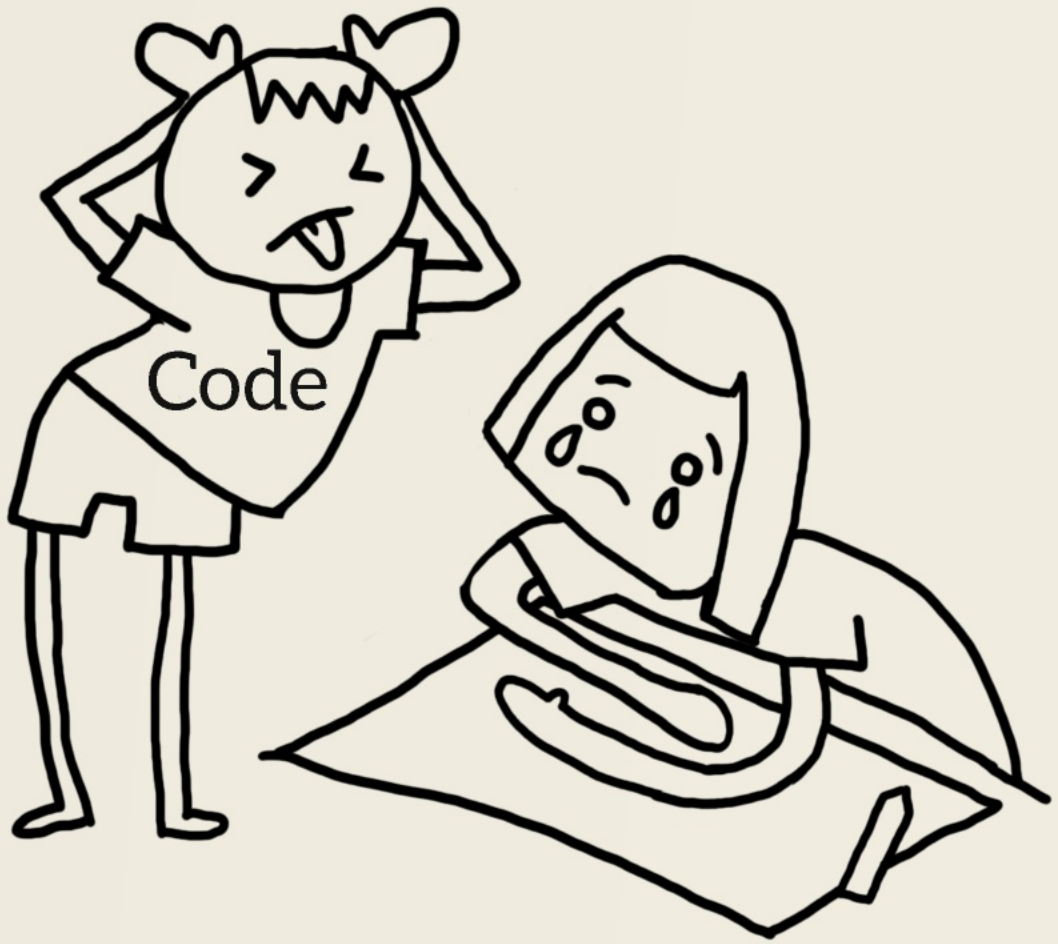
Creating and running jobs should be fast and easy!

Failing jobs should be retried by default and only bug me if they don't succeed after a while

It should be Resilient

We should have instant insights in how our jobs are doing...

Such a tool should be self-maintaining and be plug & play



Other solutions?

- Either **vendor specific** (cloud based solutions)
- Require **extra infrastructure** (Apache Kafka, Apache Hadoop, Apache Spark)
- No (open-source) one-stop shop that is:
  - **easy** to use (developer friendly)
  - **distributed** by default
  - uses **existing** infrastructure
  - includes **monitoring**
  - **retries** failed jobs automatically

JobRunr was born!

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

Resilient - retries automatically

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

Resilient - retries automatically

Dashboard - monitoring included!

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

Resilient - retries automatically

Dashboard - monitoring included!

Integrates with the **BEST**

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

Resilient - retries automatically

Dashboard - monitoring included!

Integrates with the **BEST**

spring boot

MICRONAUT™

QUARKUS


SUPER GIRL
BEST-
SELLER

```
BackgroundJob.enqueue(() -> System.out.println(
        "This is all you need for distributed job processing"
));
```

JobRunr DASHBOARD JOBS 25832 RECURRING JOBS 2 SERVERS 1

Dashboard

Estimated processing time	Uptime	Avg Process Cpu Load	Avg Process Memory Usage	Avg Process Free Memory
2 hours	27 minutes	0.30 %	204.5 MB	8.2 GB

Realtime graph

JobRunr was born!

Simple - just use Java 8 lambda's to create a background job.

Distributed & cluster-friendly

Uses existing infrastructure (either a RDMBS or a noSQL data store)

Embeddable - just add one dependency

Resilient - retries automatically

Dashboard - monitoring included!

Integrates with the **BEST**

spring boot

MICRONAUT™

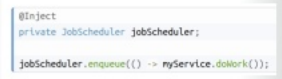
QUARKUS

JobRunr

Deep dive


< Linus Torvalds, Developer of Linux >
Talk is cheap.Show me_
{ THE CODE! }
</ Linus Torvalds, Developer of Linux >
TopHost.gr
#Programming

```
UUID jobId = BackgroundJob.enqueue(() -> myService.doWork());
```



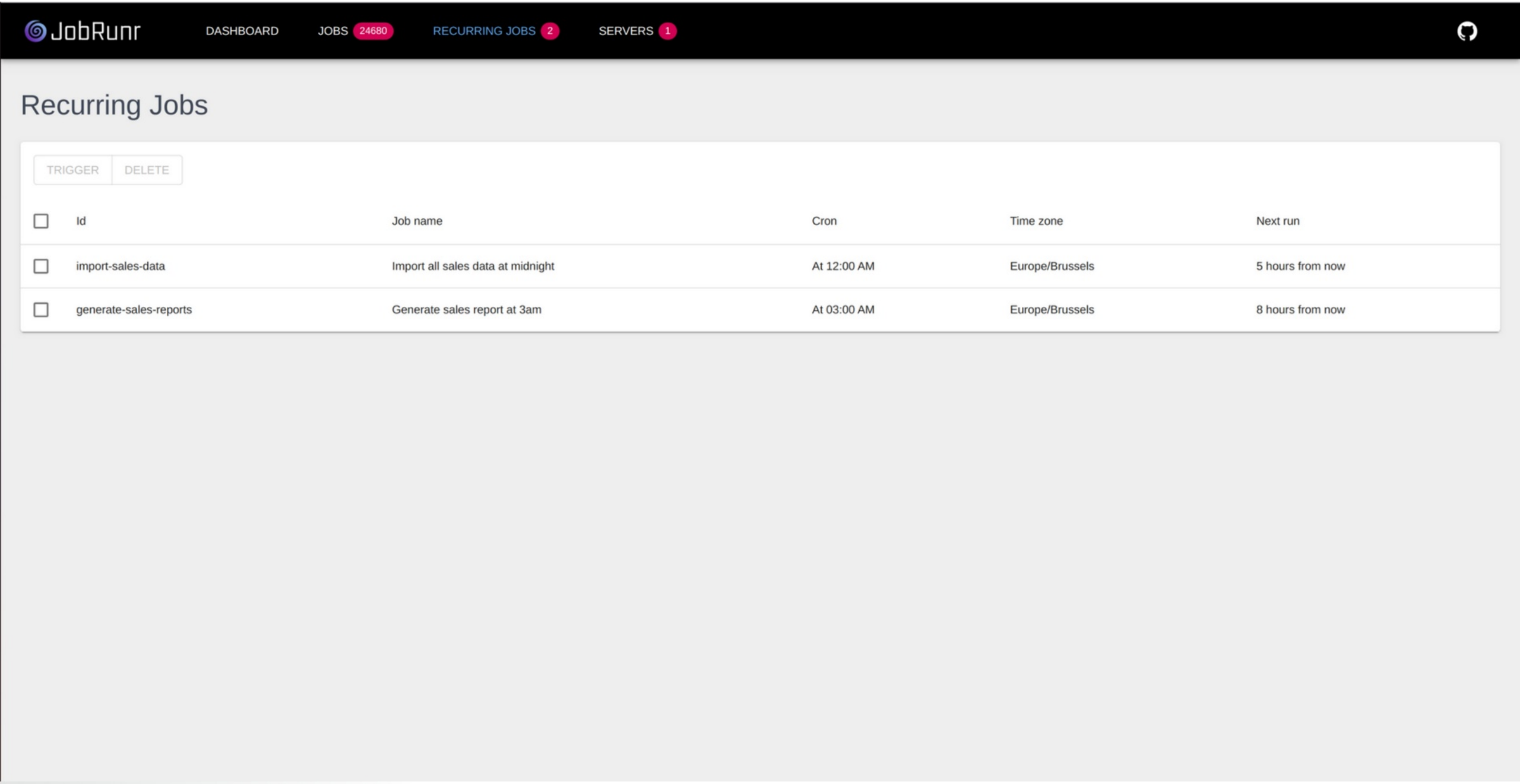
```
@Inject
private JobScheduler jobScheduler;


jobScheduler.enqueue(() -> myService.doWork());
```

```
Stream<User> userStream = userRepository.getAllUsers();
BackgroundJob.enqueue(userStream, (user) -> mailService.send(user.getId(), "mail-template-key"));
```

```
BackgroundJob.schedule<EmailService>(
    Instant.now().plusHours(24),
    x -> x.sendNewlyRegisteredEmail());
```

```
BackgroundJob.scheduleRecurrently(
   Cron.daily(),
   () -> System.out.println("Easy!"));
```


