



## Red Hat Process Automation Manager 7.5

Implementing high available event-driven  
decisioning using the decision engine on Red  
Hat OpenShift Container Platform



# Red Hat Process Automation Manager 7.5 Implementing high available event-driven decisioning using the decision engine on Red Hat OpenShift Container Platform

---

Red Hat Customer Content Services

[brms-docs@redhat.com](mailto:brms-docs@redhat.com)

## Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document describes how to implement high available event-driven decisioning using the decision engine on Red Hat OpenShift Container Platform in Red Hat Process Automation Manager 7.5.

---

## Table of Contents

PREFACE .....	3
CHAPTER 1. HIGH AVAILABLE EVENT-DRIVEN DECISIONING ON RED HAT OPENSIFT CONTAINER PLATFORM .....	4
CHAPTER 2. IMPLEMENTING THE HA CEP SERVER .....	5
CHAPTER 3. CREATING THE HA CEP CLIENT .....	7
CHAPTER 4. REQUIREMENTS FOR HA CEP CLIENT CODE .....	8
kie-remote API .....	8
Explicit timestamps .....	8
Lambda expressions for non-memory actions .....	8
APPENDIX A. VERSIONING INFORMATION .....	10



## PREFACE

As a business rules developer, you can use high available event-driven decisioning, including Complex Event Processing (CEP), in your code that uses the decision engine. You can implement high available event-driven decisioning on Red Hat OpenShift Container Platform.

You cannot use a standard deployment of Red Hat Process Automation Manager on Red Hat OpenShift Container Platform, as described in [Deploying a Red Hat Process Automation Manager environment on Red Hat OpenShift Container Platform using Operators](#), to implement high available event-driven decisioning, because the standard deployment supports only stateless processing. You must therefore create a custom implementation using the provided reference implementation.

### Prerequisites

- A Red Hat OpenShift Container Platform 4.1 environment is available and a project is created.
- A Kafka Cluster is deployed in the OpenShift environment with Red Hat AMQ Streams.
- The OpenJDK Java development environment is installed.
- Maven, Docker, and kubectl are installed.
- The **oc** OpenShift command line tool is installed.

# CHAPTER 1. HIGH AVAILABLE EVENT-DRIVEN DECISIONING ON RED HAT OPENSIFT CONTAINER PLATFORM

You can use the decision engine to implement high available event-driven decisioning on Red Hat OpenShift Container Platform.

An *event* models a fact that happened in a specific point in time. The decision engine offers a rich set of temporal operators to compare, correlate, and accumulate events. In event-driven decisioning, the decision engine processes complex series of decisions based on events. Every event can alter the state of the engine, influencing decisions for subsequent events.

You cannot use a standard deployment of Red Hat Process Automation Manager on Red Hat OpenShift Container Platform, as described in [Deploying a Red Hat Process Automation Manager environment on Red Hat OpenShift Container Platform using Operators](#), to run high available event-driven decisioning. The deployment includes Process Server (KIE Server) pods, which remain independent of each other when scaled. The states of the pods are not synchronized. Therefore, only stateless calls can be processed reliably.

The Complex Event Processing (CEP) API is useful for event-driven decisioning with the decision engine. The decision engine uses CEP to detect and process multiple events within a collection of events, to uncover relationships that exist between events, and to infer new data from the events and their relationships. For more information about CEP in the decision engine, see [Decision engine in Red Hat Process Automation Manager](#).

You can implement high available event-driven decisioning on Red Hat OpenShift Container Platform based on the reference implementation provided with Red Hat Process Automation Manager. This implementation provides an environment with safe failover.

In this reference implementation, you can scale the pod with the processing code. The replicas of the pod are not independent. One of the replicas is automatically designated *leader*. If the leader ceases to function, another replica is automatically made leader, and the processing continues without interruption or data loss.

