Turning Maven into a High scalable, resource efficient, cloud ready microservice

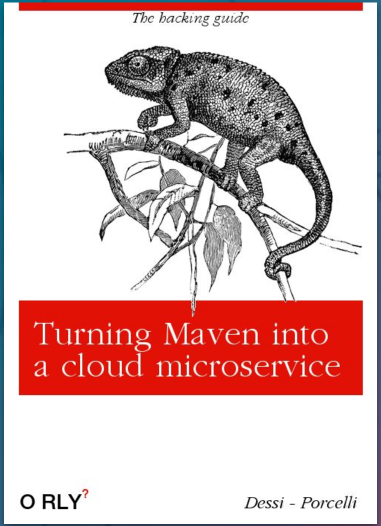
Alex Porcelli
Business Automation Architect

Max Dessì
Senior Software Engineer

Oracle Code One 24 Oct 2018

Speakers

Massimiliano Dessì has more than 18 years of experience in programming.
Co-founder of Java User Group Sardegna,
Manager of Google Developer Group Sardegna,
Author of Spring 2.5 AOP (Packt)
He works as a Senior Software Engineer for Red Hat in the BSIG (Business Systems and Intelligence Group) , on KIE projects (Knowledge Is Everything),
he lives in Cagliari, Sardinia, Europe.



The hacking guide

Turning Maven into a cloud microservice

O RLY?

Dessi - Porcelli

Alex Porcelli is Principal Software Engineer at JBoss by Red Hat, proud member of the architecture group behind the Drools, jBPM and Business Central platform, and co-founder of AppFormer. Professional developer since 1996, has been working exclusively on Open Source projects for almost a decade. Since joined Red Hat he has been focusing on web enabled of the Drools&jBPM platforms redefining all web tooling. Porcelli is also a frequent speaker in tech events like QCon, JavaOne, CodeOne, Red Hat Summit and DevNation.

QUICK INTRO TO BUSINESS CENTRAL

BUSINESS CENTRAL

ALL ABOUT BUSINESS PROCESS AND RULES

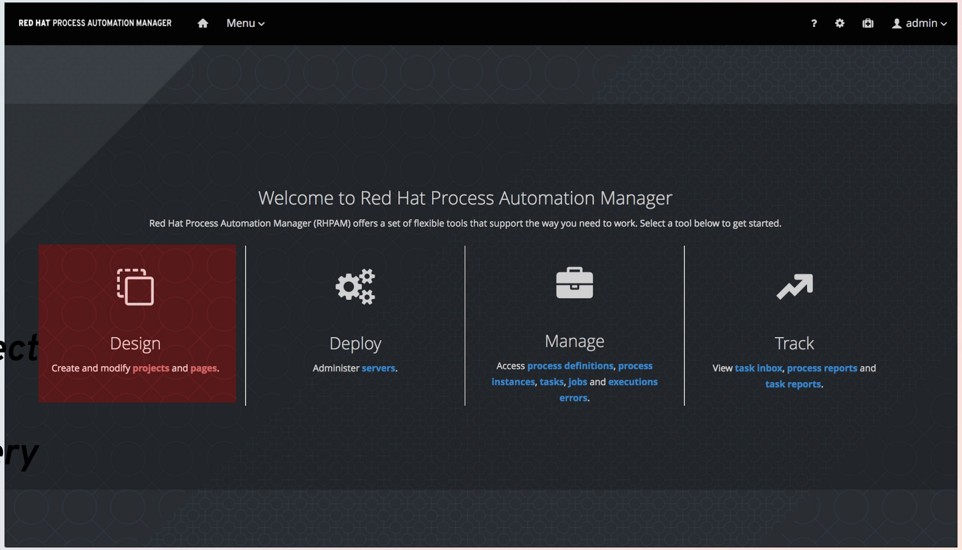
- KJAR
  - What is a KJAR
- Maven based packaging model
  - kie-maven-plugin
  - kie-takari-plugin

```
<groupId>com.example</groupId>
<artifactId>user-project</artifactId>
<packaging>kjar</packaging>

<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>7.10.0.Final</version>
      <extensions>true</extensions>
      </plugin>
  </plugins>
</build>
```

BUSINESS CENTRAL

ALL ABOUT BUSINESS PROCESS AND RULES

- Multiple Personas
- Design
  - Practitioners & Devs
  - Modeling tools
  - Hides Tech details (ie. POM)
  - Git backed storage
  - *Editors need access to project classloader for autocomplete*
    - *A build is needed on every user I/O action*



Welcome to Red Hat Process Automation Manager

Red Hat Process Automation Manager (RHPAM) offers a set of flexible tools that support the way you need to work. Select a tool below to get started.

