Red Hat Fuse 7.9

JBoss EAP에 배포

JBoss EAP(Enterprise Application Platform) 컨테이너에 애플리케이션 패키지 배포

Last Updated: 2023-07-25

Red Hat Fuse 7.9 JBoss EAP에 배포

JBoss EAP(Enterprise Application Platform) 컨테이너에 애플리케이션 패키지 배포

법적 공지

초록

이 가이드에서는 JBoss EAP 컨테이너에 애플리케이션을 배포하는 옵션에 대해 설명합니다.

차례

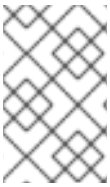
보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 에서 참조하십시오.

1장. JBOSS FUSE ON JBOSS EAP DEPLOYMENT 개요

JBoss EAP 컨테이너에 Fuse on EAP 패키지를 설치한 후 Red Hat JBoss EAP(JBoss Enterprise Application Platform)에 Fuse 애플리케이션을 배포할 수 있습니다.

이 부분에서는 EAP 하위 시스템에서 Camel을 사용하는 배포 모델을 설명합니다. Apache Camel in Fuse를 사용하면 통합된 애플리케이션을 실행할 컨테이너를 선택할 수 있습니다.



참고

Red Hat JBoss EAP에는 관리자와 개발자를 모두 충족하는 다양한 애플리케이션 배포 및 구성 옵션이 있습니다. JBOSS EAP 구성 및 배포 프로세스에 대한 자세한 내용은 Red Hat JBoss EAP 구성 가이드를 참조하십시오.

1.1. 지원되는 제품 버전

Fuse 7.9를 지원하는 JBoss EAP의 최신 버전을 보려면 지원되는 구성 페이지를 참조하십시오.

1.2. CAMEL ON EAP CRYOSTAT

EAP 하위 시스템의 Camel은 Apache Camel을 JBoss EAP 컨테이너에 직접 통합합니다. 이 하위 시스템은 EAP 패키지에 Fuse를 JBoss EAP 컨테이너에 설치한 후 사용할 수 있습니다. Camel 구성 요소를 단순화하고 기본 JBoss EAP 컨테이너와의 긴밀한 통합을 포함하여 Camel 배포에 많은 이점을 제공합니다.

Red Hat은 JBoss EAP에 Apache Camel 애플리케이션을 배포하기 위해 EAP의 Camel 배포 모델을 사용하는 것이 좋습니다.

2장. JBOSS EAP에서 애플리케이션 빌드

2.1. 개요

다음 예제에서는 Red Hat Fuse on EAP에서 **camel-cdi** 구성 요소를 사용하여 CDI 빈을 Camel 경로와 통합하는 방법을 보여줍니다.

이 예에서 Camel 경로는 서블릿 **HTTP GET** 요청에서 메시지 페이로드를 가져와 직접 엔드포인트로 전달합니다. 그런 다음 페이로드를 Camel CDI 8080 호출에 전달하여 메시지 응답을 생성하고 웹 브라우저 페이지에 출력을 표시합니다.

2.2. 프로젝트 실행

프로젝트를 실행하기 전에 설정에 maven과 Red Hat Fuse를 사용한 애플리케이션 서버가 포함되어 있는지 확인합니다. 프로젝트를 실행하려면 다음 단계를 수행합니다.

1. 독립 실행형 모드에서 애플리케이션 서버를 시작합니다.
   - Linux의 경우: **${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml**
   - Windows의 경우: **%JBOSS_HOME%\bin\standalone.extension -c standalone-full.xml**
2. 프로젝트를 빌드하고 배포합니다. **mvn install -Pdeploy**
3. 이제 **http://localhost:8080/example-camel-cdi/?name=World** 위치를 찾습니다. 다음 메시지 **Hello World from 127.0.0.1** 은 웹 페이지에 출력으로 표시됩니다. 또한 다음과 같이 **MyRouteBuilder.java** 클래스에서 Camel 경로를 볼 수 있습니다.

```
from("direct:start").bean("helloBean");
```

Quarkus DSL을 통해 Camel은 8080 레지스트리에서 **helloBean** 이라는 빈을 찾습니다. **또한 빈 클래스는 SomeBean 클래스로 인해 Camel에서 사용할 수 있습니다. @Named 주석을 사용하여 camel-cdi 는 Quarkus를 Camel 8080 레지스트리에 추가합니다.**

```
@Named("helloBean")

public class SomeBean {

    public String someMethod(String name) throws Exception {

        return String.format("Hello %s from %s", name, InetAddress.getLocalHost().getHostAddress());

    }

}
```

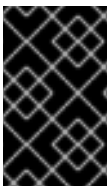
자세한 내용은 $ EAP_HOME/quickstarts/camel/camel-cdi 디렉터리를 참조하십시오.

2.3. JBOSS EAP용 BOM 파일

BOM(Maven bill of materials) 파일의 목적은 잘 작동하는 Maven 종속성 버전 집합을 제공하여 모든 Maven 아티팩트에 대해 버전을 개별적으로 정의할 필요가 없도록 하는 것입니다.

JBoss EAP용 Fuse BOM은 다음과 같은 이점을 제공합니다.

- POM에 종속성을 추가할 때 버전을 지정할 필요가 없도록 Maven 종속 항목에 대한 버전을 정의합니다.
- 특정 버전의 Fuse에서 완전히 테스트 및 지원되는 선별된 종속성 세트를 정의합니다.
- Fuse 업그레이드 간소화.



중요

Fuse BOM에서 정의한 종속성 세트만 Red Hat에서 지원합니다.

BOM 파일을 Maven 프로젝트에 통합하려면 다음 예와 같이 프로젝트의 pom.xml 파일(또는 상위 POM 파일)에 dependencyManagement 요소를 지정합니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.9.0.fuse-sb2-790065-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-eap-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
```

```
  </dependencyManagement>
  ...
</project>
```

종속성 관리 메커니즘을 사용하여 **BOM**을 지정하면 아티팩트 버전을 지정하지 *않고* **Maven** 종속성을 **POM**에 추가할 수 있습니다. 예를 들어 **camel-velocity** 구성 요소에 대한 종속성을 추가하려면 **POM**의 종속성 요소에 다음 **XML** 조각을 추가합니다.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <scope>provided</scope>
</dependency>
```

이 종속성 정의에서 **version** 요소를 생략하는 방법을 참조하십시오.

3장. 기능

이 장에서는 EAP 기능에 대한 Camel에 대한 필요한 정보를 제공합니다.

Camel 컨텍스트 정의

Camel 컨텍스트는 독립 실행형-camel.xml 및 domain.xml에서 다음과 같은 하위 시스템 정의의 일부로 구성할 수 있습니다.

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel 컨텍스트 배포

다음과 같이 -camel-context.xml 접미사를 사용하여 JBoss EAP에 camel 컨텍스트를 배포할 수 있습니다.

- 독립 실행형 XML 파일
- 다른 지원되는 배포의 일부

배포에는 여러 -camel-context.xml 파일이 포함될 수 있습니다.

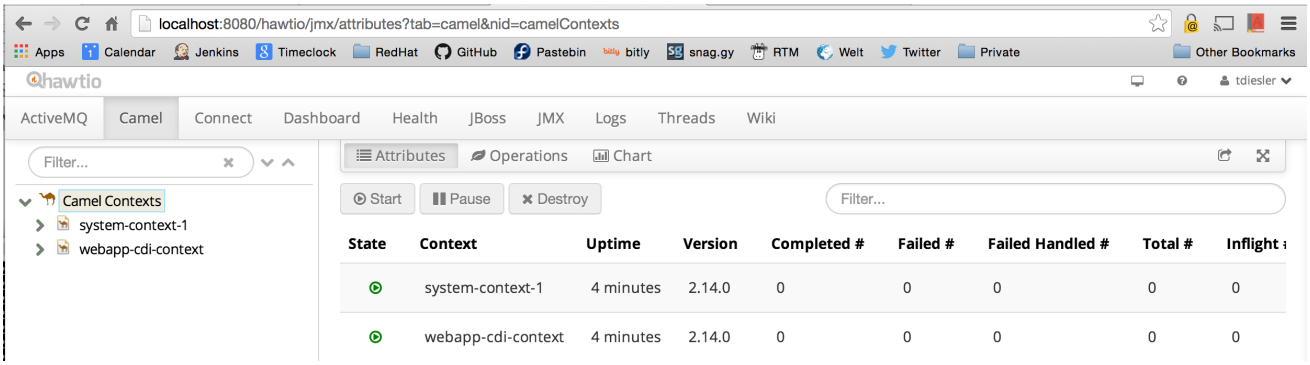
배포된 Camel 컨텍스트는 다음과 같이 CDI 삽입 가능

```
@Resource(lookup = "java:jboss/camel/context/mycontext")
CamelContext camelContext;
[discrete]
### Management Console
```

기본적으로 관리 콘솔에 대한 액세스는 보호됩니다. 따라서 먼저 관리 사용자를 설정해야 합니다.

```
$ bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
```

Hawt.io 콘솔에 하위 시스템 구성의 camel 컨텍스트가 표시되어야 합니다.



Arquillian 테스트 지원

EAP 테스트 모음의 Camel은 WildFly Arquillian 관리 컨테이너를 사용합니다. 이 명령은 이미 실행 중인 JBoss EAP 인스턴스에 연결하거나 필요한 경우 독립 실행형 서버 인스턴스를 시작할 수 있습니다.

Arquillian 테스트 사례에 삽입된 EAP 특정 유형에 이러한 Camel을 사용할 수 있는 다수의 테스트 강화가 구현되었습니다.

```
@ArquillianResource
CamelContextFactory contextFactory;

@ArquillianResource
CamelContextRegistry contextRegistry;
```

4장. 설정

이 장에서는 Camel Cryostat 및 배포 구성에 대한 필요한 정보를 제공합니다.

Camel Cryostat 구성

Camel Cryostat 구성에는 정적 시스템 경로가 포함될 수 있습니다. 그러나 시스템은 경로를 자동으로 시작합니다.

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel 배포 구성

Camel 배포의 기본 구성을 수정하려면 배포에서 WEB-INF/jboss-all.xml 또는 META-INF/jboss-all.xml 구성 파일을 편집할 수 있습니다.

jboss-all.xml 파일 내에서 <jboss-camel > XML 요소를 사용하여 camel 구성을 제어합니다.

Camel Cryostat 비활성화

배포에 camel 하위 시스템을 추가하지 않으려면 jboss-camel XML 요소에 enabled="false" 속성을 설정합니다.

jboss-all.xml 파일의 예:

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0" enabled="false"/>
</jboss>
```

구성 요소 선택

중첩된 < component > 또는 < component-module > XML 요소를 추가하면 배포에 기본 Camel 구성 요소 목록을 추가하는 대신 지정된 구성 요소만 배포에 추가됩니다.

jboss-all.xml 파일의 예:

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0">
    <component name="camel-ftp"/>
    <component-module name="org.apache.camel.component.rss"/>
  </jboss-camel>
</jboss>
```

5장. JARKARTA EE INTEGRATION

이 장에서는 Jarkarta EE와의 통합 지점에 대한 필요한 정보를 제공합니다.

5.1. CDI

Camel CDI 구성 요소는 CDI를 종속성 주입 프레임워크로 사용하여 Apache Camel에 대한 자동 구성을 제공합니다. 그러나 이는 규칙 초과 구성을 기반으로 합니다. CDI 빈에서 Camel 주석을 쉽게 사용할 수 있도록 표준 camel 8080 통합을 구현합니다.

CDI에 대한 자세한 내용은 cdi 문서를 참조하십시오.

다음 예제에서는 경로를 사용하여 Camel 컨텍스트를 사용하고 추정할 수 있는 방법을 설명합니다.

```
@Startup
@ApplicationScoped
@ContextName("cdi-context")
public class MyRouteBuilder extends RouteBuilder {

    @Override
    public void configure() throws Exception {
     from("direct:start").transform(body().prepend("Hi"));
    }
}
```

```
@Inject
@ContextName("cdi-context")
private CamelContext camelctx;
```

5.1.1. XML DSL 구성 가져오기

Camel CDI 통합을 사용하면 @ImportResource 주석을 통해 기존 XML DSL 파일을 가져올 수 있습니다.

```
@ImportResource("camel-context.xml")
class MyBean {
}
```

참고

가져온 파일의 위치는 배포 클래스 경로에 있어야 합니다. **WEB-INF** 와 같은 위치에 파일을 배치하는 것은 작동하지 않습니다. 그러나 **article -INF/classes** 는 잘 작동합니다.

5.2. EJB

관리 지원은 **migration3** 하위 시스템과 통합된 **Cryostat** 구성 요소를 통해 제공됩니다.

```
CamelContext camelctx = new DefaultCamelContext();
    camelctx.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from("direct:start").to("ejb:java:module/HelloBean");
        }
    });
```

5.3. JAXB

CryostatB 지원은 **Camel CryostatB** 데이터 형식을 통해 제공됩니다.

Camel은 **unmarshalling XML** 데이터를 **CryostatB** 주석이 달린 클래스에 지원하고 클래스에서 **XML**로 마샬링을 지원합니다. 다음은 **Camel CryostatB** 데이터 형식 클래스를 사용하여 마샬링 및 **unmarshalling**을 위한 간단한 **Camel** 경로를 보여줍니다.

5.3.1. CryostatB 주석 클래스

```
@XmlRootElement(name = "customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer implements Serializable {

    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }
```

```
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

5.3.2. CryostatB 클래스 XML 표현

```
<customer xmlns="http://org/wildfly/test/jaxb/model/Customer">
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</customer>
```

5.3.3. Camel CryostatB Unmarshalling

```
WildFlyCamelContext camelctx =
contextFactory.createCamelContext(getClass().getClassLoader());

final JaxbDataFormat jaxb = new JaxbDataFormat();
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");

camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .unmarshal(jaxb);
    }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();

// Send an XML representation of the customer to the direct:start endpoint
Customer customer = producer.requestBody("direct:start", readCustomerXml(),
Customer.class);
Assert.assertEquals("John", customer.getFirstName());
Assert.assertEquals("Doe", customer.getLastName());
```

5.3.4. Camel CryostatB Marshalling

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext();

final JaxbDataFormat jaxb = new JaxbDataFormat();
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");
```

```
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .marshal(jaxb);
   }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();
Customer customer = new Customer("John", "Doe");
String customerXML = producer.requestBody("direct:start", customer, String.class);
Assert.assertEquals(readCustomerXml(), customerXML);
```

5.4. JAX-RS

Cryostat-RS 지원은 Camel CXF-RS 에서 제공합니다.

5.4.1. CXF-RS Producer

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:cxf="http://camel.apache.org/schema/cxf"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

   <cxf:rsClient id="cxfProducer"
                 address="http://localhost:8080/rest"
                 serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

   <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
      <route>
         <from uri="direct:start" />
         <setHeader headerName="operationName">
            <simple>greet</simple>
         </setHeader>
         <setHeader headerName="CamelCxfRsUsingHttpAPI">
            <constant>false</constant>
         </setHeader>
         <to uri="cxfrs:bean:cxfProducer" />
      </route>
   </camelContext>
</beans>
```

5.4.2. CXF-RS 소비자

```
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://camel.apache.org/schema/cxf"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

    <cxf:rsServer id="cxfConsumer"
                  address="http://localhost:8080/rest"
                  serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

    <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="cxfrs:bean:cxfConsumer" />
            <setBody>
                <constant>Hello world</constant>
            </setBody>
        </route>
    </camelContext>
</beans>
```

5.4.3. Camel REST DSL을 사용한 Cryostat-RS 소비자

Camel REST DSL은 Cryostat-RS 소비자 역할을 하는 Camel 경로를 작성할 수 있는 기능을 제공합니다. 다음 RouteBuilder 클래스는 이를 보여줍니다.

```
@Startup
@ApplicationScoped
@ContextName("rest-camel-context")
public class RestConsumerRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {

    // Use the camel-undertow component to provide REST integration
    restConfiguration().component("undertow")
      .contextPath("/rest").port(8080).bindingMode(RestBindingMode.json);

    rest("/customer")
      // GET /rest/customer
      .get()
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomers")
      // GET /rest/customer/1
      .get("/{id}")
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomer")
      // POST /rest/customer
      .post()
        .type(Customer.class)
        .to("direct:createCustomer");
      // PUT /rest/customer
```

```
      .put()
        .type(Customer.class)
        .to("direct:updateCustomer");
      // DELETE /rest/customer/1
      .delete("/{id}")
        .to("direct:deleteCustomer");
  }
}
```

Camel은 바인딩 모드를 설정하여 'produces()' 또는 'type()' 구성 단계를 지정하여 JSON 데이터를 마샬링하고 해제할 수 있습니다.

참고

- REST DSL 구성은 restConfiguration().component("undertow") 로 시작됩니다.
- EAP Cryostat의 Camel은 REST DSL과 함께 사용할 수 있도록 camel-servlet 및 camel-undertow 구성 요소만 지원합니다. 그러나 다른 구성 요소를 구성하면 작동하지 않습니다.

5.4.4. 보안

Cryostat -RS 보안 섹션을 참조하십시오.

5.4.5. EAP의 Fuse 빠른 시작 예

빠른 시작 예제는 빠른 시작/camel/camel-cxf-jaxrs 디렉터리에 있는 EAP 설치의 Fuse에서 사용할 수 있습니다.

5.5. JAX-WS

WebService 지원은 Apache CXF 도 사용하는 JBoss EAP WebServices 하위 시스템과 통합된 CXF 구성 요소를 통해 제공됩니다.

5.5.1. Cryostat-WS CXF Producer

다음 코드 예제에서는 CXF를 사용하여 WildFly 웹 서비스 하위 시스템에서 배포한 웹 서비스를 사용합니다.

5.5.1.1. Cryostat-WS 웹 서비스

다음의 간단한 웹 서비스에는 두 개의 문자열 인수를 함께 연결하고 반환하는 간단한 '!' 방법이 있습니다.

JBoss EAP 웹 서비스 하위 시스템이 Cryostat-WS 주석이 포함된 클래스를 감지하면 CXF 엔드포인트를 부트스트랩합니다. 이 예에서 서비스 끝점은 http://hostname:port/context-root/greeting 에 있습니다.

```
// Service interface
@WebService(name = "greeting")
public interface GreetingService {
   @WebMethod(operationName = "greet", action = "urn:greet")
   String greet(@WebParam(name = "message") String message, @WebParam(name =
"name") String name);
}

// Service implementation
public class GreetingServiceImpl implements GreetingService{
   public String greet(String message, String name) {
      return message + " " + name ;
   }
}
```

5.5.1.2. Camel 경로 구성

이 RouteBuilder는 위에 정의된 '승인' 웹 서비스를 사용하는 CXF 생산자 엔드포인트를 구성합니다. camel-cdi 구성 요소와 함께 CDI는 RouteBuilder 및 CamelContext를 부트스트랩하는 데 사용됩니다.

```
@Startup
@ApplicationScoped
@ContextName("cxf-camel-context")
public class CxfRouteBuilder extends RouteBuilder {

   @Override
   public void configure() throws Exception {
      from("direct:start")
      .to("cxf://http://localhost:8080/example-camel-cxf/greeting?serviceClass=" +
GreetingService.class.getName());
   }
}
```

인사말 웹 서비스에는 두 개의 매개변수가 필요합니다. 이들은 ProducerTemplate 을 통해 위의 경로에 제공될 수 있습니다. 웹 서비스 메서드 인수 값은 교환 본문으로 전달되는 오브젝트 배열을 구성하여 구성됩니다.

```
String message = "Hello"
```

```
String name = "Kermit"

ProducerTemplate producer = camelContext.createProducerTemplate();
Object[] serviceParams = new Object[] {message, name};
String result = producer.requestBody("direct:start", serviceParams, String.class);
```

5.5.2. Camel CXF Cryostat-WS 소비자

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:cxf="http://camel.apache.org/schema/cxf"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

   <cxf:cxfEndpoint id="cxfConsumer"
                    address="http://localhost:8080/webservices/greeting"
                    serviceClass="org.wildfly.camel.examples.cxf.jaxws.GreetingService" />

   <camelContext id="cxfws-camel-context" xmlns="http://camel.apache.org/schema/spring">
      <route>
         <from uri="cxf:bean:cxfConsumer" />
         <to uri="log:ws" />
      </route>
   </camelContext>

</beans>
```

5.5.3. 보안

Cryostat -WS 보안 섹션을 참조하십시오.

5.5.4. EAP의 Fuse 빠른 시작 예

빠른 시작 예제는 빠른 시작/camel/camel-cxf-jaxws 디렉터리에 있는 EAP 설치의 Fuse에서 사용할 수 있습니다.

5.6. JMS

메시징 지원은 JBoss EAP Messaging(ActiveMQ Artemis) 하위 시스템과 통합된 JMS 구성 요소를 통해 제공됩니다.

JBoss Generic JMS 리소스 어댑터를 사용하여 벤더별 리소스 어댑터의 구성을 통해 또는 사용할 수

없는 경우 다른 **JMS** 구현과 통합할 수 있습니다.

5.6.1. JBoss EAP JMS 구성

표준 **JBoss EAP XML** 구성 파일을 통해 **JBoss EAP** 메시징 하위 시스템을 구성할 수 있습니다. 예를 들면 **standalone.xml** 또는 **domain.xml**입니다.

예를 들어 메모리 인스턴스에서 포함된 **ActiveMQ Artemis**를 사용합니다. 먼저 **jms-destinations** 섹션에 다음 **XML** 구성을 추가하여 메시징 하위 시스템에서 새 **JMS** 큐를 구성합니다.

```
<jms-queue name="WildFlyCamelQueue">
  <entry name="java:/jms/queue/WildFlyCamelQueue"/>
</jms-queue>
```

또는 **CLI** 스크립트를 사용하여 큐를 추가할 수도 있습니다.

```
$ jms-queue add --queue-address=WildFlyCamelQueue --
entries=queue/WildFlyCamelQueue,java:/jms/queue/WildFlyCamelQueue
```

또한 사용자 지정 **jms.xml** 배포 설명자 내에 **messaging-deployment** 구성을 생성할 수 있습니다. 자세한 내용은 **JBoss EAP** 메시징 하위 시스템 설명서의 **'jms.xml** 파일' 섹션을 참조하십시오.

5.6.2. Camel 경로 구성

다음 **JMS** 생산자 및 소비자 예제는 **JBoss EAP**의 포함된 **ActiveMQ Artemis** 서버를 사용하여 대상에 메시지를 게시하고 사용합니다.

이 예제에서는 **camel-cdi** 구성 요소와 함께 **CDI**를 사용합니다. **JMS ConnectionFactory** 인스턴스는 **JNDI** 조회를 통해 **Camel RouteBuilder**에 삽입됩니다.

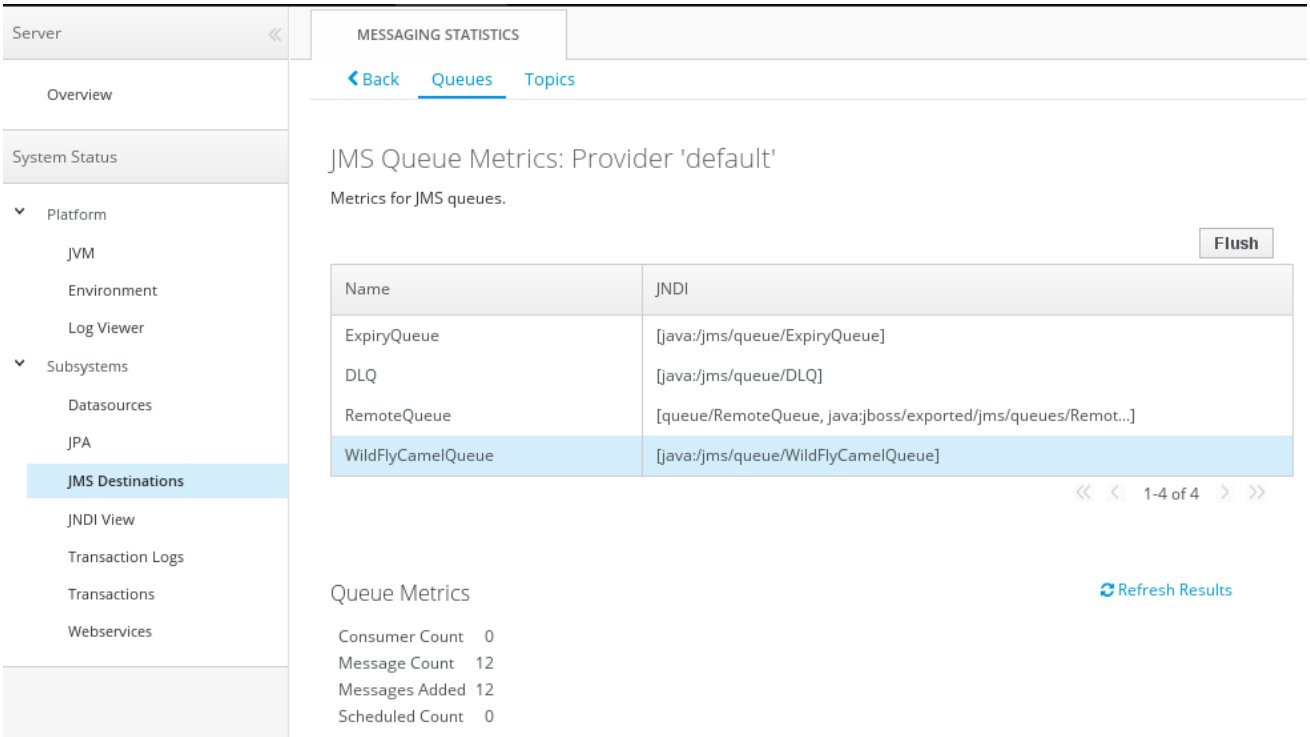
5.6.2.1. JMS Producer

DefaultJMSConnectionFactory 연결 팩토리가 **JNDI**의 **RouteBuilder**에 삽입됩니다. **JBoss EAP XML** 구성에서 메시징 하위 시스템 내에서 연결 팩토리를 찾을 수 있습니다.

다음으로 타이머 끝점은 **10**초마다 실행되어 이전에 구성된 **WildFlyCamelQueue** 대상에 **XML** 페이로드를 보냅니다.

```
@Startup
@ApplicationScoped
@ContextName("jms-camel-context")
public class JmsRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:jboss/DefaultJMSConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Override
  public void configure() throws Exception {
    JmsComponent component = new JmsComponent();
    component.setConnectionFactory(connectionFactory);

    getContext().addComponent("jms", component);

    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("jms:queue:WildFlyCamelQueue")
    .log("JMS Message sent");
  }
}
```

JMS 메시지가 **WildFlyCamelQueue** 대상에 추가될 때마다 로그 메시지가 콘솔에 출력됩니다. 메시지가 큐에 실제로 배치되었는지 확인하려면 **JBoss EAP** 관리 콘솔을 사용할 수 있습니다.



5.6.2.2. JMS 소비자

JMS 메시지를 사용하기 위해 **Camel RouteBuilder** 구현은 생산자 예제와 유사합니다.

이전과 마찬가지로, 연결 팩토리는 JNDI에서 검색되고 JMSComponent 인스턴스에 삽입 및 설정됩니다.

JMS 끝점에서 WildFlyCamelQueue 대상의 메시지를 사용하면 콘텐츠가 콘솔에 기록됩니다.

```
@Override
public void configure() throws Exception {
  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);

  getContext().addComponent("jms", component);

  from("jms:queue:WildFlyCamelQueue")
  .to("log:jms?showAll=true");
}
```

5.6.2.3. JMS 트랜잭션

Camel JMS 경로가 JMS 트랜잭션에 참여할 수 있도록 하려면 몇 가지 추가 구성이 필요합니다. camel-jms는 Spring-jms를 기반으로 빌드되었으므로 JBoss EAP의 트랜잭션 관리자 및 연결 팩토리에서 작업할 수 있도록 일부 Spring 클래스를 구성해야 합니다. 다음 코드 예제에서는 CDI를 사용하여 트랜잭션 JMS Camel 경로를 구성하는 방법을 보여줍니다.

camel-jms 구성 요소에는 org.springframework. Cryostat.PlatformTransactionManager 유형의 트랜잭션 관리자가 필요합니다. 따라서 JtaTransactionManager 를 확장한 빈을 생성하여 시작합니다. 8080에 @Named 주석이 추가되어, 8080이 Camel 8080 레지스트리 내에 등록될 수 있습니다. 또한 JBoss EAP 트랜잭션 관리자 및 사용자 트랜잭션 인스턴스는 CDI를 사용하여 삽입됩니다.

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {

  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

다음으로 사용하려는 트랜잭션 정책을 선언해야 합니다. 다시 @Named 주석을 사용하여 Camel에서 빈을 사용할 수 있도록 합니다. 트랜잭션 관리자는 원하는 트랜잭션 정책을 사용하여

TransactionTemplate을 생성할 수 있도록 삽입됩니다. 이 인스턴스의 PROPAGATION_REQUIRED.

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

이제 Camel RouteBuilder 클래스를 구성하고 Camel JMS 구성 요소에 필요한 종속성을 삽입할 수 있습니다. JBoss EAP XA 연결 팩토리는 이전에 구성한 트랜잭션 관리자와 함께 삽입됩니다.

이 예에서 RouteBuilder는 queue1에서 메시지를 사용할 때마다 queue2라는 다른 JMS 큐로 라우팅됩니다. queue2에서 사용되는 메시지로 인해 JMS 트랜잭션이 rollback() DSL 메서드를 사용하여 롤백됩니다. 이로 인해 원본 메시지가 dead letter queue(DLQ)에 배치됩니다.

```
@Startup
@ApplicationScoped
@ContextName("jms-camel-context")
public class JMSRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/JmsXA")
  private ConnectionFactory connectionFactory;

  @Inject
  CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    // Creates a JMS component which supports transactions
    JmsComponent jmsComponent =
JmsComponent.jmsComponentTransacted(connectionFactory, transactionManager);
    getContext().addComponent("jms", jmsComponent);

    from("jms:queue:queue1")
      .transacted("PROPAGATION_REQUIRED")
      .to("jms:queue:queue2");

    // Force the transaction to roll back. The message will end up on the Wildfly 'DLQ' message
queue
    from("jms:queue:queue2")
      .to("log:end")
      .rollback();
  }
```

5.6.2.4. 원격 JMS 대상

하나의 JBoss EAP 인스턴스는 원격 JNDI 를 통해 다른 JBoss EAP 인스턴스에 구성된 ActiveMQ Artemis 대상에 메시지를 보낼 수 있습니다.

이를 위해서는 몇 가지 추가 JBoss EAP 구성이 필요합니다. 먼저 내보낸 JMS 큐가 구성됩니다.

java:jboss/exported 네임스페이스에 바인딩된 JNDI 이름만 원격 클라이언트의 후보로 간주되므로 큐의 이름은 적절하게 지정됩니다.

참고

JBoss EAP 클라이언트 애플리케이션 서버 및JBoss EAP 원격 서버에서 큐를 구성해야 합니다.

```
<jms-queue name="RemoteQueue">
  <entry name="java:jboss/exported/jms/queues/RemoteQueue"/>
</jms-queue>
```

클라이언트가 원격 서버에 연결하려면 사용자 액세스 자격 증명을 구성해야 합니다. 원격 서버에서 add user 유틸리티 를 실행하여 'guest' 그룹 내에 새 애플리케이션 사용자를 생성합니다. 이 예제에는 이름이 'admin'이고 암호가 'secret'인 사용자가 있습니다.

RouteBuilder 구현은 이전 예제와 다릅니다. 연결 팩토리를 삽입하는 대신 InitialContext를 구성하고 JNDI에서 직접 검색해야 합니다.

configureInitialContext 메서드는 이 InitialContext를 생성합니다. 원격 JBoss EAP 인스턴스 호스트 이름과 포트 번호를 참조해야 하는 공급자 URL을 설정해야 합니다. 이 예제에서는 JBoss EAP JMS http-connector를 사용하지만 여기에 설명된 대안이 있습니다.

마지막으로 경로는 10초마다 구성된 원격 대상인 'RemoteQueue'로 XML 페이로드를 전송하도록 구성됩니다.

```
@Override
public void configure() throws Exception {
  Context initialContext = configureInitialContext();
  ConnectionFactory connectionFactory = (ConnectionFactory)
initialContext.lookup("java:jms/RemoteConnectionFactory");

  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);
```

```
  getContext().addComponent("jms", component);

  from("timer://foo?fixedRate=true&period=10000")
  .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
  .to("jms:queue:RemoteQueue?username=admin&password=secret")
  .to("log:jms?showAll=true");
}

private Context configureInitialContext() throws NamingException {
  final Properties env = new Properties();
  env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
  env.put(Context.PROVIDER_URL, System.getProperty(Context.PROVIDER_URL, "http-
remoting://my-remote-host:8080"));
  env.put(Context.SECURITY_PRINCIPAL, System.getProperty("username", "admin"));
  env.put(Context.SECURITY_CREDENTIALS, System.getProperty("password", "secret"));
  return new InitialContext(env);
}
```

5.6.3. 보안

JMS 보안 섹션을 참조하십시오.

5.6.4. EAP의 Fuse 빠른 시작 예

빠른 시작 예제는 빠른 시작/camel/camel-jms 디렉터리에 있는 EAP 설치의 Fuse에서 사용할 수 있습니다.

5.7. JMX

JBoss EAP Cryostat 하위 시스템과 통합되는 Cryo stat 구성 요소를 통해 관리 지원을 제공할 수 있습니다.

```
CamelContext camelctx =
contextFactory.createWildflyCamelContext(getClass().getClassLoader());
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      String host = InetAddress.getLocalHost().getHostName();
      from("jmx:platform?format=raw&objectDomain=org.apache.camel&key.context=" + host +
"/system-context-1&key.type=routes&key.name=\"route1\"" +
      "&monitorType=counter&observedAttribute=ExchangesTotal&granularityPeriod=500").
      to("direct:end");
   }
});
camelctx.start();
```

```
ConsumerTemplate consumer = camelctx.createConsumerTemplate();
MonitorNotification notifcation = consumer.receiveBody("direct:end",
MonitorNotification.class);
Assert.assertEquals("ExchangesTotal", notifcation.getObservedAttribute());
```

5.8. JNDI

JNDI 통합은 다음과 같이 JBoss EAP 특정 CamelContext에서 제공합니다.

```
InitialContext inictx = new InitialContext();
CamelContextFactory factory = inictx.lookup("java:jboss/camel/CamelContextFactory");
WildFlyCamelContext camelctx = factory.createCamelContext();
```

WildFlyCamelContext 에서 사전 구성된 이름 지정 컨텍스트를 얻을 수 있습니다.

```
Context context = camelctx.getNamingContext();
context.bind("helloBean", new HelloBean());
```

그런 다음 Camel 경로에서 참조할 수 있습니다.

```
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start").beanRef("helloBean");
   }
});
camelctx.start();
```

5.9. JPA

JPA 통합은 Camel JPA 구성 요소에서 제공합니다. 일부 JPA 주석이 지정된 클래스와 함께 persistence.xml 구성 파일을 제공하여 Camel JPA 애플리케이션을 개발할 수 있습니다.

5.9.1. persistence.xml의 예

다음 예제에서는 JBoss EAP standalone.xml 구성 파일 내에서 구성된 JBoss EAP in-memory ExampleDS 데이터 소스를 사용할 수 있습니다.

```
<persistence version="2.0"
   xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
```

```
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="camel">
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
        <class>org.wildfly.camel.test.jpa.model.Customer</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>

</persistence>
```

5.9.2. JPA 종료 예

```
@Entity
@Table(name = "customer")
public class Customer implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Long getId() {
        return id;
    }

    public void setId(final Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
```

```
        this.lastName = lastName;
    }
}
```

5.9.3. Camel JPA 끝점 / 경로 구성

JPA를 구성하면 CDI를 사용하여 EntityManager 및 UserTransaction 인스턴스를 RouteBuilder 클래스 또는 테스트 케이스에 삽입할 수 있습니다.

```
@PersistenceContext
EntityManager em;

@Inject
UserTransaction userTransaction;
```

이제 Camel 경로 및 JPA 끝점을 구성합니다.

```
WildFlyCamelContext camelctx =
contextFactory.createCamelContext(getClass().getClassLoader());

EntityManagerFactory entityManagerFactory = em.getEntityManagerFactory();

// Configure a transaction manager
JtaTransactionManager transactionManager = new JtaTransactionManager();
transactionManager.setUserTransaction(userTransaction);
transactionManager.afterPropertiesSet();

// Configure the JPA endpoint to use the correct EntityManagerFactory and
JtaTransactionManager
final JpaEndpoint jpaEndpoint = new JpaEndpoint();
jpaEndpoint.setCamelContext(camelctx);
jpaEndpoint.setEntityType(Customer.class);
jpaEndpoint.setEntityManagerFactory(entityManagerFactory);
jpaEndpoint.setTransactionManager(transactionManager);

camelctx.addRoutes(new RouteBuilder() {
@Override
public void configure() throws Exception {
    from("direct:start")
    .to(jpaEndpoint);
}
});

camelctx.start();
```

마지막으로 '고객' 엔터티를 'direct:start' 엔드포인트에 보낸 다음 ExampleDS 데이터 소스를 쿼리하여 레코드가 저장되었는지 확인할 수 있습니다.

```
Customer customer = new Customer("John", "Doe");
ProducerTemplate producer = camelctx.createProducerTemplate();
producer.sendBody("direct:start", customer);

// Query the in memory database customer table to verify that a record was saved
CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
CriteriaQuery<Long> query = criteriaBuilder.createQuery(Long.class);
query.select(criteriaBuilder.count(query.from(Customer.class)));

long recordCount = em.createQuery(query).getSingleResult();

Assert.assertEquals(1L, recordCount);
```

6장. CAMEL 구성 요소

이 장에서 지원되는 camel 구성 요소에 대한 자세한 정보

6.1. CAMEL-ACTIVEMQ

Camel ActiveMQ 통합은 activemq 구성 요소에서 제공합니다.

포함된 또는 외부 브로커와 작동하도록 구성 요소를 구성할 수 있습니다. Wildfly / EAP 컨테이너 관리 연결 풀 및 XA-Transaction 지원의 경우 ActiveMQ 리소스 어댑터 를 컨테이너 구성 파일에 구성할 수 있습니다.

6.1.1. JBoss EAP ActiveMQ 리소스 어댑터 구성

ActiveMQ 리소스 어댑터 rar 파일을 다운로드합니다. 다음 단계에서는 ActiveMQ 리소스 어댑터를 구성하는 방법을 간략하게 설명합니다.

1. JBoss EAP 인스턴스를 중지합니다.

2. 리소스 어댑터를 다운로드하여 관련 JBoss EAP 배포 디렉터리에 복사합니다. 독립 실행형 모드의 경우:

```
cp activemq-rar-5.11.1.rar ${JBOSS_HOME}/standalone/deployments/activemq-rar.rar
```

3. ActiveMQ 어댑터에 대한 JBoss EAP 리소스 어댑터 하위 시스템을 구성합니다.

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
    <resource-adapters>
        <resource-adapter id="activemq-rar.rar">
            <archive>
                activemq-rar.rar
            </archive>
            <transaction-support>XATransaction</transaction-support>
            <config-property name="UseInboundSession">
                false
            </config-property>
            <config-property name="Password">
                defaultPassword
            </config-property>
            <config-property name="UserName">
```

```
            defaultUser
          </config-property>
          <config-property name="ServerUrl">
            tcp://localhost:61616?jms.rmIdFromConnectionId=true
          </config-property>
          <connection-definitions>
            <connection-definition class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory" jndi-
name="java:/ActiveMQConnectionFactory" enabled="true" pool-name="ConnectionFactory">
              <xa-pool>
                <min-pool-size>1</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>false</prefill>
                <is-same-rm-override>false</is-same-rm-override>
              </xa-pool>
            </connection-definition>
          </connection-definitions>
          <admin-objects>
            <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/HELLOWORLDMDBQueue" use-java-context="true" pool-
name="HELLOWORLDMDBQueue">
              <config-property name="PhysicalName">
                HELLOWORLDMDBQueue
              </config-property>
            </admin-object>
            <admin-object class-name="org.apache.activemq.command.ActiveMQTopic" jndi-
name="java:/topic/HELLOWORLDMDBTopic" use-java-context="true" pool-
name="HELLOWORLDMDBTopic">
              <config-property name="PhysicalName">
                HELLOWORLDMDBTopic
              </config-property>
            </admin-object>
          </admin-objects>
        </resource-adapter>
      </resource-adapters>
    </subsystem>
```

리소스 어댑터 아카이브 파일 이름이 **activemq-rar.rar**와 다른 경우 아카이브 파일의 이름과 일치하도록 이전 구성의 아카이브 요소 내용을 변경해야 합니다.

외부 브로커에 있는 유효한 사용자의 자격 증명과 일치하도록 **UserName** 및 **Password** 구성 속성 값을 선택해야 합니다.

외부 브로커가 노출하는 실제 호스트 이름 및 포트와 일치하도록 **ServerUrl** 구성 속성 값을 변경해야 할 수 있습니다.

4) **JBoss EAP**를 시작합니다. 모든 항목이 올바르게 구성된 경우 다음과 같은 **JBoss EAP server.log**에 메시지가 표시됩니다.

```
13:16:08,412 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1-5) JBAS010406:
Registered connection factory java:/AMQConnectionFactory`
```

6.1.2. Camel 경로 구성

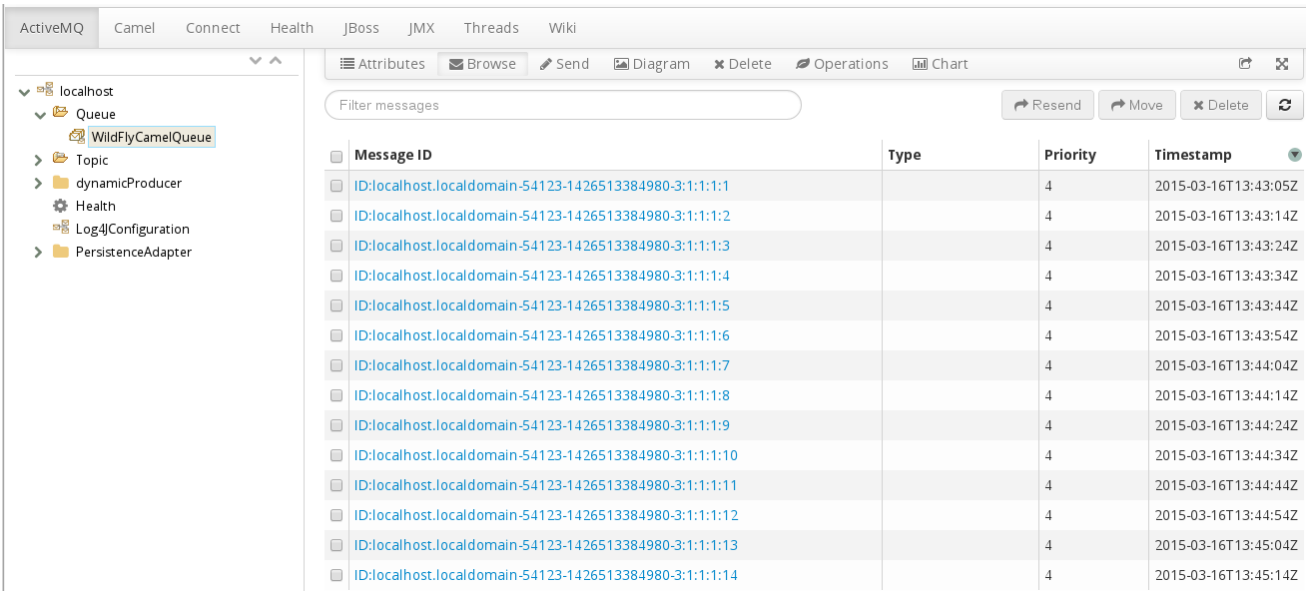
다음 **ActiveMQ** 생산자 및 소비자 예제에서는 **ActiveMQ** 포함 브로커 및 **'vm'** 전송(외부 **ActiveMQ** 브로커의 필요성을 방지)을 사용합니다.

예에서는 **camel-cdi** 구성 요소와 함께 **CDI**를 사용합니다. **JMS ConnectionFactory** 인스턴스는 **JNDI** 조회를 통해 **Camel RouteBuilder**에 삽입됩니다.

6.1.2.1. ActiveMQ Producer

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
    .log("JMS Message sent");
  }
}
```

메시지가 **WildFlyCamelQueue** 대상에 추가될 때마다 로그 메시지가 콘솔에 출력됩니다. 메시지가 실제로 큐에 배치되는지 확인하려면 **EAP** 하위 시스템에서 **Camel**에서 제공하는 **../features/hawtio.md[Hawtio** 콘솔**,window=_blank]**을 사용할 수 있습니다.



6.1.2.2. ActiveMQ 소비자

ActiveMQ 메시지를 사용하기 위해 Camel RouteBuilder 구현은 생산자 예제와 유사합니다.

ActiveMQ 끝점에서 WildFlyCamelQueue 대상의 메시지를 사용하면 콘텐츠가 콘솔에 기록됩니다.

```
@Override
public void configure() throws Exception {
 from("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
 .to("log:jms?showAll=true");
}
```

6.1.2.3. ActiveMQ 트랜잭션

6.1.2.3.1. ActiveMQ 리소스 어댑터 구성

XA 트랜잭션 지원, 연결 풀링 등을 활용하려면 ActiveMQ 리소스 어댑터가 필요합니다.

아래 XML 스니펫에서는 JBoss EAP 서버 XML 구성 내에서 리소스 어댑터가 구성된 방법을 보여줍니다. ServerURL 은 포함된 브로커를 사용하도록 설정되어 있습니다. 연결 팩토리는 JNDI 이름 java:/ActiveMQConnectionFactory 에 바인딩됩니다. 이는 다음 RouteBuilder 예제에서 조회됩니다.

마지막으로 두 개의 큐가 'queue1' 및 'queue2'라는 이름으로 구성됩니다.

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
 <resource-adapters>
  <resource-adapter id="activemq-rar.rar">
   ...
   <admin-objects>
    <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue1" use-java-context="true" pool-name="queue1pool">
     <config-property name="PhysicalName">queue1</config-property>
    </admin-object>
    <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue2" use-java-context="true" pool-name="queue2pool">
     <config-property name="PhysicalName">queue2</config-property>
    </admin-object>
   </admin-objects>
  </resource-adapter>
 </resource-adapters>
</subsystem>
```

6.1.2.4. 트랜잭션 관리자

camel-activemq 구성 요소에는 org.springframework. Cryostat.PlatformTransactionManager 유형의 트랜잭션 관리자가 필요합니다. 따라서 이 요구 사항을 충족하는 JtaTransactionManager 확장 함수를 생성하여 시작할 수 있습니다. 8080에 @Named 주석이 추가되어, 8080이 Camel 8080 레지스트리 내에 등록될 수 있습니다. 또한 JBoss EAP 트랜잭션 관리자 및 사용자 트랜잭션 인스턴스는 CDI를 사용하여 삽입됩니다.

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {

  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

6.1.2.5. 거래 정책

다음으로 사용하려는 트랜잭션 정책을 선언해야 합니다. 다시 @Named 주석을 사용하여 Camel에서 빈을 사용할 수 있도록 합니다. 트랜잭션 관리자는 원하는 트랜잭션 정책을 사용하여 TransactionTemplate 을 생성할 수 있도록 삽입됩니다. 이 인스턴스의 PROPAGATION_REQUIRED.

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

6.1.2.6. Route Builder

이제 Camel RouteBuilder 클래스를 구성하고 Camel ActiveMQ 구성 요소에 필요한 종속 항목을 삽입할 수 있습니다. 리소스 어댑터 구성에 구성한 ActiveMQ 연결 팩토리는 이전에 구성한 트랜잭션 관리자와 함께 삽입됩니다.

이 예에서 RouteBuilder는 queue1에서 메시지를 사용할 때마다 queue2라는 다른 JMS 큐로 라우팅됩니다. queue2에서 사용되는 메시지로 인해 JMS 트랜잭션이 rollback() DSL 메서드를 사용하여 롤백됩니다. 이로 인해 원본 메시지가 dead letter queue(DLQ)에 배치됩니다.

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/ActiveMQConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Inject
  private CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
    activeMQComponent.setTransacted(false);
    activeMQComponent.setConnectionFactory(connectionFactory);
    activeMQComponent.setTransactionManager(transactionManager);

    getContext().addComponent("activemq", activeMQComponent);

      errorHandler(deadLetterChannel("activemq:queue:ActiveMQ.DLQ")
      .useOriginalMessage()
      .maximumRedeliveries(0)
      .redeliveryDelay(1000));

    from("activemq:queue:queue1F")
      .transacted("PROPAGATION_REQUIRED")
      .to("activemq:queue:queue2");

    from("activemq:queue:queue2")
      .to("log:end")
      .rollback();
  }
}
```

6.1.3. 보안

JMS 보안 섹션을 참조하십시오.

6.1.4. GitHub의 코드 예제

GitHub에서 camel-activemq 애플리케이션 예를 사용할 수 있습니다.

6.2. CAMEL-JMS

camel-jms, camel-sjms 및 camel-sjms2 끝점을 원격 AMQ 7 브로커에 연결하는 데 지원되는 두 가지 방법이 있습니다.

1. JBoss EAP 구성 메시징 가이드의 Artemis Resource Adapter to Connect to Red Hat JBoss AMQ 7 이라는 섹션에 설명된 대로 pooled-connection-factory를 사용하여 원격 연결을 구성합니다.

2. 에 설명된 대로 connection-factory를 사용하여 원격 연결 장치 구성 connection-factory를 사용하여 remote-connector 구성

첫 번째 옵션은 연결 풀링 및 XA 트랜잭션 지원을 제공하므로 권장되는 방법입니다.

Cryostat 구독자를 사용하는 메시징 시나리오의 경우, Cryostat 7 사양에 따른 제약 조건으로 인해 JBoss EAP의 Fuse 7.9에서 pooled-connection-factory를 지원하지 않습니다. 이러한 시나리오에서는 표준 풀링되지 않은 연결 요소를 구성하는 것이 좋습니다.

connection-factory를 사용하여 remote-connector 구성

1. 원격 메시징 서버를 가리키는 outbound-socket-binding을 생성합니다.

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=localhost, port=61616)
```

2. 1단계에서 생성된 outbound-socket-binding을 참조하는 remote-connector를 만듭니다.

```
/subsystem=messaging-activemq/server=default/remote-connector=netty-remote-throughput:add(socket-binding=messaging-remote-throughput)
```

3. 2단계에서 생성된 remote-connector를 참조하는 connection-factory를 생성합니다.

```
/subsystem=messaging-activemq/server=default/connection-factory=simple-remote-artemis-connection-factory:add(entries=[java:/jms/RemoteJms],connectors=[netty-remote-throughput])
```

6.2.1. 메시징 브로커 및 클라이언트

초록

Fuse 7.9는 기본 내부 메시징 브로커와 함께 제공되지 않지만 외부 JMS 브로커와 상호 작용하도록 설계되었습니다.

JBoss EAP의 Fuse 7.9는 JBoss EAP에서 메시징 구성에 설명된 리소스 어댑터를 사용하여 외부 메시징 브로커에 액세스합니다.

JBoss EAP의 Fuse 7.9 메시징에 사용할 수 있는 외부 브로커, JCA 어댑터 및 Camel 구성 요소 조합에 대한 자세한 내용은 지원 구성을 참조하십시오.

JMS를 사용하여 JBoss EAP에서 Fuse를 사용하여 외부 브로커에 연결하는 방법에 대한 자세한 내용은 6.2절. “camel-jms” 을 참조하십시오.

Camel-jms 빠른 시작

JMS 메시지를 생성하고 사용하기 위해 Fuse on JBoss EAP에서 camel-jms 구성 요소를 사용하는 방법을 보여주는 빠른 시작이 제공됩니다.

이 빠른 시작에서 Camel 경로는 EAP_HOME/standalone/data/orders 의 파일을 사용하고 OrdersQueue 라는 메모리 내 ActiveMQ Artemis 큐에 콘텐츠를 배치합니다. 그런 다음 다른 Camel 경로는 OrdersQueue 내용을 사용하고 주문을 EAP_HOME/standalone/data/orders/processed 내의 개별 국가 디렉터리로 정렬합니다.

CLI 명령은 OrdersQueue CLI 스크립트를 생성 및 삭제합니다. 애플리케이션이 배포 및 배포 취소될 때 JMS OrdersQueue 를 생성하고 제거합니다. 이러한 스크립트는 EAP_HOME/quickstarts/camel-jms/src/main/resources/cli 디렉터리에 있습니다.

사전 요구 사항

이 빠른 시작을 실행하려면 작동 중인 Fuse 7.9 버전이 있어야 합니다.

외부 AMQ 7 브로커에 연결하려면 원격 JMS communication에 JBoss AMQ 사용 지침도 따라야 합니다. 그런 다음 기본 연결 팩토리와 마찬가지로 연결 팩토리를 삽입할 수 있습니다.

```
@Resource(mappedName = "java:jboss/RemoteJmsXA")
ConnectionFactory connectionFactory;
```

퀵스타트 설정

1. 독립 실행형 모드에서 JBOSS EAP를 시작합니다.

2. EAP_HOME/quickstarts/camel/camel-jms로 이동합니다.

3. mvn clean install -Pdeploy' 를 입력하여 빠른 시작을 빌드하고 배포합니다.

4. http://localhost:8080/example-camel-jms으로 이동

'Orders Received'라는 페이지가 표시됩니다. 예제 애플리케이션에 주문을 보내면 국가당 주문 목록이 이 페이지에 나열됩니다.

퀵스타트 실행

EAP_HOME/quickstarts/camel/camel-jms/src/main/resources 디렉터리에는 몇 가지 예제 XML 파일이 있습니다. Camel은 5초마다 무작위로 파일을 선택하고 처리를 위해 EAP_HOME/standalone/data/orders 에 복사합니다.

콘솔은 각 주문에 발생한 작업을 자세히 설명하는 메시지를 출력합니다. 출력은 다음과 같습니다.

```
JmsConsumer[OrdersQueue]) Sending order to the UK
JmsConsumer[OrdersQueue]) Sending order to another country
JmsConsumer[OrdersQueue]) Sending order to the US
```

파일이 소비되면 http://localhost:8080/example-camel-jms/orders 로 돌아갈 수 있습니다. 각 국가에 대한 주문 수는 1로 증가해야합니다.

처리된 모든 주문은 다음 대상으로 나뉩니다.

```
EAP_HOME/standalone/data/orders/processed/uk
EAP_HOME/standalone/data/orders/processed/us
EAP_HOME/standalone/data/orders/processed/other
```

배포 취소

예제 배포를 취소하려면 EAP_HOME/quickstarts/camel/camel-jms 로 이동합니다. mvn clean -Pdeploy.

6.3. CAMEL-MAIL

이메일과의 상호 작용은 메일 구성 요소에서 제공합니다.

기본적으로 Camel은 자체 메일 세션을 만들고 이를 사용하여 메일 서버와 상호 작용합니다. JBoss EAP는 이미 보안 연결, 사용자 이름 및 암호 암호화 등에 대한 모든 관련 지원을 제공하는 메일 하위 시스템을 제공하므로 JBoss EAP 구성 내에서 메일 세션을 구성하고 JNDI를 사용하여 Camel 엔드포인트로 전달하는 것이 좋습니다.

6.3.1. JBoss EAP 구성

먼저 메일 서버에 대한 JBoss EAP 메일 하위 시스템을 구성합니다. 다음 예제에서는 Google Mail Cryostat 및 SMTP에 대한 구성을 추가합니다.

추가 mail-session은 'default' 세션 후에 구성됩니다.

```
<subsystem xmlns="urn:jboss:domain:mail:2.0">
  <mail-session name="default" jndi-name="java:jboss/mail/Default">
    <smtp-server outbound-socket-binding-ref="mail-smtp"/>
  </mail-session>
```

```
  <mail-session debug="true" name="gmail" jndi-name="java:jboss/mail/gmail">
    <smtp-server outbound-socket-binding-ref="mail-gmail-smtp" ssl="true" username="your-username-here" password="your-password-here"/>
    <imap-server outbound-socket-binding-ref="mail-gmail-imap" ssl="true" username="your-username-here" password="your-password-here"/>
  </mail-session>
</subsystem>
```

'mail-gmail-smtp' 및 'mail-gmail-imap'의 outbound-socket-binding-ref 값을 구성할 수 있습니다.

다음 단계는 이러한 소켓 바인딩을 구성하는 것입니다. socket-binding-group 구성에 다음과 같이 바인딩을 추가할 수 있습니다.

```
<outbound-socket-binding name="mail-gmail-smtp">
  <remote-destination host="smtp.gmail.com" port="465"/>
</outbound-socket-binding>

<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="imap.gmail.com" port="993"/>
</outbound-socket-binding>
```

이렇게 하면 포트 465 및 포트 993의 imap.gmail.com의 호스트 smtp.gmail.com에 연결하도록 메일 세션이 구성됩니다. 다른 메일 호스트를 사용하는 경우 이 세부 정보가 달라집니다.

6.3.2. POP3 설정

POP3 세션을 구성해야 하는 경우 원칙이 위의 예제에 정의된 것과 동일합니다.

```
<!-- Server configuration -->
<pop3-server outbound-socket-binding-ref="mail-pop3" ssl="true" username="your-username-here"
password="your-password-here"/>

<!-- Socket binding configuration -->
<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="pop3.gmail.com" port="993"/>
</outbound-socket-binding>
```

6.3.3. Camel 경로 구성

6.3.3.1. 메일 프로듀서

이 예에서는 camel-cdi 구성 요소와 함께 CDI와 함께 SMTPS 프로토콜을 사용합니다. JBoss EAP 구성 내에서 구성한 Java 메일 세션은 JNDI를 통해 Camel RouteBuilder에 삽입됩니다.

6.3.3.1.1. 라우팅 빌더 SMTPS 예

GECDHE 메일 세션은 이전에 구성한 jndi-name 속성에 대한 참조와 함께 @Resource 주석을 사용하여 Producer 클래스에 삽입됩니다. 이를 통해 camel-mail 엔드포인트 구성에서 메일 세션을 참조할 수 있습니다.

```
public class MailSessionProducer {
   @Resource(lookup = "java:jboss/mail/greenmail")
   private Session mailSession;

   @Produces
   @Named
   public Session getMailSession() {
      return mailSession;
   }
}
```

```
public class MailRouteBuilder extends RouteBuilder {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .to("smtps://smtp.gmail.com?session=#mailSession");
   }
}
```

이메일을 보내려면 ProducerTemplate을 생성하고 필요한 이메일 헤더와 함께 적절한 본문을 보낼 수 있습니다.

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("To", "destination@test.com");
headers.put("From", "sender@example.com");
headers.put("Subject", "Camel on Wildfly rocks");

String body = "Hi,\n\nCamel on Wildfly rocks!.";

ProducerTemplate template = camelContext.createProducerTemplate();
template.sendBodyAndHeaders("direct:start", body, headers);
```

6.3.3.2. 메일 소비자

이메일을 수신하려면 Cryostat mailEndpoint를 사용할 수 있습니다. Camel 경로 구성은 다음과 같습니다.

```
public void configure() throws Exception {
  from("imaps://imap.gmail.com?session=#mailSession")
  .to("log:email");
}
```

6.3.4. 보안

6.3.4.1. SSL 구성

SSL/TLS를 사용하여 Java 메일 세션 및 관련 전송을 관리하도록 JBoss EAP를 구성할 수 있습니다. 메일 세션을 구성할 때 서버 유형에 대해 SSL 또는 TLS를 구성할 수 있습니다.

- smtp-server
- imap-server
- pop-server

ssl="true" 또는 tls="true" 속성을 설정합니다.

6.3.4.2. 암호 보안

구성 파일 내의 암호에는 일반 텍스트를 사용하지 않는 것이 좋습니다. WildFly Vault 를 사용하여 중요한 데이터를 마스킹할 수 있습니다.

6.3.4.3. Camel 보안

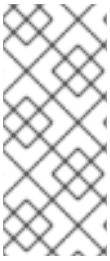
Camel 끝점 보안 문서는 메일 구성 요소 가이드에서 확인할 수 있습니다. Camel에는 보안 요약 페이지도 있습니다.

6.3.5. GitHub의 코드 예제

예제 camel-mail 애플리케이션은 GitHub에서 이메일 전송을 시도하거나 이메일을 수신해 볼 수 있습니다.

6.4. CAMEL-REST

나머지 구성 요소를 사용하면 REST 끝점을 Rest DSL 을 사용하여 정의하고 REST 전송으로 다른 Camel 구성 요소에 대한 플러그인을 사용할 수 있습니다.



참고

EAP Cryostat의 Camel은 REST DSL과 함께 사용할 수 있도록 camel-servlet 및 camel-undertow 구성 요소만 지원합니다. 그러나 하위 시스템은 작동하지 않습니다. 다른 구성 요소를 구성하려고 하면 됩니다.

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      restConfiguration().component("servlet").contextPath("camel/rest").port(8080);
      rest("/hello").get("/{name}").to("direct:hello");
      from("direct:hello").transform(simple("Hello ${header.name}"));
   }
});
```

6.5. CAMEL-REST-SWAGGER

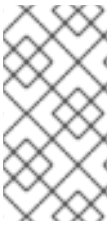
rest-swagger 구성 요소는 Swagger 문서에서 REST 생산자를 구성하고 다음과 같은 RestProducerFactory 인터페이스를 구현하는 구성 요소에 위임할 수 있습니다.

- camel-http4
- camel-undertow

6.6. CAMEL-SQL

SQL 구성 요소를 사용하면 JDBC 쿼리를 사용하여 데이터베이스를 사용할 수 있습니다. 이 구성 요소와 JDBC 구성 요소의 차이점은 SQL의 경우 쿼리는 끝점의 속성이며 쿼리에 전달된 매개 변수로 메시지 페이로드를 사용한다는 것입니다.

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("sql:select name from information_schema.users?
dataSource=java:jboss/datasources/ExampleDS")
      .to("direct:end");
   }
});
```



참고

위에 표시된 JNDI 데이터 소스 조회는 DefaultCamelContext 를 구성하는 경우에만 작동합니다. 아래 CdiCamelContext 및 SpringCamelContext 예제를 참조하십시오.

camel-cdi 구성 요소와 함께 사용하면 Jarkarta EE 주석이 있으면 Camel에서 데이터 소스를 사용할 수 있습니다. 이 예에서는 Camel이 원하는 데이터 소스를 검색할 수 있도록 @Named 주석을 사용합니다.

```
public class DatasourceProducer {
   @Resource(lookup = "java:jboss/datasources/ExampleDS")
   DataSource dataSource;

   @Produces
   @Named("wildFlyExampleDS")
   public DataSource getDataSource() {
      return dataSource;
   }
}
```

이제 camel-sql 엔드포인트 구성의 dataSource 매개변수를 통해 데이터 소스를 참조할 수 있습니다.

```
@ApplicationScoped
@ContextName("camel-sql-cdi-context")
@Startup
public class CdiRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("sql:select name from information_schema.users?dataSource=wildFlyExampleDS")
        .to("direct:end");
    }
}
```

camel-spring 을 사용할 때 경로 구성은 다음과 같습니다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
       http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-
jee.xsd">

    <jee:jndi-lookup id="wildFlyExampleDS" jndi-name="java:jboss/datasources/ExampleDS"/>

    <camelContext id="sql-spring-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="sql:select name from information_schema.users?dataSource=#wildFlyExampleDS"
/>
            <to uri="direct:end" />
        </route>
    </camelContext>

</beans>
```

6.6.1. Spring JDBC XML 네임스페이스 지원

다음 Spring JDBC XML 구성 지원이 지원됩니다.

jdbc:embedded-database

```
<jdbc:embedded-database id="datasource" type="H2">
  <jdbc:script location="db-schema.sql"/>
</jdbc:embedded-database>
```

참고

JBoss EAP는 이에 대한 기본 지원이 있으므로 H2 데이터베이스만 기본적으로 지원됩니다. 다른 포함된 데이터베이스 공급자를 사용하려면 적절한 데이터베이스 드라이버를 설치해야 합니다.

jdbc:initialize-database

```
<jdbc:initialize-database data-source="datasource">
  <jdbc:script location="classpath:db-init.sql"/>
</jdbc:initialize-database>
```

6.7. CAMEL-SOAP-REST-BRIDGE

간단한 Camel 경로는 REST 호출을 레거시 Cryostat 서비스에 브리지할 수 있습니다. Camel의 REST DSL과 함께 camel-soap-rest-bridge 구성 요소의 사용을 설명하기 위해 빠른 시작 예제가 제공됩니다. backend Cryostat API 서비스를 노출합니다.

이 빠른 시작에서는 RH SSO에서 지원하는 REST 끝점과 Cryostat 끝점 모두에 보안이 포함됩니다. 프런트 엔드 REST API는 OAuth 및 OpenID Connect를 통해 보호되며 클라이언트는 "리소스 소유자 암호 인증 정보" OAuth2 모드를 사용하여 RH SSO에서 JWT(JSON 웹 토큰) 액세스 토큰을 가져옵니다. 클라이언트는 이 토큰을 사용하여 REST 엔드포인트에 액세스합니다.

브리지 Camel 경로에서 클라이언트 ID는 SecurityContext에서 전파되며 camel-cxf 프로듀서 가 백엔드 WS-SECURITY로 통신할 때 처음에 이 클라이언트 ID를 사용하여 CXF STS 서비스에서 발급한 SAML2 토큰을 가져옵니다(ID 공급자로 RH SSO가 지원됨). SAML2 토큰은 서명되어 WS-SECURITY 헤더에 추가되고 백엔드 WS-SECURITY 보호 Cryostat 서비스는 이 SAML2 토큰의 유효성을 검사합니다.

Cryostat 호출에는 XSD 스키마 유효성 검사도 포함되어 있습니다. 토큰 유효성 검사가 성공하면 backend Cryostat 서비스에서 요청을 시작한 REST 클라이언트에 대한 응답을 반환합니다.

사전 요구 사항

1. JBoss EAP 7.3 이상을 설치했습니다.
2. Apache Maven 3.3.x 이상 버전을 설치했습니다.
3. RH SSO 7.4를 설치 및 구성했습니다. https://access.redhat.com/documentation/en-

us/red_hat_single_sign-on/7.4/html/getting_started_guide/installing-standalone_#installing-server-product에서 설치 지침을 따르십시오.

4. RH SSO EAP 어댑터를 설치했습니다. https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.4/html/getting_started_guide/securing-sample-app_#installing-client-adapter에서 설치 지침을 따르십시오.

퀵스타트 설정

1. 독립 실행형 모드에서 JBOSS EAP를 시작합니다.

2. EAP_HOME/quickstarts/camel/camel-soap-rest-bridge로 이동합니다.

3. mvn clean install -Pdeploy 를 입력하여 빠른 시작을 빌드하고 배포합니다.

4. RH SSO 구성

   a. 사용자 이름/암호 admin/admin을 사용하여 http://localhost:8180/auth 에서 RH SSO 관리 콘솔에 로그인

   b. 영역 추가를클릭합니다.

   c. 파일 선택을클릭합니다.

   d. 이 예제 폴더에서 ./src/main/resources/keycloak-config/realm-export-new.json을 선택하여 미리 정의된 realm/client/user/role을 가져옵니다.

   e. 생성을 클릭합니다.

EAP의 Fuse 빠른 시작 예

빠른 시작 및 테스트 사례 결과에 대한 추가 정보가 포함된 빠른 시작 예제는 EAP 설치의 EAP에 대한 Fuse on EAP 설치에서 사용할 수 있습니다. EAP_HOME/quickstarts/camel/soap-rest-bridge 디렉터리.

배포 취소

예제 배포를 취소하려면 **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** 디렉터리로 이동하여 **mvn clean -Pdeploy** 를 실행합니다.

6.8. 구성 요소 추가

추가 **Camel** 구성 요소에 대한 지원을 쉽게 추가할 수 있습니다.

modules.xml 정의 추가

modules.xml 설명자는 구성 요소의 클래스 로드 동작을 정의합니다. 이 구성 요소는 **module/system/layers/fuse/org/apache/camel/component** 에 구성 요소의 와 함께 배치해야 합니다. 직접 컴파일 시간 종속성에 대해 모듈 종속성을 설정해야 합니다.

camel-ftp 구성 요소의 예는 다음과 같습니다.

```
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel.component.ftp">
  <resources>
    <resource-root path="camel-ftp-2.14.0.jar" />
  </resources>
  <dependencies>
    <module name="com.jcraft.jsch" />
    <module name="javax.xml.bind.api" />
    <module name="org.apache.camel.core" />
    <module name="org.apache.commons.net" />
  </dependencies>
</module>
```

WildFly에서 이미 사용 가능한 모듈을 복제하지 말고 재사용할 수 있는지 확인하십시오.

구성 요소에 대한 참조 추가

이 모듈을 기본적으로 임의의 **Cryostat** 배포에 표시되도록 하려면 **module /system/layers/fuse/org/apache/camel/component/main/module.xml**에 대한 참조를 추가합니다.

```
<module xmlns="urn:jboss:module:1.3" name="org.apache.camel.component">
  <dependencies>
    ...
    <module name="org.apache.camel.component.ftp" export="true" services="export"/>
  </dependencies>
</module>
```

7장. 보안

JBoss EAP의 보안은 광범위한 주제입니다. JBoss EAP와 Camel은 모두 구성, 엔드포인트 및 페이로드를 보호하는 표준화된 방법을 잘 문서화하고 있습니다.

7.1. HAWTIO SECURITY

Hawtio 콘솔 보안은 다음 단계를 통해 수행할 수 있습니다.

1. standalone.xml에 시스템 속성 추가

```
<system-properties>
    <property name="hawtio.authenticationEnabled" value="true" />
    <property name="hawtio.realm" value="hawtio-domain" />
</system-properties>
```

2. 보안 하위 시스템 내에서 Hawtio의 보안 영역 추가

```
<security-domain name="hawtio-domain" cache-type="default">
    <authentication>
        <login-module code="RealmDirect" flag="required">
            <module-option name="realm" value="ManagementRealm"/>
        </login-module>
    </authentication>
</security-domain>
```

3. 관리 사용자 구성

```
$JBOSS_HOME/bin/add-user.sh -u someuser -p s3cret
```

http://localhost:8080/hawtio 로 이동하여 관리 사용자에 대해 구성된 인증 정보로 인증합니다.

7.2. CRYOSTAT-RS 보안

다음 주제에서는 Cryostat-RS 엔드포인트를 보호하는 방법을 설명합니다.

- WildFly HTTP 기본 인증

- 보안 192.0.2.s & SSL
- EventListener 보안

7.3. CRYOSTAT-WS 보안

다음 주제에서는 Cryostat-WS 엔드포인트를 보호하는 방법을 설명합니다.

- WildFly HTTP 기본 인증
- WS-Security
- CXF 보안
- 보안 192.0.2.s & SSL
- EventListener 보안

7.4. JMS 보안

다음 주제는 JMS 엔드포인트를 보호하는 방법을 설명합니다.

- ActiveMQ Artemis 보안 문서
- ActiveMQ Artemis 주소 및 JMS 대상에 대한 보안 설정
- ActiveMQ Artemis 보안 도메인 구성
- ActiveMQ Security

또한 Camel의 경로 정책 개념을 사용하여 JBoss EAP 보안 시스템과 통합할 수 있습니다.

7.5. 경로 정책

Camel은 JBoss EAP 보안 시스템과의 통합에 사용할 수 있는 RoutePolicies 의 개념을 지원합니다. 현재 보안 통합을 위해 지원되는 두 가지 시나리오가 있습니다.

7.5.1. Camel에서 Jarkarta EE 호출

제공된 경로가 보안 Jarkarta EE 구성 요소를 호출할 때 클라이언트 역할을 하며 호출과 관련된 적절한 자격 증명을 제공해야 합니다.

다음과 같이 ClientAuthorizationPolicy 를 사용하여 경로를 분리할 수 있습니다.

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .policy(new ClientAuthorizationPolicy())
      .to("ejb:java:module/AnnotatedSLSB?method=doSelected");
   }
});
```

이는 camel 메시지 처리의 일부로 인증 및 권한 부여를 수행하지 않습니다. 대신 Camel Exchange와 함께 제공되는 인증 정보를 OpenSCAP3 계층 호출과 연결합니다.

메시지 소비자를 호출하는 클라이언트는 다음과 같이 AUTHENTICATION 헤더에 적절한 자격 증명을 제공해야 합니다.

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION,
subject, String.class);
```

Jarkarta EE 계층에서 인증 및 권한 부여가 수행됩니다.

7.5.2. Camel 경로 보안

Camel 경로를 보호하려면 DomainAuthorizationPolicy 를 경로와 연결할 수 있습니다. 이 정책은 지정된 보안 도메인과 "Role2"에 대한 권한 부여에 대해 성공적으로 인증해야 합니다.

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .policy(new DomainAuthorizationPolicy().roles("Role2"))
      .transform(body().prepend("Hello "));
   }
});
camelctx.start();
```

또한 메시지 소비자를 호출하는 클라이언트는 다음과 같이 AUTHENTICATION 헤더에 적절한 자격 증명을 제공해야 합니다.

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION,
subject, String.class);
```

7.6. CXF CRYOSTAT-WS 빠른 시작 배포

이 예에서는 Red Hat Fuse on EAP에서 Red Hat Fuse와 함께 camel-cxf 구성 요소를 사용하여 Elytron Security Domain에서 보안된 Cryostat-WS 웹 서비스를 생성하고 사용하는 방법을 보여줍니다. Elytron은 EAP 7.1 이후 사용할 수 있는 새로운 보안 프레임워크입니다. 이 빠른 시작에서 Camel 경로는 직접 끝점에서 메시지 페이로드를 가져와 CXF 생산자 엔드포인트에 전달합니다. 생산자는 페이로드를 사용하여 BASIC HTTP 인증에 의해 보안되는 CXF Cryostat-WS 웹 서비스에 인수를 전달합니다.

사전 요구 사항

- Maven이 설치 및 구성되었는지 확인합니다.
- Red Hat Fuse가 설치된 애플리케이션 서버가 설치되어 구성되었는지 확인합니다.

절차

1. 애플리케이션 서버 설치의 루트 디렉터리를 가리키도록 JBOSS_HOME 환경 변수를 설정합니다.

- **Linux의 경우**

```
export JBOSS_HOME=...
```

- **Windows의 경우:**

```
set JBOSS_HOME=...
```

2. **add-user 스크립트를 사용하여 새 서버 애플리케이션 사용자 및 그룹을 생성합니다.**

- **Linux의 경우**

```
${JBOSS_HOME}/bin/add-user.sh -a -u testUser -p testPassword1+ -g testRole
```

- **Windows의 경우:**

```
%JBOSS_HOME%\bin\add-user.bat -a -u testUser -p testPassword1+ -g testRole
```

3. **독립 실행형 모드에서 애플리케이션 서버를 시작합니다.**

- **Linux의 경우**

```
${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml
```

- **Windows의 경우:**

```
%JBOSS_HOME%\bin\standalone.bat -c standalone-full.xml
```

이 프로젝트의 webapp/WEB-INF 디렉터리에 있는 jboss-web-xml 및 web.xml 파일은 애플리케이션 보안 도메인, 보안 역할 및 제약 조건을 설정합니다.

4. **프로젝트를 빌드하고 배포합니다.**

```
mvn install -Pdeploy
```

또한 이 명령은 보안 도메인과 몇 가지 다른 관리 오브젝트를 생성하는 **CLI** 스크립트 **configure-basic-security.cli** 를 호출합니다.

5. http://localhost:8080/example-camel-cxf-jaxws-secure/.

Send A '라는 제목의 페이지가 표시됩니다. 이 **UI**를 사용하면 이미 시작한 테스트 인사말 웹 서비스와 상호 작용할 수 있습니다. **WSDL** 서비스는 http://localhost:8080/webservices/greeting-security-basic?wsdl 에서 사용할 수 있습니다.

이라는 단일 서비스 작업이 있습니다. 이 작업은 **message** 및 **name** 이라는 두 개의 **String** 매개 변수를 사용합니다. 웹 서비스를 호출하면 이러한 값이 함께 연결된 응답이 반환됩니다.

Camel 보안 **CXF Cryostat-WS** 빠른 시작 테스트

1. http://localhost:8080/example-camel-cxf-jaxws-secure/.

2. **Send A(A)** 웹 양식에서 메시지와 이름을 텍스트 필드에 입력한 다음 보내기 버튼을 누릅니다.

입력한 정보는 **UI**에 인사말으로 표시됩니다. **CamelCxfWsServlet** 은 웹 **UI**에서 **POST** 요청을 처리합니다. 매개 변수 값에서 메시지와 이름을 검색하고 오브젝트 배열을 구성합니다. 이 오브젝트 배열은 **direct:start** 끝점으로 전송되는 메시지 페이로드입니다. **ProducerTemplate** 은 메시지 페이로드를 **Camel**에 전송합니다. **direct:start** 엔드포인트는 오브젝트 배열을 **cxf:bean** 웹 서비스 생산자에 전달합니다. 웹 서비스 응답은 **CamelCxfWsServlet** 에서 웹 **UI**에 인사말을 표시하는 데 사용됩니다. **src/main/webapp/WEB-INF/cxfws-security-camel-context.xml** 파일에서 전체 **Camel** 경로를 확인할 수 있습니다.

퀵스타트 배포 취소

1. 다음 명령을 실행하여 빠른 시작 배포를 취소합니다.

```
mvn clean -Pdeploy
```