JobRunr
DASHBOARD
JOBS 24680
RECURRING JOBS 2
SERVERS 1
Recurring Jobs
TRIGGER
DELETE
Id
Job name
Cron
Time zone
Next run
import-sales-data
Import all sales data at midnight
At 12:00 AM
Europe/Brussels
5 hours from now
generate-sales-reports
Generate sales report at 3am
At 03:00 AM
Europe/Brussels
8 hours from now

Default queue
Scheduled 2
Enqueued 25064
Processing 128
Succeeded 7810
Failed 1

Job Id: ff3a62d3-5f71-46c1-8c7f-530f0f0ddf0a

REQUEUE DELETE

failed job

```
import java.lang.System;

System.out.println(a test);
```

History

Job scheduled 11 hours ago

Job enqueued 30 minutes ago

Processing job 30 minutes ago

Job processing failed - An exception occurred 30 minutes ago

java.lang.IllegalStateException

java.lang.IllegalStateException
    at org.jobrunr.jobs.JobTestBuilder.aFailedJobWithRetries(JobTestBuilder.java:121)
    at org.jobrunr.storage.SimpleStorageProvider.withALotOfEnqueuedJobsThatTakeSomeTime(SimpleStorageProvider.java:269)
    at org.jobrunr.dashboard.FrontEndDevelopment.main(FrontEndDevelopment.java:13)

```java
@Job(name="Job Name", retries=2)
public void doWorkWithCustomJobFilters() {
    System.out.println("I will only be retried two times ");
}
```

JobRunr

Deep dive

JobRunr internals

1. Analyse lambda

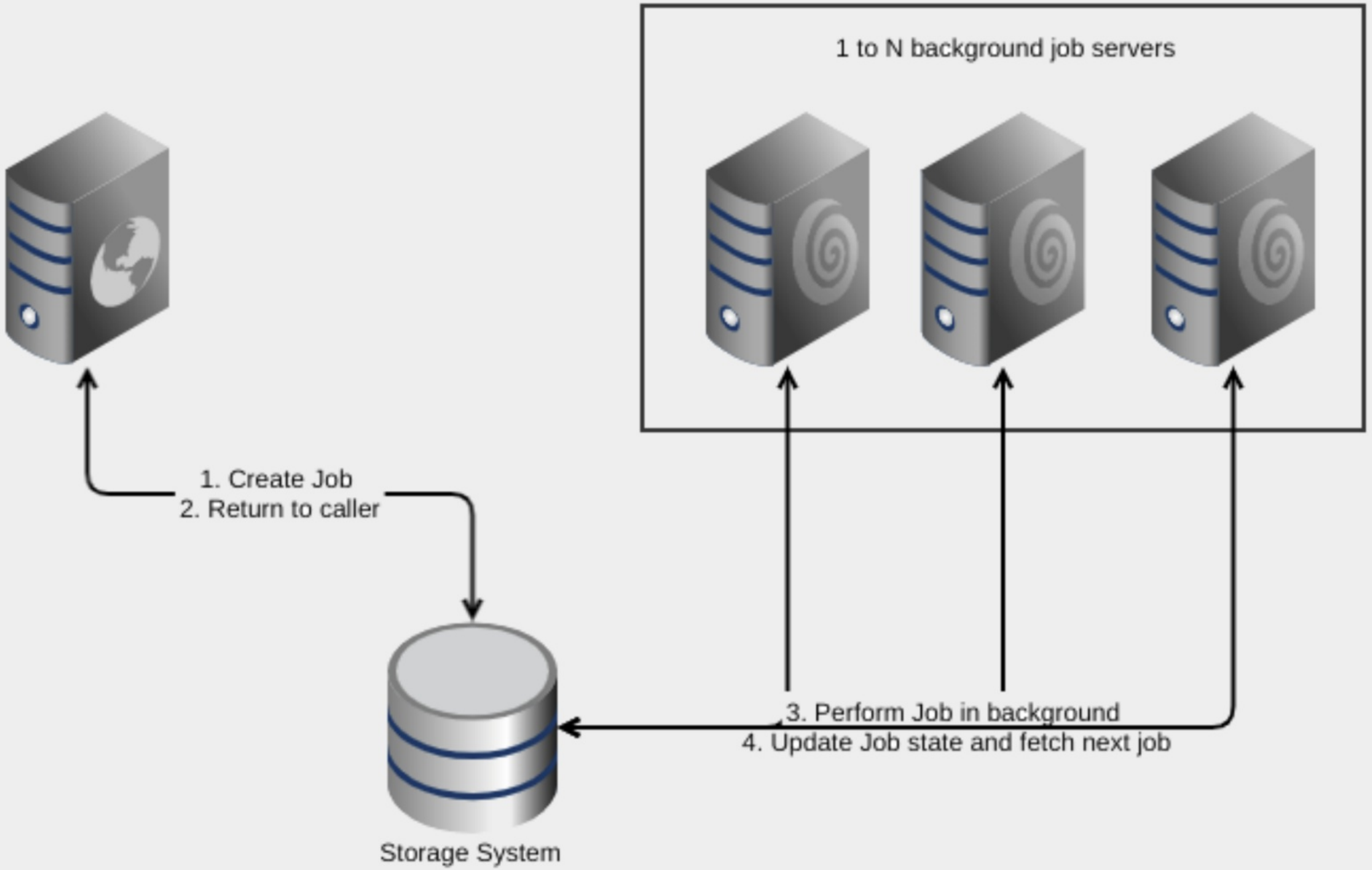
2. Convert to JSON

3. Store in RDBMS or NoSQL

4. Read from RDBMS or NoSQL

5. Process actual Job

6. Store Job outcome


1 to N background job servers
1. Create Job
2. Return to caller
3. Perform Job in background
4. Update Job state and fetch next job
Storage System

JobRunr internals

1. Analyse lambda

2. Convert to JSON

3. Store in RDBMS or NoSQL

4. Read from RDBMS or NoSQL

5. Process actual Job

6. Store Job outcome

1. Analyze lambda using ASM

```
BackgroundJob.enqueue(() -> System.out.println("This is all you need for distributed jobs!"));
```

```
// access flags 0x100A
private static synthetic lambda$testJobLambdaCallingInlineStaticMethod$efb093db$1()V throws java/lang/Exception
 L0
  LINENUMBER 75 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "This is all you need for distributed jobs!"
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
  RETURN
  MAXSTACK = 2
  MAXLOCALS = 0
```

```
BackgroundJob.enqueue(() -> testService.doWork( aString: "some string",  count: 1, Instant.now()));
```

```
// access flags 0x1002
private synthetic lambda$testJobLambdaWithMultipleParameters$a98ed11d$1()V throws java/lang/Exception
 L0
  LINENUMBER 176 L0
  ALOAD 0
  GETFIELD org/jobrunr/jobs/details/JobDetailsAsmGeneratorTest.testService : Lorg/jobrunr/stubs/TestService;
  LDC "some string"
  ICONST_1
  INVOKESTATIC java/time/Instant.now ()Ljava/time/Instant;
  INVOKEVIRTUAL org/jobrunr/stubs/TestService.doWork (Ljava/lang/String;ILjava/time/Instant;)V
  RETURN
```

Isn't this dangerous?

1. Analyze lambda using ASM

```
BackgroundJob.enqueue(() -> System.out.println("This is all you need for distributed jobs!"));
```

```
// access flags 0x100A
private static synthetic lambda$testJobLambdaCallingInlineStaticMethod$efb093db$1()V throws java/lang/Exception
 L0
  LINENUMBER 75 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "This is all you need for distributed jobs!"
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
  RETURN
  MAXSTACK = 2
  MAXLOCALS = 0
```

1. Analyze lambda using ASM

```
BackgroundJob.enqueue(() -> System.out.println("This is all you need for distributed jobs!"));
```

```
// access flags 0x100A
private static synthetic lambda$testJobLambdaCallingInlineStaticMethod$efb093db$1()V throws java/lang/Exception
 L0
  LINENUMBER 75 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "This is all you need for distributed jobs!"
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
  RETURN
  MAXSTACK = 2
  MAXLOCALS = 0
```

```
BackgroundJob.enqueue(() -> testService.doWork( aString: "some string",  count: 1, Instant.now()));
```

```
// access flags 0x1002
```

```
 L0
  LINENUMBER 75 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "This is all you need for distributed jobs!"
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
  RETURN
  MAXSTACK = 2
  MAXLOCALS = 0
```

```
BackgroundJob.enqueue(() -> testService.doWork( aString: "some string",  count: 1, Instant.now()));
```

```
// access flags 0x1002
private synthetic lambda$testJobLambdaWithMultipleParameters$a98ed11d$1()V throws java/lang/Exception
 L0
  LINENUMBER 176 L0
  ALOAD 0
  GETFIELD org/jobrunr/jobs/details/JobDetailsAsmGeneratorTest.testService : Lorg/jobrunr/stubs/TestService;
  LDC "some string"
  ICONST_1
  INVOKESTATIC java/time/Instant.now ()Ljava/time/Instant;
  INVOKEVIRTUAL org/jobrunr/stubs/TestService.doWork (Ljava/lang/String;ILjava/time/Instant;)V
```

```
BackgroundJob.enqueue(() -> testService.doWork( aString: "some string",  count: 1, Instant.now()));
```

```
// access flags 0x1002
private synthetic lambda$testJobLambdaWithMultipleParameters$a98ed11d$1()V throws java/lang/Exception
 L0
  LINENUMBER 176 L0
  ALOAD 0
  GETFIELD org/jobrunr/jobs/details/JobDetailsAsmGeneratorTest.testService : Lorg/jobrunr/stubs/TestService;
  LDC "some string"
  ICONST_1
  INVOKESTATIC java/time/Instant.now ()Ljava/time/Instant;
  INVOKEVIRTUAL org/jobrunr/stubs/TestService.doWork (Ljava/lang/String;ILjava/time/Instant;)V
  RETURN
```

Isn't this dangerous?

Your library may be nice at first glance, but there is so much black magic going on that just should never be relied upon.

For Java programs, there are three main categories of compatibility:

1. **Source:** Source compatibility concerns translating Java source code into class files.
2. **Binary:** Binary compatibility is defined in *The Java Language Specification* as preserving the ability to link without error.
3. **Behavioral:** Behavioral compatibility includes the semantics of the code that is executed at runtime.

FOSS (Oracle Quality Outreach Program)	Contact	OpenJDK 11.0.6	OpenJDK 14.0.2	OpenJDK 15 GA	OpenJDK 16 b26	Comments
JobRunr	Ronald Dehuysser	★	★	★	★	JobRunr supports JDK 15 &JDK 16 EA builds. JobRunr works with records out-of-the-box - JDK 16 b26 JobRunr works with project loom and all is ok!

```
public class JdkTest {

    @Test
    void jdk8OpenJdk() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:8-jdk-hotspot")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8OpenJ9() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:8-jdk-openj9")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8Zulu() { assertThat(buildAndTestOnImage( dockerfile: "azul/zulu-openjdk:8")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8GraalVM() {...}
```

```java
public class JdkTest {

    @Test
    void jdk8OpenJdk() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:8-jdk-hotspot")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8OpenJ9() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:8-jdk-openj9")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8Zulu() { assertThat(buildAndTestOnImage( dockerfile: "azul/zulu-openjdk:8")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk8GraalVM() {...}

    @Test
    void jdk8Ibm() { assertThat(buildAndTestOnImage( dockerfile: "ibmcom/ibmjava:8-sdk-alpine")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk11OpenJdk() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:11-jdk-hotspot")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk11OpenJ9() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:11-jdk-openj9")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk11Zulu() { assertThat(buildAndTestOnImage( dockerfile: "azul/zulu-openjdk:11")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk11GraalVM() {...}

    @Test
    void jdk14OpenJdk() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:14-jdk-hotspot")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk14OpenJ9() { assertThat(buildAndTestOnImage( dockerfile: "adoptopenjdk:14-jdk-openj9")).contains("BUILD SUCCESSFUL"); }

    @Test
    void jdk14Zulu() { assertThat(buildAndTestOnImage( dockerfile: "azul/zulu-openjdk:14")).contains("BUILD SUCCESSFUL"); }
```

No!

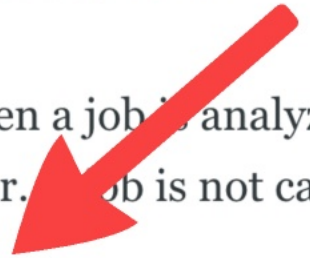
Celebration time!

I'm pleased to announce the release of JobRunr v4.0.0 (which is now available via Maven Central) and JobRunr Pro v4.0.0 which is available for customers with a subscription. As this is a major release, there are also some small breaking changes.

Some great new features to both JobRunr 4 and JobRunr Pro 4!

New features in JobRunr

This release adds a ton of new improvements to JobRunr:



- **Job Analysis Performance mode**: when a job is analyzed for the first time, JobRunr checks whether it can be cached. If so, all subsequent calls will be a lot faster. If a job is not cacheable this is displayed in the dashboard.
- **JobRequest and JobRequestHandler**: this release introduces the `JobRequest` and `JobRequesthandler`. This is a new way to create jobs using the command / commandhandler pattern (meaning you can use objects instead of lambda's to create background jobs). Read all about it in the updated documentation section.

```java
4 usages
public class SendEmailJobRequest implements JobRequest {

    3 usages
    String to;
    2 usages
    String from;
    3 usages
    String subject;
    2 usages
    String body;

    1 usage
    public SendEmailJobRequest(String to, String from, String subject, String body) {
        this.to = to;
        this.from = from;
        this.subject = subject;
        this.body = body;
    }

    1 usage
    @Override
    public Class<SendEmailJobRequestHandler> getJobRequestHandler() {
        return SendEmailJobRequestHandler.class;
    }

    @Override
    public String toString() {
        return "Sending email to " + to + " regarding " + subject;
    }
}
```

```
3 usages
public record SendEmailJobRequest(String to, String from, String subject, String body)
        implements JobRequest {

    1 usage
    @Override
    public Class<SendEmailJobRequestHandler> getJobRequestHandler() {
        return SendEmailJobRequestHandler.class;
    }
}
```

```java
2 usages
@Component
public class SendEmailJobRequestHandler implements JobRequestHandler<SendEmailJobRequest> {

    2 usages
    private JavaMailSender javaMailSender;

    public SendEmailJobRequestHandler(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }

    @Override
    @Job(name = "%0", retries = 2)
    public void run(SendEmailJobRequest jobRequest) throws Exception {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(jobRequest.from);
        message.setTo(jobRequest.to);
        message.setSubject(jobRequest.subject);
        message.setText(jobRequest.body);
        javaMailSender.send(message);
    }
}
```

```
BackgroundJobRequest.enqueue(new SendEmailRequest(
        to: "info@nljug.org",
        from: "ronald@jobrunr.io",
        subject: "JFall was awesome!",
        body: "Hi JFall Team, I quickly wanted to share that I had a blast on JFall!" +
              "The audience was really great and I look forward to next year!" +
              "Ronald"   You, Moments ago • Uncommitted changes
));
```

1. Analyze lambda using ASM

```
BackgroundJob.enqueue(() -> System.out.println("This is all you need for distributed jobs!"));
```

```
// access flags 0x100A
private static synthetic lambda$testJobLambdaCallingInlineStaticMethod$efb093db$1()V throws java/lang/Exception
 L0
  LINENUMBER 75 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "This is all you need for distributed jobs!"
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
  RETURN
  MAXSTACK = 2
  MAXLOCALS = 0
```

```
BackgroundJob.enqueue(() -> testService.doWork( aString: "some string",  count: 1, Instant.now()));
```

```
// access flags 0x1002
private synthetic lambda$testJobLambdaWithMultipleParameters$a98ed11d$1()V throws java/lang/Exception
 L0
  LINENUMBER 176 L0
  ALOAD 0
  GETFIELD org/jobrunr/jobs/details/JobDetailsAsmGeneratorTest.testService : Lorg/jobrunr/stubs/TestService;
  LDC "some string"
  ICONST_1
  INVOKESTATIC java/time/Instant.now ()Ljava/time/Instant;
  INVOKEVIRTUAL org/jobrunr/stubs/TestService.doWork (Ljava/lang/String;ILjava/time/Instant;)V
  RETURN
```

Isn't this dangerous?

2. Convert to JSON

```
{
  "lambdaType": "org.jobrunr.jobs.lambdas.JobLambda",
  "className": "java.lang.System",
  "staticFieldName": "out",
  "methodName": "println",
  "jobParameters": [
    {
      "className": "java.lang.String",
      "object": "This is all you need for distributed jobs!"
    }
  ]
}
```

the serialized job details

```
"jobDetails": {
  "cacheable": true,
  "className": "org.jobrunr.stubs.TestService",
  "staticFieldName": null,
  "methodName": "doWorkWithPath",
  "jobParameters": [
    {
      "className": "java.nio.file.Path",
      "actualClassName": "sun.nio.fs.UnixPath",
      "object": "${json-unit.ignore}"
    }
  ]
},
```

Why JSON? Again it's all about kiss(ing):
- Readable & debuggable
- Easy to use in the dashboard
- ...

```
{
  "lambdaType": "org.jobrunr.jobs.lambdas.JobLambda",
  "className": "java.lang.System",
  "staticFieldName": "out",
  "methodName": "println",
  "jobParameters": [
    {
      "className": "java.lang.String",
      "object": "This is all you need for distributed jobs!"
    }
  ]
}
```

the serialized job details

```
"jobDetails": {
  "cacheable": true,
  "className": "org.jobrunr.stubs.TestService",
  "staticFieldName": null,
  "methodName": "doWorkWithPath",
  "jobParameters": [
    {
      "className": "java.nio.file.Path",
      "actualClassName": "sun.nio.fs.UnixPath",
      "object": "${json-unit.ignore}"
    }
  ]
},
```

Why JSON? Again it's all about kiss(ing):

- Readable & debuggable
- Easy to use in the dashboard
- ...

2. Convert to JSON

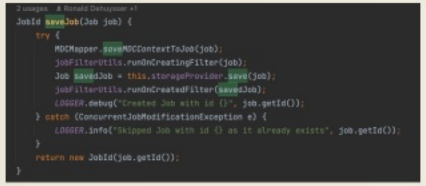
```
{
  "lambdaType": "org.jobrunr.jobs.lambdas.JobLambda",
  "className": "java.lang.System",
  "staticFieldName": "out",
  "methodName": "println",
  "jobParameters": [
    {
      "className": "java.lang.String",
      "object": "This is all you need for distributed jobs!"
    }
  ]
}
```

the serialized job details

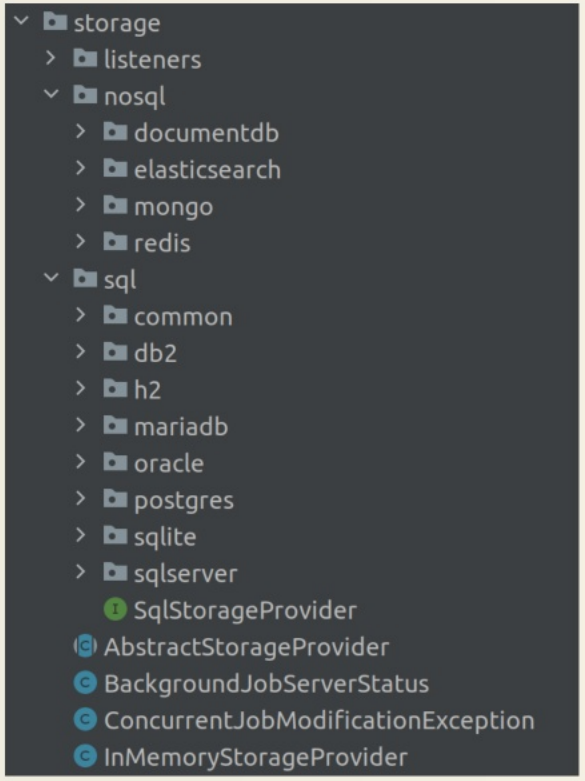
```
"jobDetails": {
  "cacheable": true,
  "className": "org.jobrunr.stubs.TestService",
  "staticFieldName": null,
  "methodName": "doWorkWithPath",
  "jobParameters": [
    {
      "className": "java.nio.file.Path",
      "actualClassName": "sun.nio.fs.UnixPath",
      "object": "${json-unit.ignore}"
    }
  ]
},
```

Why JSON? Again it's all about kiss(ing):
- Readable & debuggable
- Easy to use in the dashboard
- ...

3. Store in RDBMS or in NoSQL

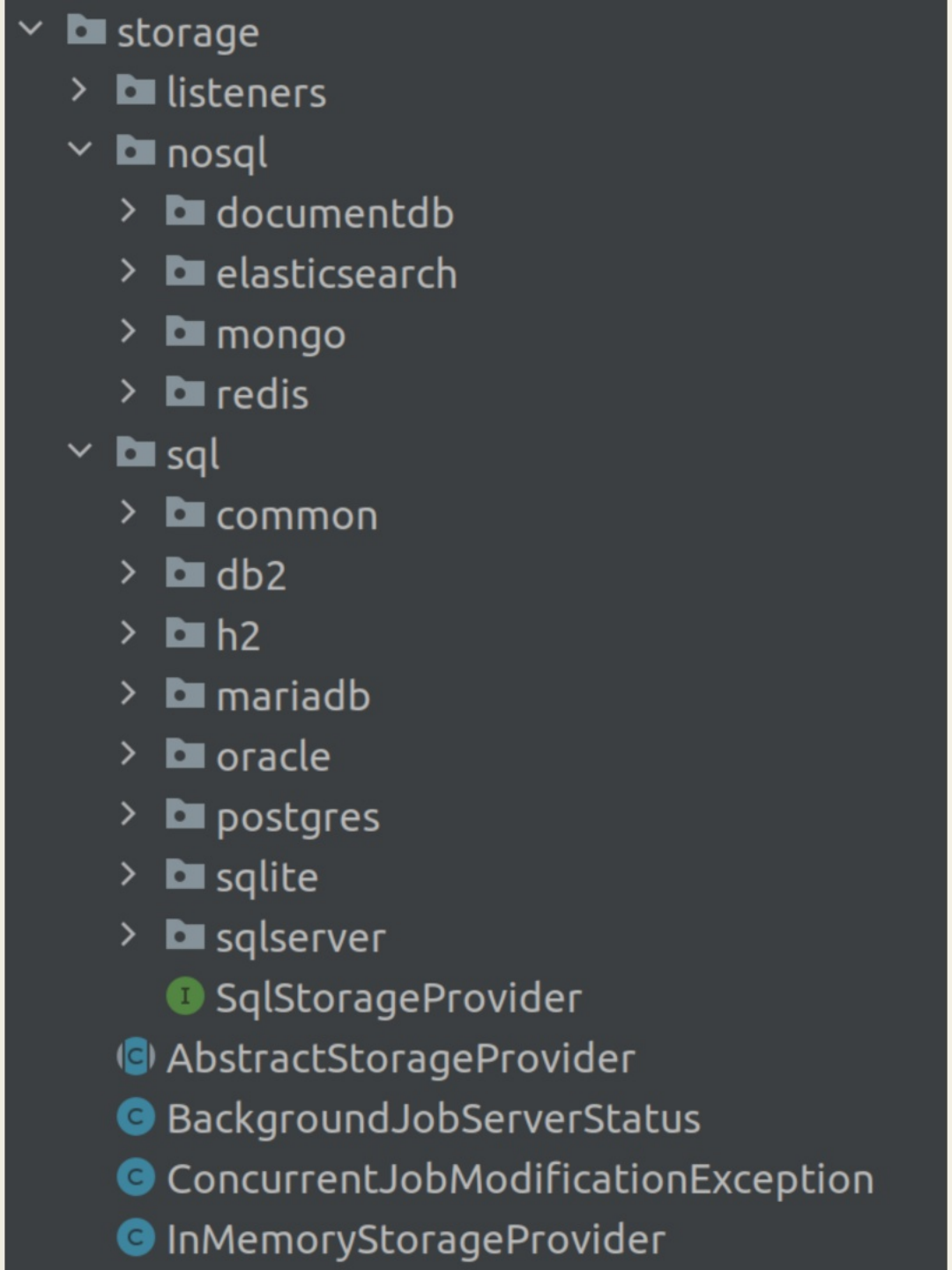


```
2 usages   & Ronald Dehuysser +1
JobId saveJob(Job job) {
    try {
        MDCMapper.saveMDCContextToJob(job);
        jobFilterUtils.runOnCreatingFilter(job);
        Job savedJob = this.storageProvider.save(job);
        jobFilterUtils.runOnCreatedFilter(savedJob);
        LOGGER.debug("Created Job with id {}", job.getId());
    } catch (ConcurrentJobModificationException e) {
        LOGGER.info("Skipped Job with id {} as it already exists", job.getId());
    }
    return new JobId(job.getId());
}
```

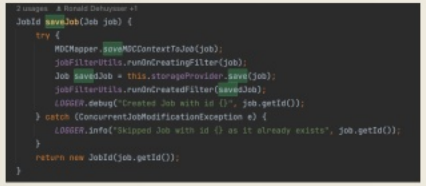


```
2 usages   Ronald Dehuysser +1
JobId saveJob(Job job) {
    try {
        MDCMapper.saveMDCContextToJob(job);
        jobFilterUtils.runOnCreatingFilter(job);
        Job savedJob = this.storageProvider.save(job);
        jobFilterUtils.runOnCreatedFilter(savedJob);
        LOGGER.debug("Created Job with id {}", job.getId());
    } catch (ConcurrentJobModificationException e) {
        LOGGER.info("Skipped Job with id {} as it already exists", job.getId());
    }
    return new JobId(job.getId());
}
```

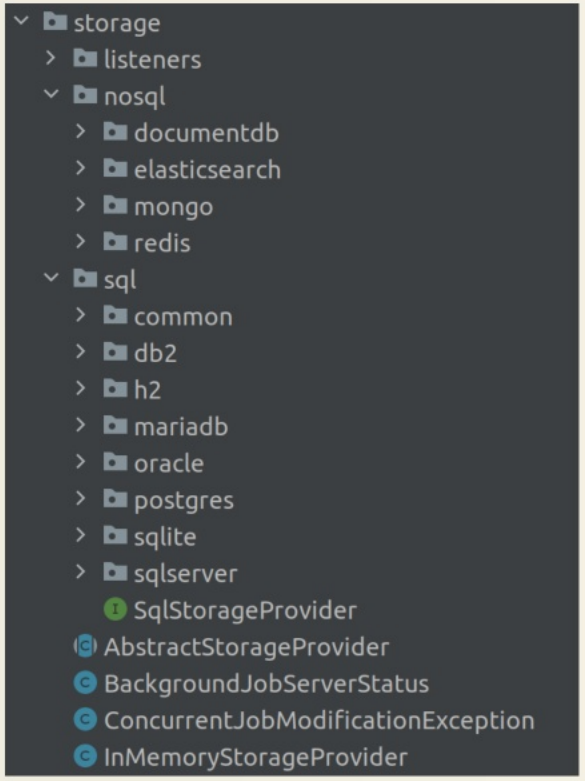

Dehuysser +1
ob job) {
er.saveMDCContextToJob(job);
erUtils.runOnCreatingFilter(job);
edJob = this.storageProvider.save(job);
erUtils.runOnCreatedFilter(savedJob);
debug("Created Job with id {}", job.getId());
ncurrentJobModificationException e) {
info("Skipped Job with id {} as it already exists", job.getId());
JobId(job.getId());


storage
listeners
nosql
documentdb
elasticsearch
mongo
redis
sql
common
db2
h2
mariadb
oracle
postgres
sqlite
sqlserver
SqlStorageProvider
AbstractStorageProvider
BackgroundJobServerStatus
ConcurrentJobModificationException
InMemoryStorageProvider

3. Store in RDBMS or in NoSQL



```
2 usages   & Ronald Dehuysser +1
JobId saveJob(Job job) {
    try {
        MDCMapper.saveMDCContextToJob(job);
        jobFilterUtils.runOnCreatingFilter(job);
        Job savedJob = this.storageProvider.save(job);
        jobFilterUtils.runOnCreatedFilter(savedJob);
        LOGGER.debug("Created Job with id {}", job.getId());
    } catch (ConcurrentJobModificationException e) {
        LOGGER.info("Skipped Job with id {} as it already exists", job.getId());
    }
    return new JobId(job.getId());
}
```



4. Read from RDBMS or NoSQL

```
void checkForEnqueuedJobs() {
    try {
        if (reentrantLock.tryLock()) {
            LOGGER.debug("Looking for enqueued jobs... ");
            final PageRequest workPageRequest = workDistributionStrategy.getWorkPageRequest();
            if (workPageRequest.getLimit() > 0) {
                final List<Job> enqueuedJobs = storageProvider.getJobs(StateName.ENQUEUED, workPageRequest);
                enqueuedJobs.forEach(backgroundJobServer::processJob);
            }
        }
    } finally {
        if (reentrantLock.isHeldByCurrentThread()) {
            reentrantLock.unlock();
        }
    }
}
```

```
private boolean updateJobStateToProcessingRunJobFiltersAndReturnIfProcessingCanStart() {
    try {
        job.startProcessingOn(backgroundJobServer);
        saveAndRunStateRelatedJobFilters(job);
        LOGGER.debug("Job(id={}, jobName='{}') processing started", job.getId(), job.getJobName());
        return job.hasState(PROCESSING);
    } catch (ConcurrentJobModificationException e) {
        // processing already started on other server
        return false;
    }
}
```

4. Read from RDBMS or NoSQL

```
void checkForEnqueuedJobs() {
    try {
        if (reentrantLock.tryLock()) {
            LOGGER.debug("Looking for enqueued jobs... ");
            final PageRequest workPageRequest = workDistributionStrategy.getWorkPageRequest();
            if (workPageRequest.getLimit() > 0) {
                final List<Job> enqueuedJobs = storageProvider.getJobs(StateName.ENQUEUED, workPageRequest);
                enqueuedJobs.forEach(backgroundJobServer::processJob);
            }
        }
    } finally {
        if (reentrantLock.isHeldByCurrentThread()) {
            reentrantLock.unlock();
        }
    }
}
```

```
private boolean updateJobStateToProcessingRunJobFiltersAndReturnIfProcessingCanStart() {
```

```
        }
    }
}
```

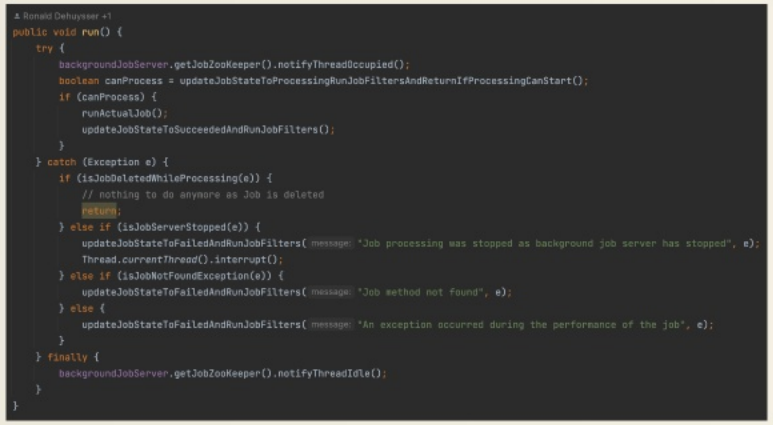
```
private boolean updateJobStateToProcessingRunJobFiltersAndReturnIfProcessingCanStart() {
    try {
        job.startProcessingOn(backgroundJobServer);
        saveAndRunStateRelatedJobFilters(job);
        LOGGER.debug("Job(id={}, jobName='{}') processing started", job.getId(), job.getJobName());
        return job.hasState(PROCESSING);
    } catch (ConcurrentJobModificationException e) {
        // processing already started on other server
        return false;
    }
}
```

4. Read from RDBMS or NoSQL

```
void checkForEnqueuedJobs() {
    try {
        if (reentrantLock.tryLock()) {
            LOGGER.debug("Looking for enqueued jobs... ");
            final PageRequest workPageRequest = workDistributionStrategy.getWorkPageRequest();
            if (workPageRequest.getLimit() > 0) {
                final List<Job> enqueuedJobs = storageProvider.getJobs(StateName.ENQUEUED, workPageRequest);
                enqueuedJobs.forEach(backgroundJobServer::processJob);
            }
        }
    } finally {
        if (reentrantLock.isHeldByCurrentThread()) {
            reentrantLock.unlock();
        }
    }
}
```

```
private boolean updateJobStateToProcessingRunJobFiltersAndReturnIfProcessingCanStart() {
    try {
        job.startProcessingOn(backgroundJobServer);
        saveAndRunStateRelatedJobFilters(job);
        LOGGER.debug("Job(id={}, jobName='{}') processing started", job.getId(), job.getJobName());
        return job.hasState(PROCESSING);
    } catch (ConcurrentJobModificationException e) {
        // processing already started on other server
        return false;
    }
}
```

5. Process actual job




```java
public void run() throws Exception {
    Class<?> jobToPerformClass = getJobToPerformClass();
    Object jobToPerform = getJobToPerform(jobToPerformClass);
    Method jobMethodToPerform = getJobMethodToPerform(jobToPerformClass);
    invokeJobMethod(jobToPerform, jobMethodToPerform);
}
```

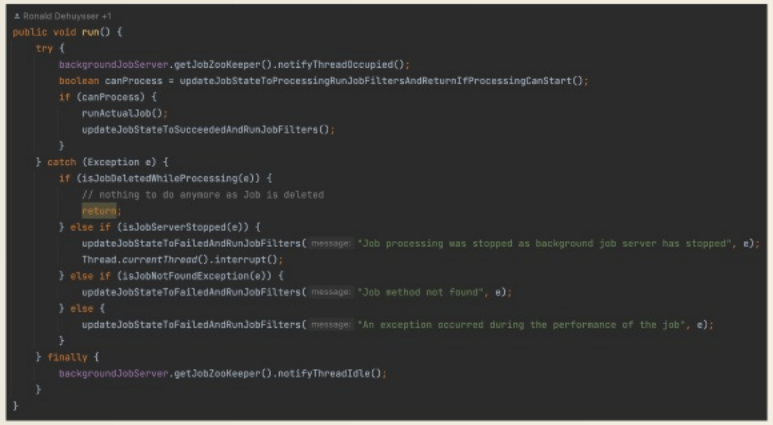
Ronald Dehuysser +1

```java
public void run() {
    try {
        backgroundJobServer.getJobZooKeeper().notifyThreadOccupied();
        boolean canProcess = updateJobStateToProcessingRunJobFiltersAndReturnIfProcessingCanStart();
        if (canProcess) {
            runActualJob();
            updateJobStateToSucceededAndRunJobFilters();
        }
    } catch (Exception e) {
        if (isJobDeletedWhileProcessing(e)) {
            // nothing to do anymore as Job is deleted
            return;
        } else if (isJobServerStopped(e)) {
            updateJobStateToFailedAndRunJobFilters( message: "Job processing was stopped as background job server has stopped", e);
            Thread.currentThread().interrupt();
        } else if (isJobNotFoundException(e)) {
            updateJobStateToFailedAndRunJobFilters( message: "Job method not found", e);
        } else {
            updateJobStateToFailedAndRunJobFilters( message: "An exception occurred during the performance of the job", e);
        }
    } finally {
        backgroundJobServer.getJobZooKeeper().notifyThreadIdle();
    }
}
```

```
        } else if (isJobNotFoundException(e)) {
            updateJobStateToFailedAndRunJobFilters( message: "Job method not found", e);
        } else {
            updateJobStateToFailedAndRunJobFilters( message: "An exception occurred during the performance of the job", e);
        }
    } finally {
        backgroundJobServer.getJobZooKeeper().notifyThreadIdle();
    }
}
```

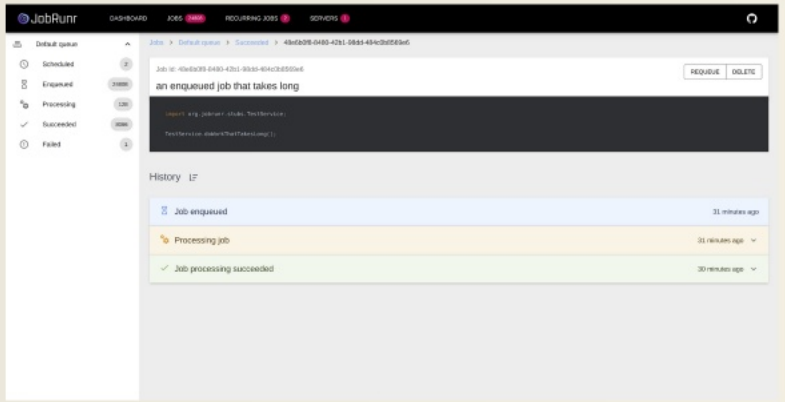
```
public void run() throws Exception {
    Class<?> jobToPerformClass = getJobToPerformClass();
    Object jobToPerform = getJobToPerform(jobToPerformClass);
    Method jobMethodToPerform = getJobMethodToPerform(jobToPerformClass);
    invokeJobMethod(jobToPerform, jobMethodToPerform);
}
```