Design
Create and modify projects and pages.

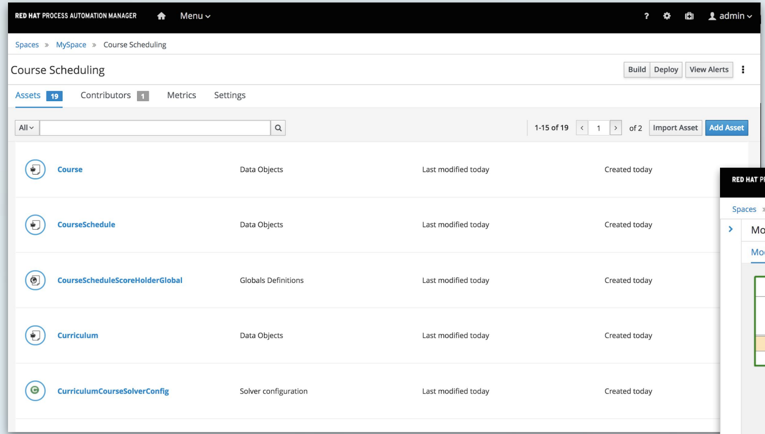
Deploy
Administer servers.

Manage
Access process definitions, process instances, tasks, jobs and executions errors.

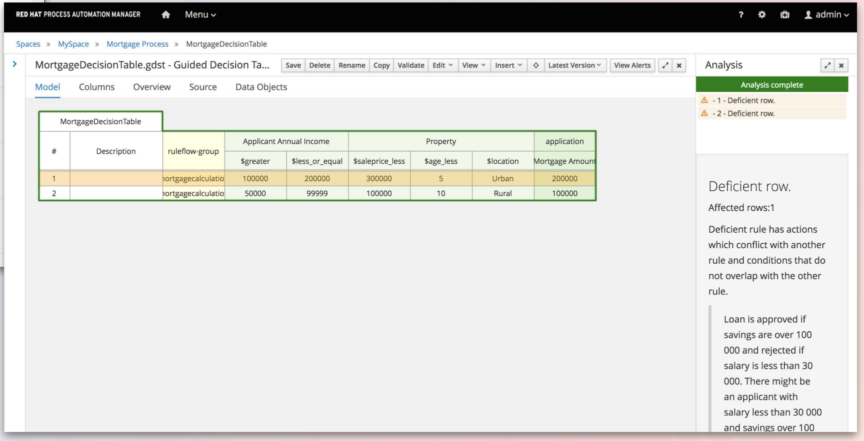
Track
View task inbox, process reports and task reports.

BUSINESS CENTRAL

DESIGN ENVIRONMENT



Course	Data Objects	Last modified today	Created today
CourseSchedule	Data Objects	Last modified today	Created today
CourseScheduleScoreHolderGlobal	Globals Definitions	Last modified today	Created today
Curriculum	Data Objects	Last modified today	Created today
CurriculumCourseSolverConfig	Solver configuration	Last modified today	Created today



MortgageDecisionTable

#	Description	ruleflow-group	Applicant Annual Income		Property			application
			$greater	$less_or_equal	$saleprice_less	$age_less	$location	Mortgage Amount
1		mortgagecalculatio	100000	200000	300000	5	Urban	200000
2		mortgagecalculatio	50000	99999	100000	10	Rural	100000

Deficient row.

Affected rows:1

Deficient rule has actions which conflict with another rule and conditions that do not overlap with the other rule.

> Loan is approved if savings are over 100 000 and rejected if salary is less than 30 000. There might be an applicant with salary less than 30 000 and savings over 100

BUSINESS CENTRAL REQUIREMENTS

CURRENT IN-HOUSE BUILD SYSTEM

The GOOD

- Builds Incrementally
- Multi-thread support
- Uses POM.xml as input
- Just works
    - In production for ~6y

The BAD

- Stateful
- Hard to scale out
- Hard to work on CI/CD
- Don’t respect Maven semantics
    - Problems from customers (RFE)

REQUEST:

REPLACE CURRENT BUILD WITH MAVEN

REQUIREMENTS GATHERING

INCREMENTAL COMPILATION

REQUIREMENT

Business Central user shouldn’t have to wait for a full compilation if the user has just changed a POJO or a Business Rule.