The election of the leader is implemented with Kubernetes ConfigMaps. Coordination of the leader with other replicas is performed with exchanged messages through Kafka. The leader is always the first to process an event. When processing is complete, the leader notifies other replicas. A replica that is not the leader starts executing an event only after it has been completely processed on the leader.

When a new replica joins the cluster, this replica requests a snapshot of the current Drools session from the leader. The leader can use a recent existing snapshot if one is available in a Kafka topic. If a recent snapshot is not available, the leader produces a new snapshot on demand. After receiving the snapshot, the new replica deserializes it and eventually executes the last events not included in the snapshot before starting to process new events in coordination with the leader.



## CHAPTER 2. IMPLEMENTING THE HA CEP SERVER

The high-availability (HA) CEP server runs on the Red Hat OpenShift Container Platform environment. It includes all necessary Drools rules and other code required to process events.

You must prepare the source, build it, and then deploy it on Red Hat OpenShift Container Platform.

### Procedure

1. Download the **rhpam-7.5.1-reference-implementation.zip** product deliverable file from the [Software Downloads](#) page of the Red Hat Customer Portal.
2. Extract the contents of the file and then uncompress the **rhpam-7.5.1-openshift-drools-hacep-distribution.zip** file.
3. Change to the **openshift-drools-hacep-distribution/sources** directory.
4. Review and modify the server code based on the sample project in the **sample-hacep-project/sample-hacep-project-kjar** directory. The complex event processing logic is defined by the DRL rules in the **src/main/resources/org.drools.cep** subdirectory.
5. Build the project using the standard Maven command:

```
mvn clean install -DskipTests
```

6. Use the OpenShift operator infrastructure to install Red Hat AMQ Streams. For information about installing Red Hat AMQ Streams, see [Using AMQ Streams on OpenShift Container Platform](#).
7. Using the **KafkaTopic** resource on Red Hat OpenShift Container Platform, create the topics from all the YAML files in the **kafka-topics** subdirectory.  
For instructions about creating topics using the **KafkaTopic** resource, see [Using the topic operator](#) in the Red Hat AMQ documentation.
8. In order to enable application access to the ConfigMap that is used in the leader election, you must configure role-based access control. Change to the **springboot** directory and enter the following commands:

```
oc create -f kubernetes/service-account.yaml
oc create -f kubernetes/role.yaml
oc create -f kubernetes/role-binding.yaml
```

For more information about configuring role-based access control in Red Hat OpenShift Container Platform, see [Using RBAC to define and apply permissions](#) in the Red Hat OpenShift Container Platform product documentation.

9. In the **springboot** directory, edit the **Dockerfile** file. Replace the line that contains the **microdnf** command with the following line:

```
RUN microdnf install java-1.8.0-openjdk-headless && microdnf clean all
```



### NOTE

This workaround is required because of a [recently found issue with microdnf](#).

10. In the **springboot** directory, enter the following commands to build the Docker image and push it into the Docker registry that is configured on your system. (Consider configuring a private registry before running these commands). This build imports the built **sample-hacep-project-kjar** code as a Maven dependency and includes it in the **BOOT-INF/lib** directory of the **openshift-kie-springboot.jar** file. The Docker build then uses the JAR file to create the image.

```
docker login --username=<user username>
docker build -t <user_username>/openshift-kie-springboot:<tag> .
docker push <user_username>/openshift-kie-springboot:<tag>
```

Alternatively, to use Podman for the build process, replace **docker** with **podman** in the commands.

11. In the OpenShift user interface, create a YAML source. Paste in the contents of the **kubernetes/deployment.yaml** and change the Docker image name as necessary. Click **Deploy** to deploy the server.

## CHAPTER 3. CREATING THE HA CEP CLIENT

You must adapt your CEP client code to communicate with the HA CEP server image. You can use the sample project included in the reference implementation for your client code. You can run your client code inside or outside the OpenShift environment.