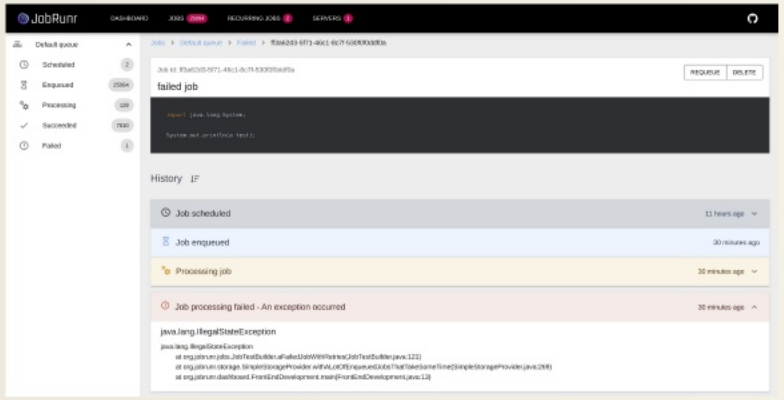
5. Process actual job

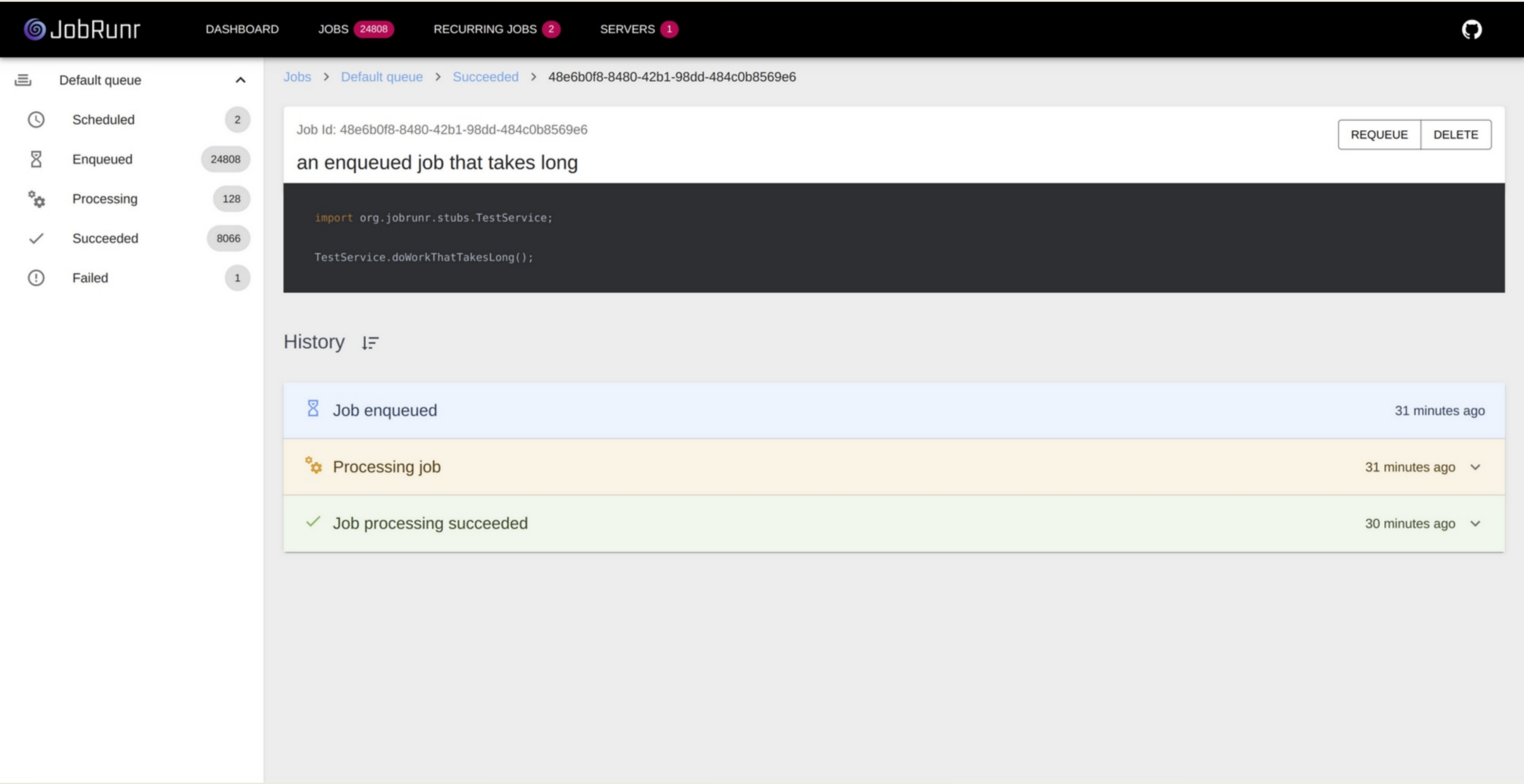


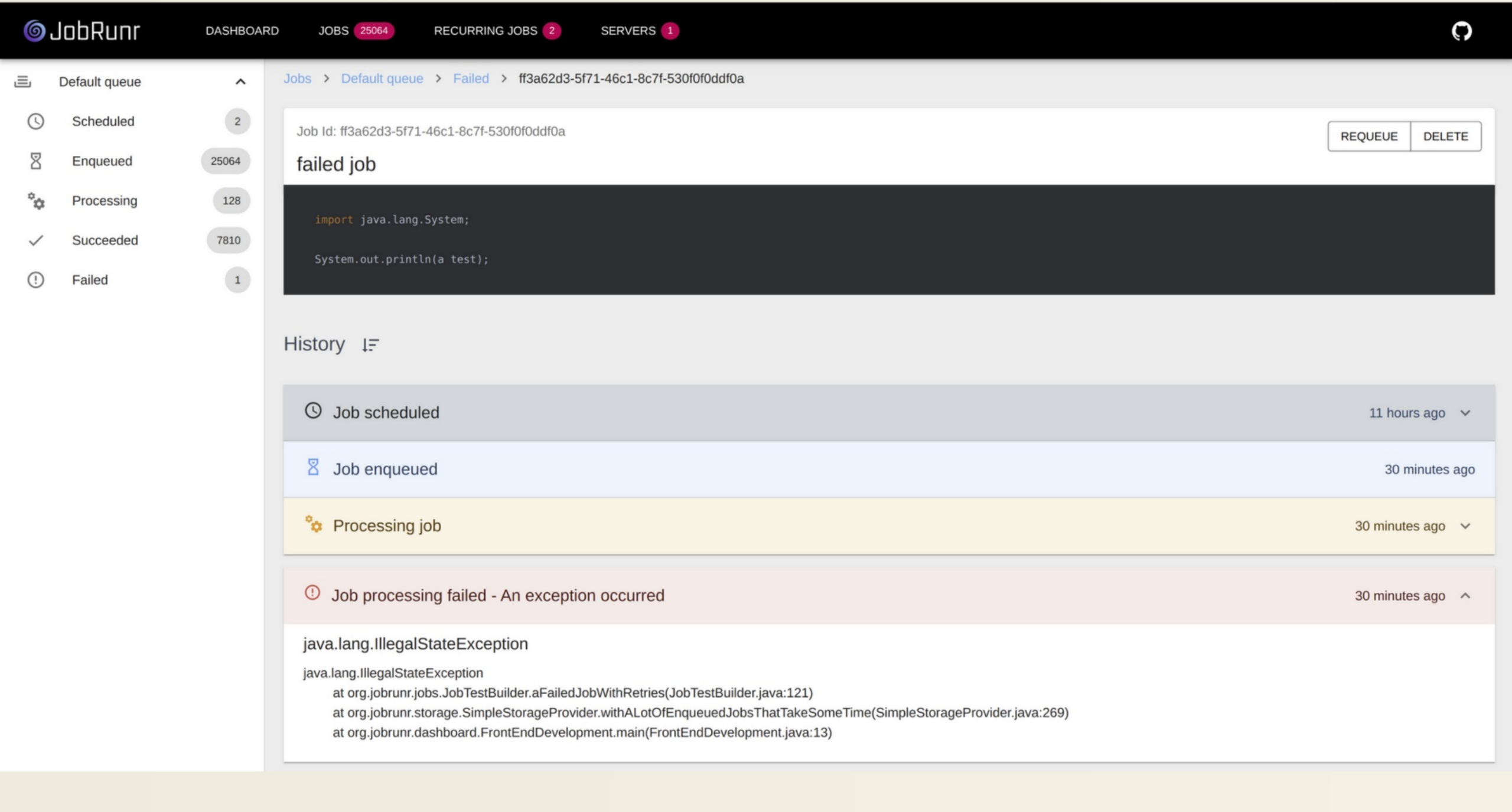

```java
public void run() throws Exception {
    Class<?> jobToPerformClass = getJobToPerformClass();
    Object jobToPerform = getJobToPerform(jobToPerformClass);
    Method jobMethodToPerform = getJobMethodToPerform(jobToPerformClass);
    invokeJobMethod(jobToPerform, jobMethodToPerform);
}
```

6. Store Job outcome

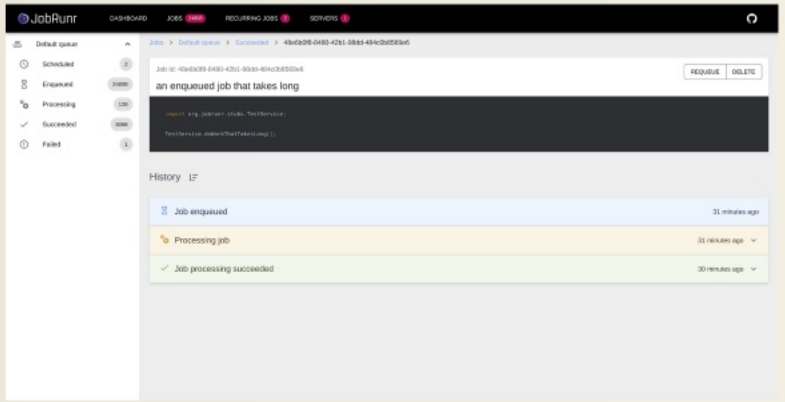


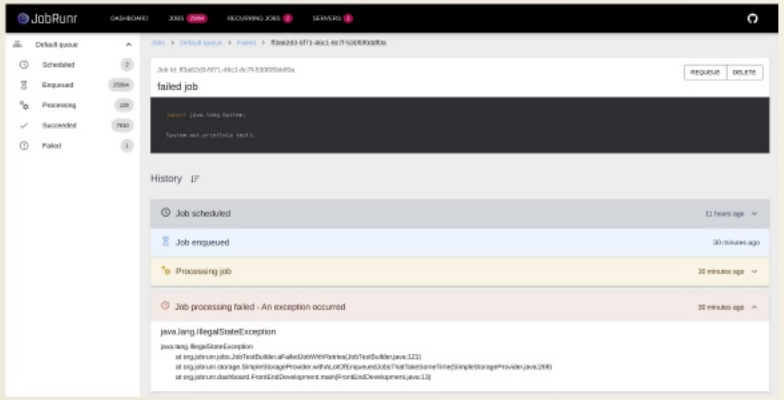



JobRunr
DASHBOARD
JOBS 24808
RECURRING JOBS 2
SERVERS 1
Default queue
Scheduled 2
Enqueued 24808
Processing 128
Succeeded 8066
Failed 1
Jobs > Default queue > Succeeded > 48e6b0f8-8480-42b1-98dd-484c0b8569e6
Job Id: 48e6b0f8-8480-42b1-98dd-484c0b8569e6
REQUEUE
DELETE
an enqueued job that takes long
import org.jobrunr.stubs.TestService;
TestService.doWorkThatTakesLong();
History
Job enqueued
31 minutes ago
Processing job
31 minutes ago
Job processing succeeded
30 minutes ago


JobRunr
DASHBOARD
JOBS 25064
RECURRING JOBS 2
SERVERS 1
Default queue
Scheduled 2
Enqueued 25064
Processing 128
Succeeded 7810
Failed 1
Jobs > Default queue > Failed > ff3a62d3-5f71-46c1-8c7f-530f0f0ddf0a
Job Id: ff3a62d3-5f71-46c1-8c7f-530f0f0ddf0a
REQUEUE
DELETE
failed job
import java.lang.System;
System.out.println(a test);
History
Job scheduled 11 hours ago
Job enqueued 30 minutes ago
Processing job 30 minutes ago
Job processing failed - An exception occurred 30 minutes ago
java.lang.IllegalStateException
java.lang.IllegalStateException
at org.jobrunr.jobs.JobTestBuilder.aFailedJobWithRetries(JobTestBuilder.java:121)
at org.jobrunr.storage.SimpleStorageProvider.withALotOfEnqueuedJobsThatTakeSomeTime(SimpleStorageProvider.java:269)
at org.jobrunr.dashboard.FrontEndDevelopment.main(FrontEndDevelopment.java:13)

6. Store Job outcome





JobRunr internals

1. Analyse lambda

2. Convert to JSON

3. Store in RDBMS or NoSQL

4. Read from RDBMS or NoSQL

5. Process actual Job

6. Store Job outcome

JobRunr

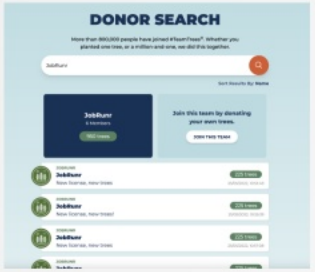
Deep dive

JobRunr today

- Used for Medical Image processing, Web crawling, Order fulfillment, Document generation, e-procurement, service desk automation...
- 300.000 downloads / month from Maven Central
- About 1bn jobs processed per day

JobRunr Pro with:

- Extended dashboard
- Spring @Transaction integration
- Queues, Batches, Job Chaining, Server tags, ...