Most of Business Central rich editors provide some level of autocomplete that is based on the project content (Rules, Processes, POJOs, Configs, etc.).

On every user I/O action, it needs trigger an incremental build.

RESPECT/PRESERVE USER’S POM

REQUIREMENT

Business Central users shouldn’t require to change their POM’s if they already work outside Business central.

The system should be able to adapt on the fly User’s POM in order to make changed needed that result in a incremental build.

LOW LATENCY AND MEMORY FOOTPRINT

REQUIREMENT

Business Central is a web based tool, that has to support multiple concurrent users.

Maven is designed as stateless, however its bootstrap is quite heavy (multiple containers have to be initialized, like plexus/sisu/guice).

ADJUSTABLE FOR DIFFERENT USE CASES

REQUIREMENT

Business Central has different interactions that relies on the build system: Incremental for every user I/O operation, JUnit/Rules/Process execution, Packaging and Deployment.

Every use case, requires some configuration/tweak of Maven execution.

MULTI-THREAD

REQUIREMENT

Business Central should enable multiple users interact with projects, including collaboration in the same project.

Everyone that had to check a Maven output log of a build with a “-T 2” knows that logs are not isolated per thread.

Maven Embedder can’t even execute correctly more than 4 threads at same time (parameter parser get lost).

MULTI-USER/MULTI-M2REPO

REQUIREMENT

Business Central users may work in the same project in a isolated way (forked).

Avoid users step on other toes when consuming and producing artifacts (KJARs).

µService/Cloud Native

REQUIREMENT

The build system should be consumed by Business Central “as a Service” and should be developed as a cloud native component.

Builds could be executed co-located (within Business Central, suitable for incremental compilation) or executed remotely.

It’s expect that on heavy load the build system be auto-scaled by cloud infrastructure.

In & Off-Process

REQUIREMENT

Business Central should be able to keep working even if a malicious code is executed by a Business Rule or Process (ie. System.exit).

For efficient user interaction incremental compilation should be, as much as possible, be executed in the same JVM. However it’s necessary some isolation of JUnit/Rules/Process execution to avoid shut down the Application Server using System.exit.

RETURN RICH DATA STRUCTURES

REQUIREMENT

The kie-maven-plugin and kie-takari-plugin create some in-memory objects that are super heavy to create (basically all metadata information for all rules and process in a pre-compiled way).

Original Maven API basically returns only two values: FAIL or SUCCESS.

ASYNC API

REQUIREMENT

Compile, Packaging, JUnit Runs… or any Maven execution can take some time (like download the internet!), and we should not block Caller to wait for the completion of those heavy tasks.

Useful also for remote execution.

DETAILED REQUIREMENT LIST

The devil is in the details

- Incremental compilation
- Respect/Preserve User’s POM
- Low latency and memory footprint
- Adjustable for Different Use Cases
  - Compilation, JUnit Run, etc
- Multi-thread
- Multi-user
  - Multi M2 repos

- µService/Cloud Native
  - Local and Remote executions
- Embedded & Off-process mode
- Return rich data structures
  - Beyond Fail/Success
  - Collect data from PlugIn’s
- Async API

MAVEN AS A SERVICE FEATURES

THE SOLUTION

Provide an “enhanced compiler” with request-response behavior as simple as possible in its use and configuration and rich in terms of objects in the response compared to “plain” Maven

REQUEST

- Maven repo per request
- Project Path
- Maven CLI arguments
- Settings file
- Unique Compilation ID

```
interface CompilationRequest {

    AFCliRequest getKieCliRequest();

    WorkspaceCompilationInfo getInfo();

    String getMavenRepo();

    String[] getOriginalArgs();

    Map<String, Object> getMap();

    String getRequestUUID();

    Boolean skipAutoSourceUpdate();

    Boolean skipProjectDependenciesCreationList();

    Boolean getRestoreOverride();
}
```

RESPONSE

- Result of the build
- In Memory Log
- Dependency List
- Target folder content

```
interface CompilationResponse {

    Boolean isSuccessful();

    List<String> getMavenOutput();

    Optional<Path> getWorkingDir();

    List<String> getDependencies();

    List<URI> getDependenciesAsURI();

    List<URL> getDependenciesAsURL();

    List<String> getTargetContent();

    List<URI> getTargetContentAsURI();

    List<URL> getTargetContentAsURL();
}
```