### Procedure

1. Download the **rhpm-7.5.1-reference-implementation.zip** product deliverable file from the [Software Downloads](#) page of the Red Hat Customer Portal.
2. Extract the contents of the file and then uncompress the **rhpm-7.5.1-openshift-drools-hacep-distribution.zip** file.
3. Change to the **openshift-drools-hacep-distribution/sources** directory.
4. Review and modify the client code based on the sample project in the **sample-hacep-project/sample-hacep-project-client** directory. Ensure that the code fulfills the additional requirements described in [Chapter 4, Requirements for HA CEP client code](#).
5. In the **sample-hacep-project/sample-hacep-project-client** directory, generate a keystore, using **password** as a password. Enter the following command:

```
keytool -genkeypair -keyalg RSA -keystore src/main/resources/keystore.jks
```

6. Extract the HTTPS certificate from the OpenShift environment and add it to the keystore. Enter the following commands:

```
oc extract secret/my-cluster-cluster-ca-cert --keys=ca.crt --to=- > src/main/resources/ca.crt
keytool -import -trustcacerts -alias root -file src/main/resources/ca.crt -keystore
src/main/resources/keystore.jks -storepass password -noprompt
```

7. In the **src/main/resources** subdirectory of the project, create a **configuration.properties** file with the following content:

```
ssl.keystore.location=keystore.jks
ssl.truststore.location=keystore.jks
ssl.keystore.password=password
ssl.truststore.password=password
bootstrap.servers=http://<serveraddress>
```

Replace **<serveraddress>** with the address that the route for the Kafka server uses.

8. Build the project using the standard Maven command:

```
mvn clean install
```

9. Change the **sample-hacep-project-client** project directory and enter the following command to run the client:

```
mvn exec:java -Dexec.mainClass="org.kie.hacep.sample.client.ClientProducerDemo
```

This command executes the **main** method of the **ClientProducerDemo** class.

## CHAPTER 4. REQUIREMENTS FOR HA CEP CLIENT CODE

When developing client code for high-availability CEP, you must follow certain additional requirements.

### kie-remote API

The code must use the **kie-remote** API and not the **kie** API. The **kie-remote** API is specified in the **org.kie:kie-remote** Maven artifact. You can find the source code in the **kie-remote** Maven module.

### Explicit timestamps

The decision engine needs to determine the sequence in which events happen. For this reason, every event must have an associated timestamp assigned to it. In a high-availability environment, make this timestamp a property of the JavaBean that models the event. You must then annotate the event class with the **@Timestamp** annotation, where the name of the timestamp attribute itself is the parameter, as in the following example:

```
@Role(Role.Type.EVENT)
@Timestamp("myTime")
public class StockTickEvent implements Serializable {

    private String company;
    private double price;
    private long myTime;
}
```

If you do not provide a timestamp attribute, Drools assigns a timestamp to every event based on the time when the event is inserted by the client into a remote session. However, this mechanism depends on the clocks on the client machines. If clocks between different clients diverge, inconsistencies can occur between events inserted by these hosts.

### Lambda expressions for non-memory actions

Working memory actions (actions to insert, modify, or delete information in the working memory of the decision engine) must be processed on every node of the cluster. Actions that are not memory actions must be executed only on the leader.

For example, the code might include the following rule:

```
rule FindAdult when
    $p : Person(age >= 18)
then
    modify($p) { setAdult(true) }; // working memory action
    sendEmailTo($p); // side effect
end
```

When this rule is triggered, the person must be marked as an adult on every node. However, only the leader must send the email, so that only one copy of the email is sent.

Therefore, in your code, you must wrap the email action (called a *side effect*) in a lambda expression, as shown in the following example:

```
rule FindAdult when
    $p : Person(age >= 18)
then
    modify($p) { setAdult(true) };
    DroolsExecutor.getInstance().execute( () -> sendEmailTo($p) );
end
```

-

## APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, March 01, 2021.