Back to the future !

- Better GraalVM native support
- Faster!
- Feature requests from you!
- Big announcement !

JobRunr today

- Used for Medical Image processing, Web crawling, Order fulfillment, Document generation, e-procurement, service desk automation...

JobRunr Pro with:

- Extended dashbo
- Spring @Transact
- Queues, Batches, tags, ...

DONOR

- Used for Medical Image processing, Web crawling, Order fulfillment, Document generation, e-procurement, service desk automation...
- 300.000 downloads / month from Maven Central
- About 1bn jobs processed per day

JobRunr Pro with:

- Extended dashboard
- Spring @Transaction integration
- Queues, Batches, Job Chaining, Server tags, ...

DONOR SEARCH

More than 800,000 people have joined #TeamTrees®. Whether you planted one tree, or a million-and-one, we did this together.

JobRunr

Sort Results By: **Name**

JobRunr
6 Members
950 trees

Join this team by donating your own trees.

JOIN THIS TEAM

JOBRUNR
JobRunr
New license, new trees
225 trees
25/05/2022, 10:53:49

JOBRUNR
JobRunr
New license, new trees!
225 trees
25/05/2022, 10:55:39

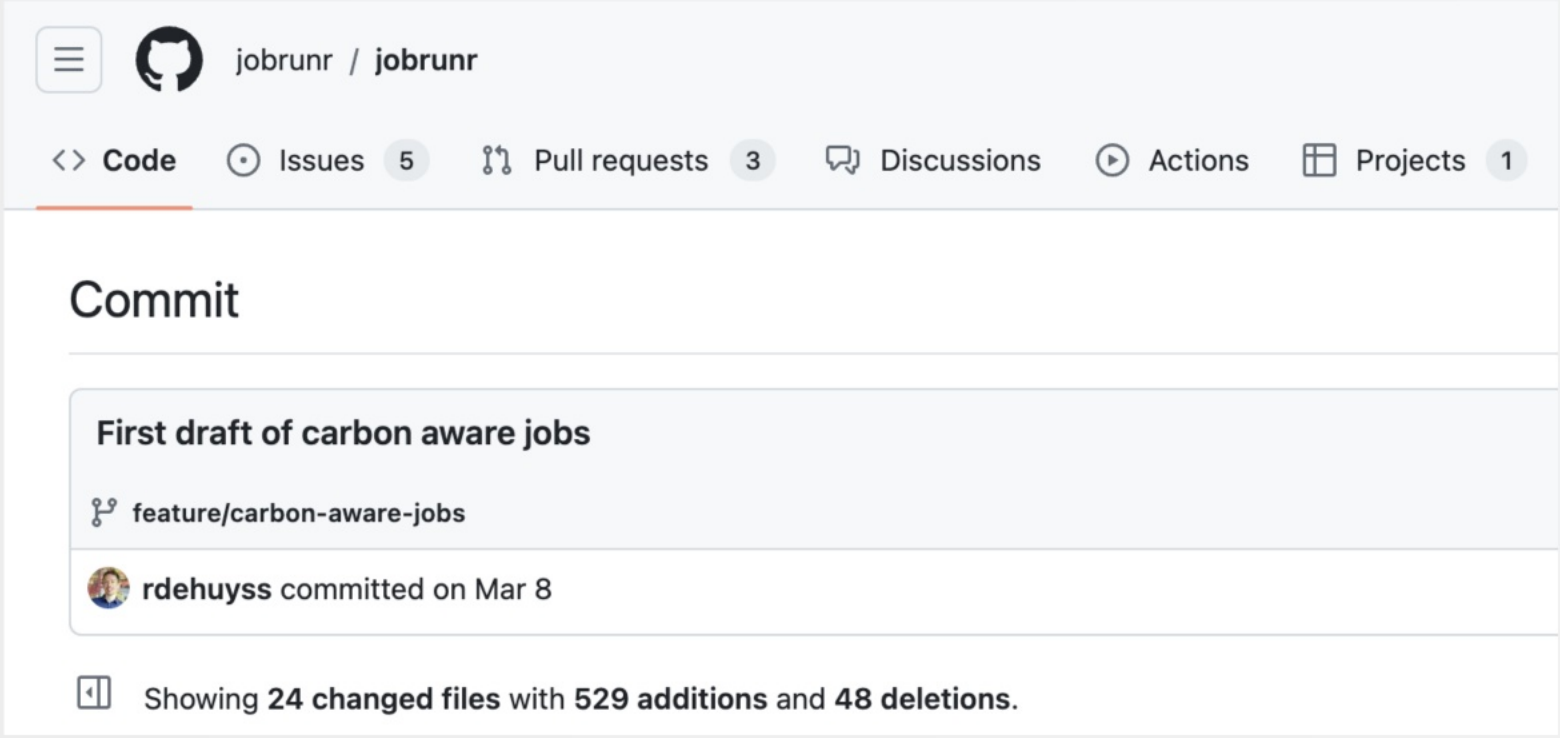
JOBRUNR
JobRunr
New license, new trees
225 trees
25/05/2022, 10:57:08

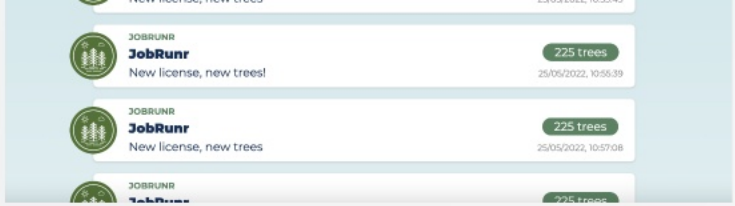
JOBRUNR
JobRunr
225 trees

Back to the future !

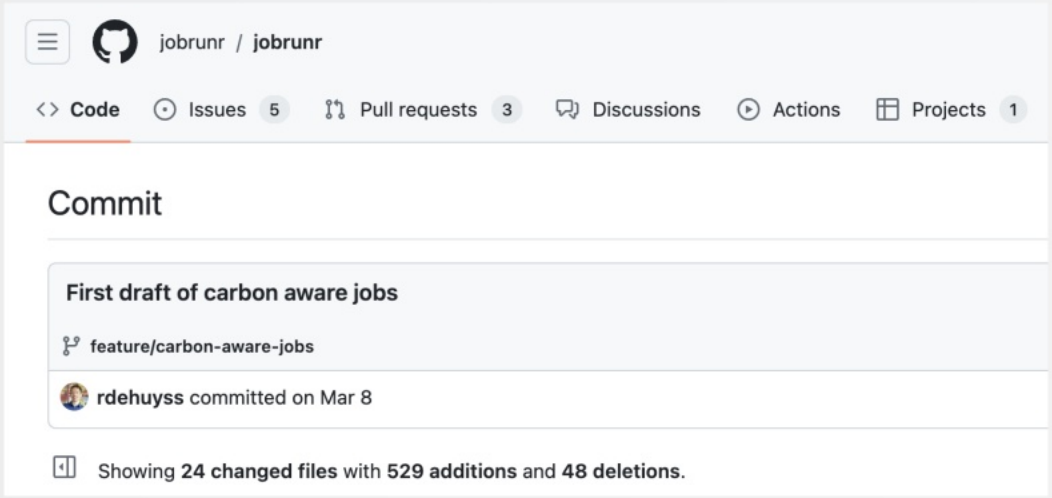
- Better GraalVM native support
- Faster!
- Feature requests from you!
- Big announcement !

Back to the future !





Back to the future !



JobRunr

Deep dive

But first...

A big shoutout to ...

Conclusion

Remember to ...

START IT
@KBC


With the support of
Flanders
Investment
& Trade
Flanders
State of
the Art

JobRunr DeepDive Conclusion

- only one dependency
- use a Java 8 lambda or JobRequest
- distributed by default
- built-in dashboard
- resilient thanks to automatic retries

Remember to ...

- give feedback on this talk!
- star the GitHub repo if you like JobRunr
- Convice your manager to ...
- Enjoy the rest of the conference!

But first...

JobRunr

Deep dive

Time to eat my own dog food

Live coding!

JobRunr

Deep dive