KIE RESPONSE EXTENSION

- Drools live objects extracted from the Maven plugin
- Drools live objects generated on the fly (no .class file)
- Classloaders contents

```
interface KieCompilationResponse
          extends CompilationResponse {

    Optional<KieModuleMetaInfo> getKieModuleMetaInfo();

    Optional<KieModule> getKieModule();

    Map<String, byte[]> getProjectClassLoaderStore();

    Set<String> getEventTypeClasses();
}
```

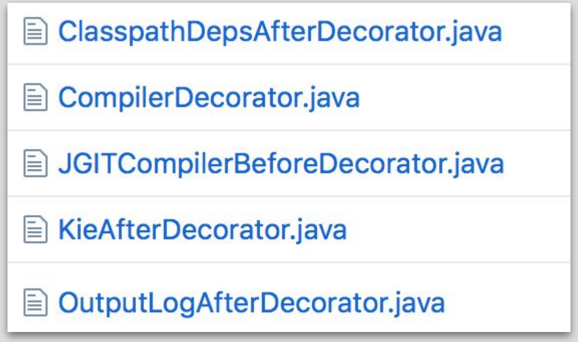
COMPILER

We add objects to the Maven result (a simple int) and we use a pipeline of decorators to add behaviours before and after compilation

```
interface AFCompiler<T extends CompilationResponse> {

    T compile(final CompilationRequest req);

    T compile(final CompilationRequest req,
                   final Map<Path, InputStream> override);

    Boolean cleanInternalCache();
}
```

PIPELINE: CHAIN OF DECORATORS

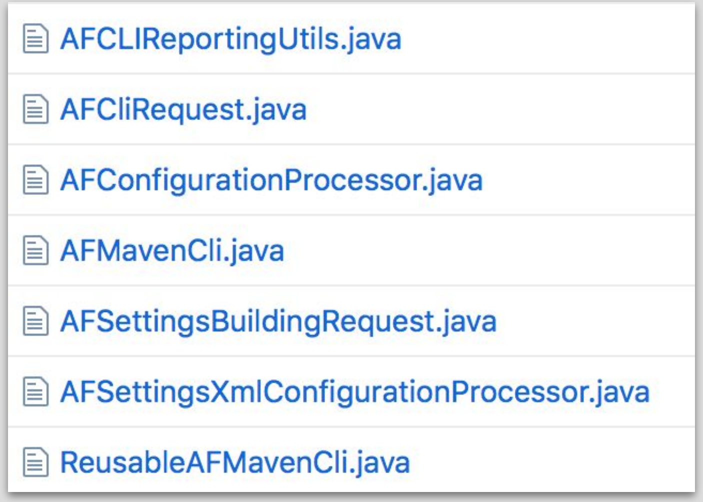
- Static factory for common use cases
- Configurable per object
- Possible to provide your own decorator



ClasspathDepsAfterDecorator.java
CompilerDecorator.java
JGITCompilerBeforeDecorator.java
KieAfterDecorator.java
OutputLogAfterDecorator.java

MAVEN API VERY RESTRICT

The first problem is how to open the "sealed" Maven API, we change the use of Plexus/Sisu we can reuse the same IoC container per project saving a huge amount of time, because the major part of the time in a Maven startup is the creation of this container


AFCLIReportingUtils.java
AFCliRequest.java
AFConfigurationProcessor.java
AFMavenCli.java
AFSettingsBuildingRequest.java
AFSettingsXmlConfigurationProcessor.java
ReusableAFMavenCli.java

CLASSLOADERS

Maven use classworlds to manage the various classloaders used for its tasks

- System Classloader
- Core Classloader
- Plugin Classloaders
- Custom Classloaders

https://maven.apache.org/guides/mini/guide-maven-classloading.html

CLASSLOADERS

If we want read an object inside the plugin and export it to the client code we have to cross the barriers of those classloaders.

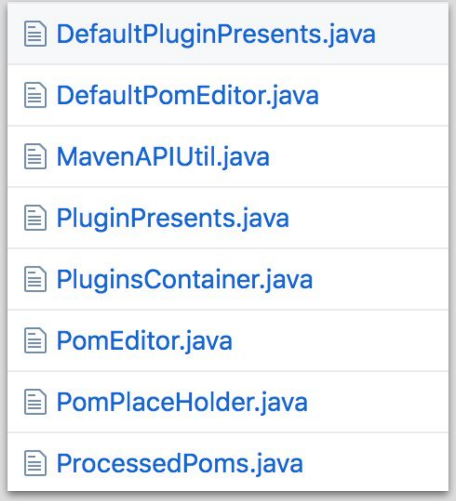
- ReusableAFMavenCli
- BuildMojo
- BaseMavenCompiler

IN MEMORY PLUGINS

A cool side effects when you cross the barriers of the classloaders is the option to turn the plugins impl from FS to in memory.

