Red Hat Process Automation Manager 7.7

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

Last Updated: 2021-07-15

Red Hat Process Automation Manager 7.7 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

Legal Notice

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 7.7.

Table of Contents

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, 4.2, or 4.3 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 OPENSHIFT CONTAINER PLATFORM

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 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*.

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.

With the default implementation method, the service is built into the HA CEP server as a fat KJAR. In this case, build and deploy the server again to change the version of the service. The content of the working memory is lost when you switch to the new version. For instructions about the default implementation method, see Chapter 2, *Implementing the HA CEP server* .

If you require upgrading versions of the service without losing the content of the working memory, use an alternate implementation method and provide the KJAR and all dependencies in a Maven repository. In this implementation method, use an **UpdateKJarGAV** call from the client code to trigger deployment of a new KJAR version. This call is processed by the leader and then other replicas, and each of the pods then loads the new KJAR. The contents of the working memory remain in place. For instructions about this implementation method, see Chapter 3, *Implementing the HA CEP server with a Maven repository for updating the KJAR service*.

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.

Prepare the source, build it, and then deploy it on Red Hat OpenShift Container Platform.

Alternatively, use a different process to deploy the HA CEP server where you can update the KJAR service at any time. For instructions about this process, see Chapter 3, *Implementing the HA CEP server with a Maven repository for updating the KJAR service*.

Prerequisites

- You are logged into the project with administrator privilege using the **oc** command-line tool.

Procedure

1. Download the **rhpam-7.7.0-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-7.7.0-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. Enable the OpenShift operator for Red Hat AMQ Streams and then create an AMQ Streams (kafka) cluster in the project. For information about installing Red Hat AMQ Streams, see *Using AMQ Streams on OpenShift*.
7. To create the kafka topics that are required for operation of the server, remain in the **openshift-drools-hacep-distribution/sources** directory and run the following commands:

```
oc apply -f kafka-topics/control.yaml
oc apply -f kafka-topics/events.yaml
oc apply -f kafka-topics/kiesessioninfos.yaml
oc apply -f kafka-topics/snapshot.yaml
```

8. In order to enable application access to the ConfigMap that is used in the leader election, 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. Use the **oc** command to detect the valid values for the user ID and group ID that you must set in the Docker file. Enter the following command:

```
oc describe project <projectname>
```

In this command, replace **<projectname>** with the name of your project.

In the output of the command, find lines that list **supplemental-groups** and **uid-range**, similar to the following example:

```
openshift.io/sa.scc.supplemental-groups=1000160000/10000
openshift.io/sa.scc.uid-range=1000160000/10000
```

10. In the **springboot** directory, edit the **Dockerfile** file. Find the following line:

```
RUN groupadd -r app -g <id_group> && useradd -u <id_user> -r -g app -m -d /app -s
/sbin/nologin -c "App user" app && chmod 755 /app----
```

In this line, replace **<id_group>** with a value in the **supplemental-groups** interval and **<id_user>** with a value in the **uid-range** interval:

```
RUN groupadd -r app -g 1000160001 && useradd -u 1000160001 -r -g app -m -d /app -s
/sbin/nologin -c "App user" app && chmod 755 /app
```

11. In the **springboot** directory, enter the following commands to create the image for deployment and push it into the repository configured for your OpenShift environment:

```
oc new-build --binary --strategy=docker --name openshift-kie-springboot
oc start-build openshift-kie-springboot --from-dir=. --follow
```

12. Enter the following command to detect the name of the image that was built:

```
oc get is/openshift-kie-springboot -o template --template='{{range .status.tags}}{{range
.items}}{{.dockerImageReference}}{{end}}{{end}}'
```

13. Open the **kubernetes/deployment.yaml** file in a text editor.

14. Replace **<id_user>** with the user ID that you entered in the Docker file.

15. Replace the existing image URL with the result of the previous command.

16. Remove all characters at the end of the line starting with the @ symbol, then add **:latest** to the line. For example:

```
image: image-registry.openshift-image-registry.svc:5000/hacep/openshift-kie-
springboot:latest
```

17. Save the file.

18. Enter the following command to deploy the image:

```
oc apply -f kubernetes/deployment.yaml
```

CHAPTER 3. IMPLEMENTING THE HA CEP SERVER WITH A MAVEN REPOSITORY FOR UPDATING THE KJAR SERVICE

You can implement the HA CEP server that retrieves the KJAR service and all dependencies from a Maven repository that you provide. In this case, you can update the KJAR service at any time by updating it in the Maven repository and then making a call from the client code.

Prepare the source, build it, and then deploy it on Red Hat OpenShift Container Platform. Set certain environment variables in the **deployment.yaml** file before deploying the server. To use a Maven repository, you must set the **UPDATABLEKJAR** variable to **true**.

Prerequisites

- You are logged into the project with administrator privilege using the **oc** command-line tool.
- You configured a Maven repository that is accessible from your Red Hat OpenShift Container Platform environment.

Procedure

1. Download the **rhpam-7.7.0-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-7.7.0-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
   ```

   Upload the resulting KJAR and any required dependencies to the Maven repository.
6. Enable the OpenShift operator for Red Hat AMQ Streams and then create an AMQ Streams (kafka) cluster in the project. For information about installing Red Hat AMQ Streams, see *Using AMQ Streams on OpenShift*.
7. To create the kafka topics that are required for operation of the server, remain in the **openshift-drools-hacep-distribution/sources** directory and run the following commands:

   ```
   oc apply -f kafka-topics/control.yaml
   oc apply -f kafka-topics/events.yaml
   oc apply -f kafka-topics/kiesessioninfos.yaml
   oc apply -f kafka-topics/snapshot.yaml
   ```
8. In order to enable application access to the ConfigMap that is used in the leader election, 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. Use the **oc** command to detect the valid values for the user ID and group ID that you must set in the Docker file. Enter the following command:

```
oc describe project <projectname>
```

In this command, replace **<projectname>** with the name of your project.

In the output of the command, find lines that list **supplemental-groups** and **uid-range**, similar to the following example:

```
openshift.io/sa.scc.supplemental-groups=1000160000/10000
openshift.io/sa.scc.uid-range=1000160000/10000
```

10. In the **springboot** directory, edit the **pom.xml** file to remove the following dependency:

```
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>sample-hacep-project-kjar</artifactId>
</dependency>
```

11. In the **springboot** directory, edit the **Dockerfile** file. Find the following line:

```
RUN groupadd -r app -g <id_group> && useradd -u <id_user> -r -g app -m -d /app -s
/sbin/nologin -c "App user" app && chmod 755 /app----
```

In this line, replace **<id_group>** with a value in the **supplemental-groups** interval and **<id_user>** with a value in the **uid-range** interval:

```
RUN groupadd -r app -g 1000160001 && useradd -u 1000160001 -r -g app -m -d /app -s
/sbin/nologin -c "App user" app && chmod 755 /app
```

12. In the **springboot** directory, enter the following commands to create the image for deployment and push it into the repository configured for your OpenShift environment:

```
oc new-build --binary --strategy=docker --name openshift-kie-springboot
oc start-build openshift-kie-springboot --from-dir=. --follow
```

13. Enter the following command to detect the name of the image that was built:

```
oc get is/openshift-kie-springboot -o template --template='{{range .status.tags}}{{range
.items}}{{.dockerImageReference}}{{end}}{{end}}'
```

14. Open the **kubernetes/deployment.yaml** file in a text editor.

15. Replace **<id_user>** with the user ID that you entered in the Docker file.

16. Replace the existing image URL with the result of the previous command.
17. Remove all characters at the end of the line starting with the @ symbol, then add **:latest** to the line. For example:

```
image: image-registry.openshift-image-registry.svc:5000/hacep/openshift-kie-springboot:latest
```

18. Under the **containers:** line and the **env:** line, set environment variables as in the following example:

```
containers:
  - env:
    - name: UPDATABLEKJAR
      value: "true"
    - name: KJARGAV
      value: <GroupID>:<ArtifactID>:<Version>
    - name: MAVEN_LOCAL_REPO
      value: /app/.m2/repository
    - name: MAVEN_MIRROR_URL
      value: http://<nexus_url>/repository/maven-releases/
    - name: MAVEN_SETTINGS_XML
      value: /app/.m2/settings.xml
```

In this example, replace the value of the **KJARGAV** variable with the group, artifact, and version (GAV) of your KJAR service and the value of the **MAVEN_MIRROR_URL** variable with the URL to the Maven repository that contains your KJAR service.

Optionally, set other variables. For a list of supported environment variables, see Section 3.1, "Optional environment variables supported by the HA CEP server".

19. Save the file.
20. Enter the following command to deploy the image:

```
oc apply -f kubernetes/deployment.yaml
```

For instructions about triggering a KJAR update from the client code, see Chapter 4, *Creating the HA CEP client*.

3.1. OPTIONAL ENVIRONMENT VARIABLES SUPPORTED BY THE HA CEP SERVER

Optionally, set any of the following environment variables for a HA CEP server that is configured to use a Maven repository. Use the **deployment.yaml** file to set the variables at deployment time.

Table 3.1. Optional environment variables supported by the HA CEP server

Name	Description	Example
MAVEN_LOCAL_REPO	Directory to use as the local Maven repository.	**/root/.m2/repository**

Name	Description	Example
MAVEN_MIRROR_URL	The base URL of a Maven mirror that can be used for retrieving artifacts.	**http://nexus3-my-kafka-project.192.168.99.133.nip.io/repository/maven-public/**
MAVEN_MIRRORS	If set, multi-mirror support is enabled. The value contains a list of mirror prefixes, divided by commas. If this variable is set, the names of other **MAVEN_MIRROR_*** variables must contain a prefix, for example, **DEV_MAVEN_MIRROR_URL** and **QE_MAVEN_MIRROR_URL**	**DEV,QE**
MAVEN_REPOS	If set, multi-repo support is enabled. The value contains a list of repo prefixes, divided by commas. If this variable is set, the names of other **MAVEN_REPO_*** variables must contain a prefix, for example, **DEV_MAVEN_REPO_URL** and **QE_MAVEN_REPO_URL**.	**DEV,QE**
MAVEN_SETTINGS_XML	The location of a custom Maven **settings.xml** file to use	**/root/.m2/settings.xml**
***prefix*_MAVEN_MIRROR_ID**	Identifier to be used for the specified mirror. If omitted, a unique ID is generated.	**internal-mirror**
***prefix*_MAVEN_MIRROR_OF**	Repository IDs mirrored by this mirror. Defaults to **external:***	**external:*,!my-repo**
***prefix*_MAVEN_MIRROR_URL**	The URL of the mirror	**http://10.0.0.1:8080/repository/internal**
***prefix*_MAVEN_REPO_HOST**	Maven repository host name	**repo.example.com**
***prefix*_MAVEN_REPO_ID**	Maven repository ID	**my-repo**
***prefix*_MAVEN_REPO_LAYOUT**	Maven repository layout	**default**
***prefix*_MAVEN_REPO_USERNAME**	Maven repository username	**mavenUser**
***prefix*_MAVEN_REPO_PASSPHRASE**	Maven repository passphrase	**maven1!**

Name	Description	Example
***prefix*_MAVEN_REPO_PASSWORD**	Maven repository password	**maven1!**
***prefix*_MAVEN_REPO_PATH**	Maven repository path	**/maven2/**
***prefix*_MAVEN_REPO_PORT**	Maven repository port	**8080**
***prefix*_MAVEN_REPO_PRIVATE_KEY**	Local path to a private key for connecting to the Maven repository	**${user.home}/.ssh/id_dsa**
***prefix*_MAVEN_REPO_PROTOCOL**	Maven repository protocol	**http**
***prefix*_MAVEN_REPO_RELEASES_ENABLED**	Maven repository releases enabled	**true**
***prefix*_MAVEN_REPO_RELEASES_UPDATE_POLICY**	Maven repository releases update policy	**always**
***prefix*_MAVEN_REPO_SERVICE**	Maven repository OpenShift service. This value is used if a URL or host/port/protocol is not specified.	**buscentr-myapp**
***prefix*_MAVEN_REPO_SNAPSHOTS_ENABLED**	Maven repository snapshots enabled	**true**
***prefix*_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY**	Maven repository snapshots update policy	**always**
***prefix*_MAVEN_REPO_URL**	Fully qualified URL for the Maven repository	**http://repo.example.com:8080/maven2/**

CHAPTER 4. CREATING THE HA CEP CLIENT

You must adapt your CEP client code to communicate with the HA CEP server image. 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 **rhpam-7.7.0-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-7.7.0-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 5, *Requirements for HA CEP client and server code* .

5. To update the KJAR version in an implementation that uses the method described in Chapter 3, *Implementing the HA CEP server with a Maven repository for updating the KJAR service* , add an **UpdateKJarGAV** call to the client, similar to the following code:

```
        TopicsConfig envConfig = TopicsConfig.getDefaultTopicsConfig();
        Properties props = getProperties();
        try (RemoteStreamingKieSession producer =
RemoteStreamingKieSession.create(props, envConfig)){
            producer.updateKJarGAV("org.kie:fake-jar:0.1");
        }
```

   Ensure that a KJAR with the specified GAV is available in the Maven repository when this call is executed.

6. 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
```

7. 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
```

8. In the **src/main/resources** subdirectory of the project, open the **configuration.properties** file and replace **<bootstrap-hostname>** with the address that the route for the Kafka server provides.

9. Build the project using the standard Maven command:

```
mvn clean install
```

10. 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 5. REQUIREMENTS FOR HA CEP CLIENT AND SERVER CODE

When developing client and server code for high-availability CEP, follow certain additional requirements.

kie-remote API

The client 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. 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, 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 Friday, June 25, 2021.