



Red Hat Process Automation Manager 7.7

デシジョンエンジンを使用した高可用性イベント
駆動型デシジョン機能の Red Hat OpenShift
Container Platform への実装

ガイド

Red Hat Process Automation Manager 7.7 デシジョンエンジンを使用した 高可用性イベント駆動型デシジョン機能の Red Hat OpenShift Container Platform への実装

ガイド

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Implementing_high_available_event-driven_decisioning_using_the_decision_engine_on_Red_Hat_OpenShift_Container_Platform.ent file |.

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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.

概要

本書では、Red Hat Process Automation Manager 7.7 で、Red Hat OpenShift Container Platform に、デシジョンエンジンを使用して高可用性イベント駆動型デシジョン機能を実装する方法を説明します。

目次

はじめに	3
第1章 RED HAT OPENSIFT CONTAINER PLATFORM での高可用性イベント駆動型デシジョン機能	4
第2章 HA CEP サーバーの実装	5
第3章 MAVE リポジトリを使用した HA CEP サーバーを実装して KJAR サービスを更新する手順	8
3.1. HA CEP サーバーがサポートする環境変数 (オプション)	11
第4章 HA CEP クライアントの作成	14
第5章 HA CEP クライアントおよびサーバーコードの要件	16
kie-remote API	16
明示的なタイムスタンプ	16
メモリー以外のアクションの Lambda 式	16
付録A バージョン情報	18

はじめに

ビジネスルール開発者は、デシジョンエンジンを使用するコードで、複合イベント処理 (CAP: Complex Event Processing) など、高可用性イベント駆動型デシジョン機能を使用できます。高可用性イベント駆動型デシジョン機能は、Red Hat OpenShift Container Platform に実装できます。

『[Operatorを使用した Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 環境のデプロイメント](#)』の記載のとおり、Red Hat OpenShift Container Platform では、Red Hat Process Automation Manager の標準デプロイメントを使用して、高可用性イベント駆動型デシジョン機能を実装できません。標準デプロイメントは、ステータス処理しかサポートしないためです。そのため、指定の参照実装を使用して、カスタム実装を作成する必要があります。

前提条件

- Red Hat OpenShift Container Platform 4.1、4.2 または 4.3 環境が利用でき、プロジェクトが作成されている。
- Red Hat AMQ Streams を含む OpenShift 環境に、Kafka Cluster がデプロイされている。
- OpenJDK Java 開発環境がインストールされている。
- Maven、Docker、および kubectl がインストールされている。
- OpenShift コマンドラインツール **oc** がインストールされている。

第1章 RED HAT OPENSIFT CONTAINER PLATFORM での高可用性イベント駆動型デシジョン機能

デシジョンエンジンを使用して、Red Hat OpenShift Container Platform に高可用性イベント駆動型デシジョン機能を実装します。

イベントは、特定の時点で発生するファクトをモデル化します。デシジョンエンジンは、一時オペレーターが豊富にあり、イベントの比較、相関、累積ができます。イベント駆動型のデシジョン機能では、デシジョンエンジンがイベントをもとに一連の複雑なデシジョンを処理します。イベントはすべて、エンジンの状態を変更でき、後続のイベントのデシジョンに影響を与えます。

『[Operatorを使用した Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 環境のデプロイメント](#)』の記載のとおり、Red Hat OpenShift Container Platform では、Red Hat Process Automation Manager の標準デプロイメントを使用して、高可用性イベント駆動型デシジョン機能を実行できません。デプロイメントには、KIE Server Pod が含まれており、スケーリング時も Pod ごとに独立したままになります。Pod の状態は同期されません。そのため、ステートレス呼び出しのみを確実に処理できます。

複合イベント処理 (CEP) API は、デシジョンエンジンを含むイベント駆動型デシジョン機能で便利です。デシジョンエンジンは、CEP を使用してイベントコレクションにある複数のイベントを検出して処理し、イベント間に存在する関係を明確にして、このようなイベントや関係をもとに新規データを推測します。デシジョンエンジンでの CEP [に関する情報は、「Red Hat Process Automation Manager のデシジョンエンジン」](#)を参照してください。

Red Hat Process Automation Manager が提供する参照実装をもとに、Red Hat OpenShift Container Platform に高可用性イベント駆動型デシジョン機能を実装します。この実装を使用すると、安全にフェイルオーバーできる環境が実現できます。

この参照実装では、処理コードを使用して Pod をスケーリングできます。Pod のレプリカは独立していません。レプリカの1つが自動的にリーダーとして指定されます。リーダーが機能を停止した場合には、別のリーダーが自動的にリーダーになり、中断やデータの損失なしに、処理が続行されます。

リーダーの選択は、Kubernetes ConfigMaps で実装されます。リーダーと他のレプリカは、Kafka を介してメッセージを交換することで連携します。リーダーが必ず、最初にイベントを処理します。処理が完了したら、リーダーは他のレプリカに通知します。リーダーではないレプリカは、リーダーでの処理が完了してからでないとイベントは実行されません。

新規レプリカがクラスターに参加すると、このレプリカは、リーダーから、現在の Drools セッションのスナップショットを要求します。Kafka トピックで利用可能なスナップショットがある場合に、リーダーは既存で最新のスナップショットを使用できます。最新のスナップショットがない場合はリーダーがオンデマンドで新しいスナップショットを生成します。スナップショットを受信後に、新しいレプリカはそのスナップショットをデシリアライズし、最終的に、スナップショットに含まれていない最後のイベントを実行し、その後リーダーと連携して新規イベントの処理は開始されません。

デフォルトの実装方法では、このサービスは HA CEP サーバーに KJAR として組み込まれています。このような場合は、サーバーをもう一度ビルドしてデプロイし、サービスのバージョンを変更します。新規バージョンに切り替えると、作業メモリーの内容は失われます。デフォルトの実装方法に関する詳細は、[2章 HA CEP サーバーの実装](#)を参照してください。

作業メモリーの内容を失わずにサービスのバージョンをアップグレードする場合には、別の実装方法を使用して、KJAR と全依存関係を Maven リポジトリに用意します。この実装方法では、クライアントコードから `UpdateKJarGAV` 呼び出しを使用して、新規 KJAR バージョンのデプロイメントをトリガーします。この呼び出しは、リーダーと他のレプリカが処理し、各 Pod が新しい KJAR を読み込みます。作業メモリーの内容は、そのまま残ります。この実装方法に関する詳細は、[3章 Maven リポジトリを使用した HA CEP サーバーを実装して KJAR サービスを更新する手順](#)を参照してください。

第2章 HA CEP サーバーの実装

高可用性 (HA) CEP サーバーは、Red Hat OpenShift Container Platform 環境で実行します。このサーバーには、必要なすべての Drools ルールと、イベント処理に必要なその他のコードが含まれています。

ソースを準備して、ビルドし、Red Hat OpenShift Container Platform にデプロイします。

または、別のプロセスを使用して、いつでも KJAR サービスを更新できる HA CEP サーバーをデプロイします。このプロセスに関する詳細は、[3章Mave リポジトリを使用したHA CEP サーバーを実装してKJAR サービスを更新する手順](#)を参照してください。

前提条件

- **oc** コマンドラインツールを使用して、管理者権限があるプロジェクトにログインしている。

Procedure

1. Red Hat カスタマーポータル[の Software Downloads ページ](#)から **rhpam-7.7.0-reference-implementation.zip** 製品配信可能ファイルをダウンロードします。
2. ファイルの内容を展開してから、**rhpam-7.7.0-openshift-drools-hacep-distribution.zip** ファイルを展開します。
3. **openshift-drools-hacep-distribution/sources** ディレクトリーに移動します。
4. **sample-hacep-project/sample-hacep-project-kjar** ディレクトリー内のサンプルプロジェクトをもとに、サーバーのコードを確認して変更します。複合イベント処理のロジックは、**src/main/resources/org/drools/cep** サブディレクトリーの DRL ルールで定義します。
5. 標準の Maven コマンドを使用してプロジェクトをビルドします。

```
mvn clean install -DskipTests
```

6. Red Hat AMQ Streams 向けの OpenShift operator を有効にして、プロジェクトで AMQ Streams (kafka) クラスターを作成します。Red Hat AMQ Streams [のインストールに関する情報は、『AMQStreams on OpenShift の使用』](#)を参照してください。
7. サーバーの操作に必要な kafka のトピックを作成するには、**openshift-drools-hacep-distribution/sources** ディレクトリーで、以下のコマンドを実行します。

```
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. アプリケーションが、リーダーの選択に使用する ConfigMap にアクセスできるように、ロールベースのアクセス制御を設定します。**springboot** ディレクトリーに移動して、以下のコマンドを入力します。

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

Red Hat OpenShift Container Platform のロールベースのアクセス制御に関する詳細は、Red Hat OpenShift Container Platform [ドキュメントの「RBAC を使用したパーミッションの定義および適用」](#) を参照してください。

9. **oc** コマンドを使用して、ユーザー ID およびグループ ID の有効な値（Docker ファイルに設定する必要あり）を検出します。以下のコマンドを実行します。

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

このコマンドで、**<projectname>** をプロジェクトの名前に置き換えます。

以下の例のように、**supplemental-groups** と **uid-range** が表示されている行を検索します。

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

10. **springboot** ディレクトリーで、**Dockerfile** ファイルを編集します。以下の行を見つけます。

```
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----
```

この行で、**<id_group>** を、**supplemental-groups** の間隔の値に、**<id_user>** は **uid-range** の間隔の値に置き換えます。

```
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. **springboot** ディレクトリーで、以下のコマンドを実行してデプロイメント用のイメージを作成し、OpenShift 環境用に設定したリポジトリーにプッシュします。

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

12. 以下のコマンドを実行して、ビルドしたイメージの名前を検出します。

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

13. テキストエディターで **kubernetes/deployment.yaml** ファイルを開きます。

14. **<id_user>** を、**Docker** ファイルに入力したユーザー ID に置き換えます。

15. 既存のイメージ URL を直前のコマンドの結果に置き換えます。

16. 文頭に @ の記号がある行の最後にある文字をすべて削除し、その行に :latest を追加します。以下に例を示します。

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

17. ファイルを保存します。

18. 以下のコマンドを実行してイメージをデプロイします。

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

第3章 MAVEN リポジトリを使用した HA CEP サーバーを実装して KJAR サービスを更新する手順

HA CEP サーバーを実装して、KJAR サービスと全依存関係を指定の Maven リポジトリから取得できます。このような場合、Maven リポジトリでサービスを更新して、クライアントコードから呼び出しを行うことで、いつでも KJAR サービスを更新できます。

ソースを準備して、ビルドし、Red Hat OpenShift Container Platform にデプロイします。サーバーをデプロイする前に、`deployment.yaml` ファイルに特定の環境変数を設定します。Maven リポジトリを使用するには、`UPDATABLEKJAR` 変数を `true` に設定する必要があります。

前提条件

- `oc` コマンドラインツールを使用して、管理者権限があるプロジェクトにログインしている。
- Red Hat OpenShift Container Platform 環境からアクセス可能な Maven リポジトリを設定している。

Procedure

1. Red Hat カスタマーポータル [Software Downloads](#) ページから `rhpm-7.7.0-reference-implementation.zip` 製品配信可能ファイルをダウンロードします。
2. ファイルの内容を展開してから、`rhpm-7.7.0-openshift-drools-hacep-distribution.zip` ファイルを展開します。
3. `openshift-drools-hacep-distribution/sources` ディレクトリに移動します。
4. `sample-hacep-project/sample-hacep-project-kjar` ディレクトリ内のサンプルプロジェクトをもとに、サーバーのコードを確認して変更します。複合イベント処理のロジックは、`src/main/resources/org/drools/cep` サブディレクトリの DRL ルールで定義します。
5. 標準の Maven コマンドを使用してプロジェクトをビルドします。

```
mvn clean install -DskipTests
```

作成した KJAR と必要な依存関係を Maven リポジトリにアップロードします。

6. Red Hat AMQ Streams 向けの OpenShift operator を有効にして、プロジェクトで AMQ Streams (kafka) クラスターを作成します。Red Hat AMQ Streams のインストールに関する情報は、『[AMQStreams on OpenShift の使用](#)』を参照してください。

7. サーバーの操作に必要な kafka のトピックを作成するには、`openshift-drools-hacp-distribution/sources` ディレクトリで、以下のコマンドを実行します。

```
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. アプリケーションが、リーダーの選択に使用する ConfigMap にアクセスできるように、ロールベースのアクセス制御を設定します。springboot ディレクトリに移動して、以下のコマンドを入力します。

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

Red Hat OpenShift Container Platform のロールベースのアクセス制御に関する詳細は、Red Hat OpenShift Container Platform ドキュメントの「[RBAC を使用したパーミッションの定義および適用](#)」を参照してください。

9. `oc` コマンドを使用して、ユーザー ID およびグループ ID の有効な値 (Docker ファイルに設定する必要あり) を検出します。以下のコマンドを実行します。

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

このコマンドで、`<projectname>` をプロジェクトの名前に置き換えます。

以下の例のように、`supplemental-groups` と `uid-range` が表示されている行を検索します。

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

10. **springboot** ディレクトリーで **pom.xml** ファイルを編集して、以下の依存関係を削除します。

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

11. **springboot** ディレクトリーで、**Dockerfile** ファイルを編集します。以下の行を見つけます。

```
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----
```

この行で、**<id_group>** を、**supplemental-groups** の間隔の値に、**<id_user>** は **uid-range** の間隔の値に置き換えます。

```
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. **springboot** ディレクトリーで、以下のコマンドを実行してデプロイメント用のイメージを作成し、**OpenShift** 環境用に設定したリポジトリーにプッシュします。

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

13. 以下のコマンドを実行して、ビルドしたイメージの名前を検出します。

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

14. テキストエディターで **kubernetes/deployment.yaml** ファイルを開きます。

15. **<id_user>** を、**Docker** ファイルに入力したユーザー ID に置き換えます。

16. 既存のイメージ URL を直前のコマンドの結果に置き換えます。

- 17.

文頭に @ の記号がある行の最後にある文字をすべて削除し、その行に :latest を追加します。以下に例を示します。

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

18.

containers: 行と **env:** 行の下で、以下の例のように環境変数を設定します。

```
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
```

この例では KJARGAV 変数は、KJAR サービスのグループ、アーティファクト、バージョン (GAV) に置き換え、MAVEN_MIRROR_URL 変数の値は、KJAR サービスを含む Maven リポジトリの URL に置き換えます。

必要に応じて他の変数を設定します。サポート対象の環境変数の一覧は、「[HA CEP サーバーがサポートする環境変数 \(オプション\)](#)」を参照してください。

19.

ファイルを保存します。

20.

以下のコマンドを実行してイメージをデプロイします。

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

クライアントコードから KJAR の更新をトリガーする方法は、[4章 HA CEP クライアントの作成](#) を参照してください。

3.1. HA CEP サーバーがサポートする環境変数 (オプション)

オプションで、HA CEP サーバーの以下の環境変数を設定して、Maven リポジトリを使用するよ

うに設定します。 `deployment.yaml` ファイルを使用して、デプロイメント時に変数を設定します。

表3.1 HA CEP サーバーがサポートする環境変数 (オプション)

名前	説明	例
MAVEN_LOCAL_REPO	ローカルの Maven リポジトリとして使用するディレクトリ。	<code>/root/.m2/repository</code>
MAVEN_MIRROR_URL	アーティファクトの取得に使用可能な Maven ミラーのベース URL。	<code>http://nexus3-my-kafka-project.192.168.99.133.nip.io/repository/maven-public/</code>
MAVEN_MIRRORS	設定すると、マルチミラーサポートが有効になります。この値には、コンマで区切れたミラーのプリフィックス一覧が含まれます。この変数を設定した場合には、他の MAVEN_MIRROR_* 変数の名前に DEV_MAVEN_MIRROR_URL や QE_MAVEN_MIRROR_URL などのプリフィックスを含める必要があります。	DEV,QE
MAVEN_REPOS	設定すると、マルチリポジトリサポートが有効になります。この値には、コンマで区切れたリポジトリのプリフィックス一覧が含まれます。この変数を設定した場合には、他の MAVEN_REPO_* 変数の名前に DEV_MAVEN_REPO_URL や QE_MAVEN_REPO_URL などのプリフィックスを含める必要があります。	DEV,QE
MAVEN_SETTINGS_XML	使用するカスタムの Maven ファイル <code>settings.xml</code> の場所。	<code>/root/.m2/settings.xml</code>
prefix_MAVEN_MIRROR_ID	指定のミラーに使用する ID。省略する場合には、一意の ID が生成されます。	internal-mirror
prefix_MAVEN_MIRROR_OF	このミラーでミラリングされるリポジトリ ID。デフォルト値は external:* です。	external:*;!my-repo
prefix_MAVEN_MIRROR_URL	ミラーの URL。	<code>http://10.0.0.1:8080/repository/internal</code>
prefix_MAVEN_REPO_HOST	Maven リポジトリのホスト名。	repo.example.com
prefix_MAVEN_REPO_ID	Maven リポジトリ ID。	my-repo

名前	説明	例
<code>prefix_MAVEN_REPO_LAYOUT</code>	Maven リポジトリのレイアウト。	default
<code>prefix_MAVEN_REPO_USERNAME</code>	Maven リポジトリのユーザー名。	mavenUser
<code>prefix_MAVEN_REPO_PASSPHRASE</code>	Maven リポジトリのパスフレーズ。	maven1!
<code>prefix_MAVEN_REPO_PASSWORD</code>	Maven リポジトリのパスワード。	maven1!
<code>prefix_MAVEN_REPO_PATH</code>	Maven リポジトリのパス。	/maven2/
<code>prefix_MAVEN_REPO_PORT</code>	Maven リポジトリのポート。	8080
<code>prefix_MAVEN_REPO_PRIVATE_KEY</code>	Maven リポジトリに接続するのに使用する秘密鍵へのローカルパス。	\$(user.home)/.ssh/id_dsa
<code>prefix_MAVEN_REPO_PROTOCOL</code>	Maven リポジトリのプロトコル。	http
<code>prefix_MAVEN_REPO_RELEASES_ENABLED</code>	Maven リポジトリのリリースが有効。	true
<code>prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY</code>	Maven リポジトリリリース更新ポリシー。	always
<code>prefix_MAVEN_REPO_SERVICE</code>	Maven リポジトリの OpenShift サービス。この値は、URL または host/port/protocol が指定されていない場合に使用します。	buscentr-myapp
<code>prefix_MAVEN_REPO_SNAPSHOTS_ENABLED</code>	Maven リポジトリのスナップショットが有効。	true
<code>prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY</code>	Maven リポジトリスナップショット更新ポリシー。	always
<code>prefix_MAVEN_REPO_URL</code>	Maven リポジトリの完全修飾 URL	http://repo.example.com:8080/maven2/

第4章 HA CEP クライアントの作成

CEP クライアントコードを HA CEP サーバーイメージと通信できるように、適応する必要があります。お使いのクライアントコード向けの参照実装に含まれるサンプルプロジェクトを使用してください。また、OpenShift 環境内外を問わず、クライアントコードを実行できます。

Procedure

1. Red Hat カスタマーポータル [の Software Downloads ページ](#) から `rhpmam-7.7.0-reference-implementation.zip` 製品配信可能ファイルをダウンロードします。
2. ファイルの内容を展開してから、`rhpmam-7.7.0-openshift-drools-hacep-distribution.zip` ファイルを展開します。
3. `openshift-drools-hacep-distribution/sources` ディレクトリーに移動します。
4. `sample-hacep-project/sample-hacep-project-client` ディレクトリーのサンプルプロジェクトをもとにクライアントコードをレビューし、変更します。このコードが [5章 HA CEP クライアントおよびサーバーコードの要件](#) に記載の追加要件を満たしていることを確認します。
5. [3章 Maven リポジトリーを使用した HA CEP サーバーを実装して KJAR サービスを更新する手順](#) で説明した方法を使用する実装で KJAR バージョンを更新するには、以下のコードのように、`UpdateKJarGAV` 呼び出しをクライアントに追加します。

```

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

```

この呼び出しの実行時に、GAV を指定した KJAR が Maven リポジトリーで利用できるようにしてください。

6. `sample-hacep-project/sample-hacep-project-client` ディレクトリーで、パスワードに `password` と指定してキーストアを生成します。以下のコマンドを実行します。

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

7.

OpenShift 環境から HTTPS 証明書を展開して、キーストアーに追加します。以下のコマンドを実行します。

```
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.

プロジェクトの **src/main/resources** サブディレクトリーで、**configuration.properties** ファイルを開き、**<bootstrap-hostname>** を Kafka サーバーのルートが提供するアドレスに置き換えます。

9.

標準の Maven コマンドを使用してプロジェクトをビルドします。

```
mvn clean install
```

10.

sample-hacep-project-client プロジェクトのディレクトリーに移動して、以下のコマンドを入力し、クライアントを実行します。

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

このコマンドは、**ClientProducerDemo** クラスの **main** メソッドを実行します。

第5章 HA CEP クライアントおよびサーバーコードの要件

高可用性 CEP のクライアントおよびサーバーコードを開発する場合には、以下のような特定の追加要件に準拠します。

kie-remote API

クライアントコードは、kie API ではなく kie-remote API を使用する必要があります。kie-remote API は、`org.kie:kie-remote` Maven アーティファクトに指定します。また、ソースコードは、Maven モジュール `kie-remote` にあります。

明示的なタイムスタンプ

デシジョンエンジンは、イベントの発生する順番を決定する必要があります。このような理由から、イベントには必ず、タイムスタンプを割り当てます。高可用性環境では、イベントをモデル化する `JavaBean` のプロパティーに、このタイムスタンプを指定します。次に、イベントクラスに `@Timestamp` アノテーションをつける必要があります。以下の例のように、ここではタイムスタンプ属性自体の名前がパラメーターとなります。

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

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

タイムスタンプ属性を指定しない場合には、クライアントがリモートセッションにイベントを挿入するタイミングをもとに、`Drools` が全イベントにタイムスタンプを割り当てます。ただし、このメカニズムは、クライアントマシンのクロックにより異なります。異なるクライアント間でクロックにずれがある場合は、このようなホストが挿入したイベント間で不整合が発生する可能性があります。

メモリー以外のアクションの Lambda 式

作業メモリーアクション (デシジョンエンジンの作業メモリー内の情報を挿入、変更、または削除するアクション) は、クラスターの全ノードで処理する必要があります。メモリーアクションではないアクションは、リーダーでのみ実行する必要があります。

たとえば、今回のコードには、以下のルールが含まれます。

```
rule FindAdult when
    $p : Person(age >= 18)
then
```

```
modify($p) { setAdult(true) }; // working memory action
sendEmailTo($p); // side effect
end
```

このルールがトリガーされると、対象となる人は、すべてのノードで大人としてマークする必要があります。ただし、送信されるメール数が1通だけとなるように、リーダーだけがメールを送信できます。

そのため、以下の例のように、lambda 式のメールアクション (副作用 と呼ばれる) をラップします。

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

付録A バージョン情報

本書の最終更新日：2021年6月25日（金）