INCREMENTAL COMPILER

To spent less time in front of a Maven build we change on the fly the pom to add the Takari compiler in every module of the project tied with the compiler



DefaultPluginPresents.java
DefaultPomEditor.java
MavenAPIUtil.java
PluginPresents.java
PluginsContainer.java
PomEditor.java
PomPlaceHolder.java
ProcessedPoms.java

POM PROCESSOR

The Compiler is aware

of the plugins in the POM

It changes on the fly

the Pom turning off the Default Compiler

and adds Takari and the Kie Plugin

ASYNC API

```
interface AFCompilerService {

    CompletableFuture<KieCompilationResponse> build(Path projectPath, String mavenRepo);

    CompletableFuture<KieCompilationResponse> build(Path projectPath, String mavenRepo,
                                                    Map<Path, InputStream> override);

    CompletableFuture<KieCompilationResponse> build(Path projectPath, String mavenRepo,
                                                    Boolean skipPrjDependenciesCreationList);

    CompletableFuture<KieCompilationResponse> buildAndInstall(Path projectPath, String mavenRepo);

    CompletableFuture<KieCompilationResponse> buildAndInstall(Path projectPath, String mavenRepo,
                                                              Boolean skipPrjDependenciesCreationList);

    CompletableFuture<KieCompilationResponse> buildSpecialized(Path projectPath, String mavenRepo, String[] args);

    CompletableFuture<KieCompilationResponse> buildSpecialized(Path projectPath,
                                                               String mavenRepo,
                                                               String[] args,
                                                               Boolean skipPrjDependenciesCreationList);

}
```

COMPILER SERVICE

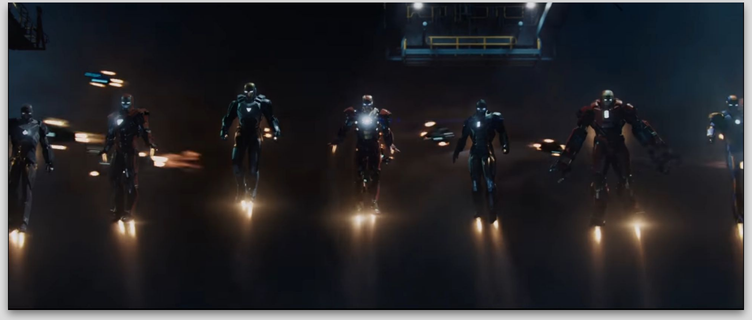
We could use the compiler core

to enable a compiler service

enabling local Maven executors

or remote Maven executors

on other nodes/pods/containers/VMs



COMPILER SERVICE

We could demand a build to a remote

node/pod/container/vm

and reads a remote File System

because

the core compiler

can use use a

JGITCompilerBeforeDecorator

LOCAL/REMOTE EXECUTORS

The compiler service need to know if the current “machine” owns enough resources to run a local executors or if is better to ask to a remote executor to satisfy the build.

To invoke a remote executors It needs to know the address to reach the chosen instance.

REST ENDPOINT

```
@Path("/build/maven/")
@RequestScoped
public class MavenRestHandler extends Application {

    @POST
    @Produces(MediaType.APPLICATION_OCTET_STREAM)
    public void postAsync(@Suspended AsyncResponse ar,
                          @HeaderParam("project") String projectRepo,
                          @HeaderParam("mavenrepo") String mavenRepo) throws Exception {
        CompletableFuture<KieCompilationResponse> response = compilerService.build(projectRepo, mavenRepo);
        response.whenCompleteAsync((kieCompilationResponse, throwable) -> {
            if (throwable != null) {
                logger.error(throwable.getMessage());
                ar.resume(Response.serverError().build());
            } else {
                byte[] bytes = RestUtils.serialize(new DefaultHttpCompilationResponse(kieCompilationResponse));
                ar.resume(Response.ok(bytes).build());
            }
        });
    }
}
```

Resources

https://github.com/desmax74

https://twitter.com/desmax74

https://www.slideshare.net/desmax74

https://github.com/porcelli

https://twitter.com/porcelli

https://pt.slideshare.net/alexandre_porcelli/

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos