Red Hat Fuse 7.6

OpenShift 기반 Fuse 가이드

OpenShift에서 Red Hat Fuse를 사용하여 설치 및 개발

Last Updated: 2023-12-21

Red Hat Fuse 7.6 OpenShift 기반 Fuse 가이드

OpenShift에서 Red Hat Fuse를 사용하여 설치 및 개발

법적 공지

초록

OpenShift에서 Fuse 사용 가이드

차례

머리말

OpenShift의 Red Hat Fuse를 사용하면 OpenShift Container Platform에 Fuse 애플리케이션을 배포할 수 있습니다.

1장. 사전 준비 사항

릴리스 노트

이 릴리스에 대한 중요한 정보는 릴리스 노트 에서 참조하십시오.

버전 호환성 및 지원

버전 호환성 및 지원에 대한 자세한 내용은 Red Hat JBoss Fuse 지원 구성 페이지를 참조하십시오.

Windows O/S 지원

OpenShift의 Fuse용 개발자 툴링(**oc** 클라이언트 및 컨테이너 개발 키트)은 Windows O/S에서 완전히 지원됩니다. Linux 명령줄 구문에 표시된 예제는 Windows O/S에서도 작동할 수 있으며 Windows 명령줄 구문을 따르도록 적절하게 수정되는 경우입니다.

1.1. 비교: OPENSHIFT에서 FUSE 독립 실행형 및 FUSE

몇 가지 주요 기능 차이점이 있습니다.

- OpenShift에서 Fuse를 사용한 애플리케이션 배포는 Docker 이미지 내에 패키지된 애플리케이션 및 모든 필수 런타임 구성 요소로 구성됩니다. 애플리케이션은 Fuse Standalone과 마찬가지로 런타임에 배포되지 않으며, 애플리케이션 이미지 자체는 OpenShift를 통해 배포 및 관리되는 완전한 런타임 환경입니다.

- OpenShift 환경에서 패치는 Fuse Standalone과 다릅니다. 각 애플리케이션 이미지는 완전한 런타임 환경이므로 마찬가지입니다. 패치를 적용하기 위해 애플리케이션 이미지가 다시 빌드되어 OpenShift 내에서 재배포됩니다. 핵심 OpenShift 관리 기능을 사용하면 롤링 업그레이드 및 병렬 배포를 통해 업그레이드 중에 애플리케이션의 가용성을 유지할 수 있습니다.

- Fuse에서 Fabric에서 제공하는 프로비저닝 및 클러스터링 기능은 Kubernetes 및 OpenShift에서 동등한 기능으로 교체되었습니다. OpenShift가 애플리케이션을 배포하고 확장하는 과정의 일부로 이 작업을 자동으로 수행하므로 개별 하위 컨테이너를 만들거나 구성할 필요가 없습니다.

- Fabric 엔드포인트는 OpenShift 환경 내에서 사용되지 않습니다. 대신 Kubernetes 서비스를 사용해야 합니다.

- 메시징 서비스는 A-MQ for OpenShift 이미지를 사용하여 생성 및 관리되며 Karaf 컨테이너에는 직접 *포함되지 않습니다*. OpenShift의 Fuse는 Kubernetes를 통해 OpenShift의 메시징 서비스에 원활하게 연결할 수 있도록 향상된 버전의 camel-amq 구성 요소를 제공합니다.

- Karaf 쉘을 사용하여 Karaf 인스턴스를 실행하는 실시간 업데이트는 애플리케이션 컨테이너를 다시 시작하거나 확장하면 업데이트가 유지되지 않으므로 권장되지 않습니다. 이는 불변 아키텍처의 기본이며 OpenShift 내에서 확장성 및 유연성을 달성하는 데 필수적입니다.

- Red Hat Fuse 구성 요소에 직접 연결된 Maven 종속 항목은 Red Hat에서 지원합니다. 사용자가 도입한 타사 Maven 종속 항목은 지원되지 않습니다.

- SSH 에이전트는 Apache Karaf micro-container에 포함되어 있지 않으므로 bin/client 콘솔 클라이언트를 사용하여 연결할 수 없습니다.

- OpenShift 애플리케이션의 Fuse 내의 프로토콜 호환성 및 Camel 구성 요소: HTTP 기반이 아닌 통신은 TLS 및 SNI를 사용하여 OpenShift 외부에서 Fuse 서비스(Camel 소비자 끝점)로 라우팅할 수 있어야 합니다.

2장. 관리자 시작하기

OpenShift 관리자인 경우 다음을 통해 OpenShift 배포에 대해 OpenShift 클러스터를 준비할 수 있습니다.

1. Red Hat Container Registry에 대한 인증 구성.
2. OpenShift 이미지 및 템플릿에 Fuse 설치.

2.1. RED HAT CONTAINER REGISTRY 인증 구성

OpenShift 이미지 스트림 및 템플릿에서 Red Hat Fuse를 가져오고 사용하려면 Red Hat 컨테이너 레지스트리에 대한 인증을 구성해야 합니다.

절차

1. 관리자로 OpenShift Server에 로그인합니다.

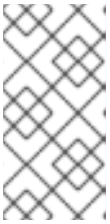
```
oc login -u system:admin
```

2. 이미지 스트림을 설치하려는 OpenShift 프로젝트에 로그인합니다. OpenShift 이미지 스트림에서 Fuse 프로젝트를 사용하는 것이 좋습니다.

```
oc project openshift
```

3. **Red Hat Customer Portal 계정 또는 Red Hat Developer Program 계정 자격 증명을 사용하여 docker-registry 시크릿을 생성합니다. & lt;pull_secret_name >을 생성하려는 보안 이름으로 바꿉니다.**

```
oc create secret docker-registry <pull_secret_name> \
  --docker-server=registry.redhat.io \
  --docker-username=CUSTOMER_PORTAL_USERNAME \
  --docker-password=CUSTOMER_PORTAL_PASSWORD \
  --docker-email=EMAIL_ADDRESS
```



참고

이미지 스트림이 있고 registry.redhat.io를 사용하는 모든 네임스페이스에 docker-registry 시크릿을 생성해야 합니다.

4. **Pod의 이미지를 가져오는 데 시크릿을 사용하려면 서비스 계정에 보안을 추가합니다. 서비스 계정 이름은 서비스 계정 Pod에서 사용하는 이름과 일치해야 합니다. 다음 예제에서는 default 서비스 계정을 사용합니다.**

```
oc secrets link default <pull_secret_name> --for=pull
```

5. 빌드 이미지를 푸시하고 가져오는 데 시크릿을 사용하려면 **Pod** 내부에서 보안을 마운트할 수 있어야 합니다. 시크릿을 마운트하려면 다음 명령을 사용합니다.

```
oc secrets link builder <pull_secret_name>
```

Red Hat 계정 사용자 이름과 암호를 사용하여 시크릿을 생성하지 않으려면 레지스트리 서비스 계정을 사용하여 인증 토큰을 생성해야 합니다.

자세한 내용은 다음을 참조하십시오.

- **Red Hat Container** 레지스트리 인증
- **Red Hat Registry** 액세스 및 구성

2.2. OPENSHIFT 4.X 서버에 FUSE 이미지 스트림 및 템플릿 설치

OpenShift Container Platform 4.1은 **OpenShift** 네임스페이스에서 작동하는 **Samples Operator**를 사용하여 **RHEL(Red Hat Enterprise Linux)** 기반 **OpenShift Container Platform** 이미지 스트림 및 템플릿을 설치하고 업데이트합니다. **OpenShift** 이미지 스트림 및 템플릿에 **Fuse**를 설치하려면 다음을 수행합니다.

- **Samples Operator** 재구성
- **Fuse** 이미지 스트림과 템플릿을 **Skipped Imagestreams** 및 **Skipped Templates** 필드에 추가합니다.
  - 건너뛴 이미지 스트림: **Samples Operator** 인벤토리에 있지만 클러스터 관리자가 **Operator**에서 무시하거나 관리하지 않도록 하려는 이미지 스트림입니다.
  - 건너뛰기된 템플릿: **Samples Operator** 인벤토리에 있지만 클러스터 관리자가 **Operator**에서 무시하거나 관리하지 않도록 하려는 템플릿입니다.

사전 요구 사항

- **OpenShift Server**에 액세스할 수 있습니다.
- **Red Hat Container Registry**에 대한 인증을 구성했습니다.
- 선택적으로 설치한 후 **OpenShift** 대시보드에 **Fuse** 템플릿을 표시하려면 먼저 **OpenShift** 문서(**https://docs.openshift.com/container-platform/4.1/applications/service_brokers/installing-service-catalog.html)**에 설명된 대로 서비스 카탈로그 및 템플릿 서비스 브로커를 설치해야 합니다.**https://docs.openshift.com/container-platform/4.1/applications/service_brokers/installing-service-catalog.html**

절차

1. **OpenShift 4 Server**를 시작합니다.
2. 관리자로 **OpenShift** 서버에 로그인합니다.

```
oc login -u system:admin
```

3. **docker-registry** 시크릿을 생성한 프로젝트를 사용하고 있는지 확인합니다.

```
oc project openshift
```

4. **Samples Operator**의 현재 구성을 확인합니다.

```
oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator -o yaml
```

5. 추가된 **fuse** 템플릿 및 이미지 스트림을 무시하도록 **Samples Operator**를 구성합니다.

```
oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

6. **Fuse** 이미지 스트림 **Skipped Imagestreams** 섹션을 추가하고 **Fuse** 및 **Spring Boot 2** 템플릿을 **Skipped Templates** 섹션에 추가합니다.

```
[...]
spec:
  architectures:
```

```
  - x86_64
  managementState: Managed
  skippedImagestreams:
  - fis-java-openshift
  - fis-karaf-openshift
  - fuse7-console
  - fuse7-eap-openshift
  - fuse7-java-openshift
  - fuse7-karaf-openshift
  - jboss-fuse70-console
  - jboss-fuse70-eap-openshift
  - jboss-fuse70-java-openshift
  - jboss-fuse70-karaf-openshift
  - fuse-apicurito-generator
  - apicurito-ui
  skippedTemplates:
  - s2i-fuse76-eap-camel-amq
  - s2i-fuse76-eap-camel-cdi
  - s2i-fuse76-eap-camel-cxf-jaxrs
  - s2i-fuse76-eap-camel-cxf-jaxws
  - s2i-fuse76-eap-camel-jpa
  - s2i-fuse76-karaf-camel-amq
  - s2i-fuse76-karaf-camel-log
  - s2i-fuse76-karaf-camel-rest-sql
  - s2i-fuse76-karaf-cxf-rest
  - s2i-fuse76-spring-boot-camel
  - s2i-fuse76-spring-boot-camel-amq
  - s2i-fuse76-spring-boot-camel-config
  - s2i-fuse76-spring-boot-camel-drools
  - s2i-fuse76-spring-boot-camel-infinispan
  - s2i-fuse76-spring-boot-camel-rest-sql
  - s2i-fuse76-spring-boot-camel-rest-3scale
  - s2i-fuse76-spring-boot-camel-xa
  - s2i-fuse76-spring-boot-camel-xml
  - s2i-fuse76-spring-boot-cxf-jaxrs
  - s2i-fuse76-spring-boot-2-camel-amq
  - s2i-fuse76-spring-boot-2-camel-config
  - s2i-fuse76-spring-boot-2-camel-drools
  - s2i-fuse76-spring-boot-2-camel-infinispan
  - s2i-fuse76-spring-boot-2-camel-rest-3scale
  - s2i-fuse76-spring-boot-2-camel-rest-sql
  - s2i-fuse76-spring-boot-2-camel
  - s2i-fuse76-spring-boot-2-camel-xa
  - s2i-fuse76-spring-boot-2-camel-xml
  - s2i-fuse76-spring-boot-2-cxf-jaxrs
  - s2i-fuse76-spring-boot-2-cxf-jaxws
  - s2i-fuse76-spring-boot-2-cxf-jaxrs-xml
  - s2i-fuse76-spring-boot-2-cxf-jaxws-xml
```

7. **OpenShift 이미지 스트림에 Fuse를 설치합니다.**

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003
```

```
oc create -n openshift -f ${BASEURL}/fis-image-streams.json
```

참고

Error from server(AlreadyExists): imagestreams.image.openshift.io <imagestreamname >이라는 메시지와 함께 오류가 표시되면 다음 명령을 사용하여 기존 이미지 스트림을 최신 상태로 교체합니다.

```
oc replace --force -n openshift -f ${BASEURL}/fis-image-streams.json
```

8. **OpenShift 빠른 시작 템플릿에 Fuse를 설치합니다.**

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 eap-camel-jpa-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json \
 spring-boot-camel-amq-template.json \
 spring-boot-camel-config-template.json \
 spring-boot-camel-drools-template.json \
 spring-boot-camel-infinispan-template.json \
 spring-boot-camel-rest-sql-template.json \
 spring-boot-camel-rest-3scale-template.json \
 spring-boot-camel-template.json \
 spring-boot-camel-xa-template.json \
 spring-boot-camel-xml-template.json \
 spring-boot-cxf-jaxrs-template.json \
 spring-boot-cxf-jaxws-template.json ;
 do
 oc create -n openshift -f \
 https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-
2.1.0.fuse-760043-redhat-00003/quickstarts/${template}
 done
```

9. **Spring Boot 2 빠른 시작 템플릿을 설치합니다.**

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
```

```
spring-boot-2-camel-xml-template.json \
spring-boot-2-cxf-jaxrs-template.json \
spring-boot-2-cxf-jaxws-template.json \
spring-boot-2-cxf-jaxrs-xml-template.json \
spring-boot-2-cxf-jaxws-xml-template.json ;
do oc create -n openshift -f \
https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-sb2-760039-redhat-00001/quickstarts/${template}
done
```

10. *(선택 사항)* **OpenShift 템플릿에 설치된 Fuse를 확인합니다.**

```
oc get template -n openshift
```

2.2.1. OpenShift 4.x에서 Fuse 콘솔 설정

OpenShift 4.x에서 Fuse 콘솔을 설정하려면 보안, 설치 및 배포가 필요합니다.

먼저 2.2.1.1절. "OpenShift 4.x에서 Fuse Console을 보호하기 위한 인증서 생성" 에 설명된 대로 Fuse 콘솔을 보호할 수 있도록 클라이언트 인증서를 생성해야 합니다.

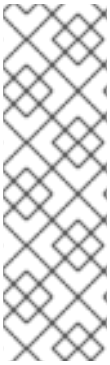
클라이언트 인증서를 생성한 후 Fuse Console을 설치하고 배포하기 위한 다음과 같은 옵션이 있습니다.

- **2.2.1.2절. "OperatorHub를 사용하여 OpenShift 4.x에 Fuse Console 설치 및 배포"**

  Fuse Console Operator를 사용하여 특정 네임스페이스의 Fuse 애플리케이션에 액세스할 수 있도록 Fuse Console을 설치하고 배포할 수 있습니다.

- **2.2.1.3절. "명령줄을 사용하여 OpenShift 4.x에 Fuse Console 설치 및 배포"**

  명령줄과 Fuse Console 템플릿 중 하나를 사용하여 Fuse Console을 설치하고 배포하여 OpenShift 클러스터 또는 특정 네임스페이스에 있는 여러 네임스페이스의 Fuse 애플리케이션에 액세스할 수 있습니다.

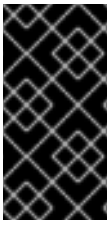


참고

- Fuse Console의 사용자 관리는 OpenShift에서 처리합니다.
- 배포 후 Fuse Console에 액세스하는 사용자의 경우 OpenShift에서 Fuse를 사용할 수 없습니다.

2.2.1.1. OpenShift 4.x에서 Fuse Console을 보호하기 위한 인증서 생성

OpenShift 4.x에서는 Fuse Console 프록시와 Jolokia 에이전트 간에 안전하게 연결을 유지하려면 Fuse Console을 배포하기 전에 클라이언트 인증서를 생성해야 합니다. 서비스 서명 인증 기관 개인 키를 사용하여 클라이언트 인증서에 서명해야 합니다.



중요

각 OpenShift 클러스터에 대해 별도의 클라이언트 인증서를 생성하고 서명해야 합니다. 두 개 이상의 클러스터에 동일한 인증서를 사용하지 마십시오.

사전 요구 사항

- OpenShift 클러스터에 대한 클러스터 관리자 액세스 권한이 있어야 합니다.
- 두 개 이상의 OpenShift 클러스터에 대한 인증서를 생성하고 이전에 현재 디렉터리에 다른 클러스터에 대한 인증서를 생성한 경우 다음 중 하나를 수행하여 현재 클러스터에 대한 다른 인증서를 생성하십시오.
  - 현재 디렉터리에서 기존 인증서 파일(예: ca.crt,ca.key, ca.srl)을 삭제합니다.
  - 다른 작업 디렉터리로 변경합니다. 예를 들어 현재 작업 디렉터리 이름이 cluster1 인 경우 새 cluster2 디렉터리를 생성하고 작업 디렉터리를 해당 디렉터리로 변경합니다.

```
mkdir ../cluster2
```

```
cd ../cluster2
```

절차

1. 클러스터 관리자 액세스 권한이 있는 사용자로 **OpenShift**에 로그인합니다.

```
oc login -u <user_with_cluster_admin_role>
```

2. 다음 명령을 실행하여 서비스 서명 인증 기관 키를 검색합니다.

- 인증서를 검색하려면 다음을 수행합니다.

```
oc get secrets/signing-key -n openshift-service-ca -o "jsonpath={.data['tls\.crt']}" | base64
--decode > ca.crt
```

- 개인 키를 검색하려면 다음을 수행합니다.

```
oc get secrets/signing-key -n openshift-service-ca -o "jsonpath={.data['tls\.key']}" |
base64 --decode > ca.key
```

3. **easyrsa,openssl** 또는 **cfssl** 을 사용하여 **Kubernetes** 인증서 관리에 설명된 대로 클라이언트 인증서를 생성합니다.

다음은 **openssl**을 사용하는 예제 명령입니다.

a. 개인 키를 생성합니다.

```
openssl genrsa -out server.key 2048
```

b. **CSR** 구성 파일을 작성합니다.

```
cat <<EOT >> csr.conf
 [ req ]
 default_bits = 2048
 prompt = no
 default_md = sha256
 distinguished_name = dn

 [ dn ]
 CN = fuse-console.fuse.svc

 [ v3_ext ]
 authorityKeyIdentifier=keyid,issuer:always
```

```
  keyUsage=keyEncipherment,dataEncipherment,digitalSignature
  extendedKeyUsage=serverAuth,clientAuth
EOT
```

c. **CSR을 생성합니다.**

```
openssl req -new -key server.key -out server.csr -config csr.conf
```

d. **서명된 인증서를 발급합니다.**

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt
-days 10000 -extensions v3_ext -extfile csr.conf
```

다음 단계

Fuse Console 설치 방법에 따라 다음 섹션에 설명된 대로 Fuse Console의 시크릿을 생성하려면 이 인증서가 필요합니다.

- Fuse Console Operator 사용
- 명령줄 사용

2.2.1.2. OperatorHub를 사용하여 OpenShift 4.x에 Fuse Console 설치 및 배포

OpenShift 4.x에 Fuse Console을 설치하려면 OpenShift OperatorHub에 제공된 Fuse Console Operator를 사용할 수 있습니다. Fuse 콘솔을 배포하려면 설치된 Operator의 인스턴스를 만듭니다.

사전 요구 사항

- **OpenShift 클러스터에 대한 클러스터 관리자 액세스 권한이 있어야 합니다.**
- **OpenShift 4.x에서 Fuse 콘솔을 보호하기 위해 인증서 생성에 설명된 대로 Fuse Console의 클라이언트 인증서를 생성 했습니다.**

절차

Fuse 콘솔을 설치하고 배포하려면 다음을 수행합니다.

1. 웹 브라우저에서 클러스터 관리자 액세스 권한이 있는 사용자로 **OpenShift** 콘솔에 로그인합니다.

2. **홈 > 프로젝트**를 선택한 다음 **Fuse** 콘솔을 추가할 프로젝트를 선택합니다.

3. **Operators** 를 클릭한 다음 **OperatorHub** 를 클릭합니다.

4. 검색 필드 창에서 **Fuse Console** 을 입력하여 **Operator** 목록을 필터링합니다.

5. **Fuse Console Operator** 를 클릭합니다.

6. **Fuse Console Operator** 설치 창에서 설치를 클릭합니다.

   Create Operator Subscription 양식이 열립니다.

   - 설치 모드 의 경우 **Fuse Console Operator**를 특정 네임스페이스(현재 **OpenShift** 프로젝트)에 설치합니다.

     그런 다음 **Operator**를 설치한 후 **Fuse Console**을 배포하여 클러스터의 모든 네임스페이스에서 애플리케이션을 모니터링하거나 **Fuse Console Operator**가 설치된 네임스페이스에서만 애플리케이션을 모니터링하도록 선택할 수 있습니다.

   - 승인 전략 의 경우 자동 또는 수동 을 선택하여 **OpenShift**에서 **Fuse Console Operator**에 대한 업데이트를 처리하는 방법을 구성할 수 있습니다.

     - 자동 업데이트를 선택하면 새 버전의 **Fuse Console Operator**가 사용 가능할 때 **OLM(Operator Lifecycle Manager)**은 개입 없이 **Fuse Console**의 실행 중인 인스턴스를 자동으로 업그레이드합니다.

     - 수동 업데이트를 선택하면 최신 버전의 **Operator**가 사용 가능할 때 **OLM**에서 업데이트 요청을 생성합니다. 클러스터 관리자는 **Fuse Console Operator**가 새 버전으로 업데이트되도록 해당 업데이트 요청을 수동으로 승인해야 합니다.

7

7.

Subscribe를 클릭합니다.

OpenShift는 현재 네임스페이스에 **Fuse Console Operator**를 설치합니다.

8.

설치를 확인하려면 **Operator**를 클릭한 다음 **Installed Operators** 를 클릭합니다. **Operator** 목록에서 **Fuse Console**을 볼 수 있습니다.

9.

터미널 창에서 **OpenShift 4.x에서 Fuse Console 보안에서 생성한 인증서를** 사용하여 시크릿을 생성하고, **APP_NAME** 이 **Fuse Console** 배포 이름(예: **fuse-console**)을 사용하여 **Fuse** 콘솔에 마운트합니다.

```
oc create secret tls APP_NAME-tls-proxying --cert server.crt --key server.key
```

예를 들어 **Fuse Console** 애플리케이션의 이름이 **fuse-console** 인 경우 다음 명령을 입력합니다.

```
oc create secret tls fuse-console-tls-proxying --cert server.crt --key server.key
```

성공하면 이 명령은 보안이 생성되었는지 확인하는 응답을 반환합니다. 예를 들면 다음과 같습니다.

```
secret/fuse-console-operator-tls-proxying created
```

10.

OpenShift 웹 콘솔을 사용하여 **Fuse** 콘솔을 배포하려면 다음을 수행합니다.

a.

설치된 **Operator** 목록에서 이름 열에서 **Fuse Console** 을 클릭합니다.

b.

제공된 **API** 아래의 개요 페이지에서 인스턴스 생성을 클릭합니다. 새 **CRD(Custom Resource Definition)** 파일이 열립니다.

기본적으로 **Fuse Console**은 현재 네임스페이스에 배포됩니다.

c.

Fuse Console을 배포하여 현재 네임스페이스 내에서 애플리케이션을 검색하고 관리하려면 다음 단계로 건너뜁니다.

선택적으로 클러스터(및 인증된 사용자)에 있는 모든 네임스페이스에서 애플리케이션

을 검색하고 관리하려면 **spec.type** 필드의 값을 네임스페이스 에서 클러스터로 변경하여 **CRD** 파일을 편집합니다.

d. **생성**을 클릭합니다.

Fuse Console CRD 페이지가 열리고 새 **Fuse Console** 서비스가 표시됩니다.

11. **Fuse** 콘솔을 열려면 다음을 수행합니다.

a. **Fuse** 콘솔 **URL**을 가져옵니다.

- **OpenShift** 웹 콘솔 내에서 **네트워킹 > 경로**를 선택합니다.
- 명령줄에서 **oc get routes** 명령을 입력합니다.

b. 웹 브라우저에서 **Fuse Console URL**을 연 다음 로그인합니다.

필요한 권한이 나열된 브라우저에서 권한 부여 페이지가 열립니다.

c. 선택한 권한 허용을 클릭합니다.

브라우저에서 **Fuse Console**이 열리고 액세스할 수 있는 권한이 있는 **Fuse** 애플리케이션 포드가 표시됩니다.

12. 확인할 애플리케이션에 대한 연결을 클릭합니다.

Fuse Console에 애플리케이션이 표시되는 새 브라우저 창이 열립니다.

2.2.1.3. 명령줄을 사용하여 OpenShift 4.x에 Fuse Console 설치 및 배포

OpenShift 4.x에서는 명령줄에서 **Fuse** 콘솔을 설치하고 배포할 다음 배포 옵션 중 하나를 선택할 수 있습니다.

- 클러스터 - Fuse Console은 OpenShift 클러스터의 여러 네임스페이스(프로젝트)에 배포된 Fuse 애플리케이션을 검색하고 연결할 수 있습니다. 이 템플릿을 배포하려면 OpenShift 클러스터에 대한 관리자 역할이 있어야 합니다.

- 네임스페이스 - Fuse Console은 특정 OpenShift 프로젝트(네임스페이스)에 액세스할 수 있습니다. 이 템플릿을 배포하려면 OpenShift 프로젝트에 대한 관리자 역할이 있어야 합니다.

Fuse Console 템플릿의 매개변수 목록을 보려면 다음 OpenShift 명령을 실행합니다.

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fuse-console-namespace-os4.json
```

사전 요구 사항

- Fuse 콘솔을 설치하고 배포하기 전에 OpenShift 4.x에서 Fuse Console을 보호하기 위해 인증서 생성에 설명된 대로 서비스 서명 인증 기관으로 서명된 클라이언트 인증서를 생성해야 합니다.

- OpenShift 클러스터에 대한 클러스터 관리자 역할이 있어야 합니다.

- OpenShift 4.x 서버에 Fuse 이미지 스트림 설치 및 템플릿에 설명된 대로 Fuse 콘솔 이미지 스트림(다른 Fuse 이미지 스트림과 함께)이 설치됩니다.

절차

1. 다음 명령을 사용하여 모든 템플릿 목록을 검색하여 Fuse Console 이미지 스트림이 설치되었는지 확인합니다.

```
oc get template -n openshift
```

2. 선택적으로 이미 설치된 이미지 스트림을 새 릴리스 태그로 업데이트하려면 다음 명령을 사용하여 Fuse Console 이미지를 openshift 네임스페이스로 가져옵니다.

```
oc import-image fuse7/fuse7-console:1.6 --from=registry.redhat.io/fuse7/fuse-console:1.6 --confirm -n openshift
```

3. 다음 명령을 실행하여 **Fuse Console APP_NAME** 값을 가져옵니다.

- 클러스터 템플릿의 경우:

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fuse-console-cluster-os4.json
```

- 네임스페이스 템플릿의 경우:

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fuse-console-namespace-os4.json
```

4. **OpenShift 4.x에서 Fuse Console 보안에서 생성한 인증서에서** 다음 명령을 사용하여 시크릿을 생성하고 **Fuse Console**에 마운트합니다(여기서 **APP_NAME** 은 **Fuse Console** 애플리케이션의 이름입니다).

```
oc create secret tls APP_NAME-tls-proxying --cert server.crt --key server.key
```

5. 다음 명령을 실행하여 새 애플리케이션을 생성합니다(**$project** 는 **OpenShift** 프로젝트의 이름, **$APP_NAME** 은 위 단계에서 얻은 애플리케이션의 이름이며 **$DOMAIN_NAME** 은 **Fuse Console**에 액세스할 수 있는 호스트 이름입니다.

- 클러스터 템플릿의 경우:

```
oc new-app -n $project -f {templates-base-url}/fuse-console-cluster-os4.json -p ROUTE_HOSTNAME=$APP_NAME.$DOMAIN_NAME -p APP_NAME=$APP_NAME
```

- 네임스페이스 템플릿의 경우:

```
oc new-app -n myproject -f {templates-base-url}/fuse-console-namespace-os4.json
```

6. 다음 명령을 실행하여 **Fuse Console** 배포의 상태 및 **URL**을 가져옵니다.

```
oc status
```

7.

브라우저에서 Fuse 콘솔에 액세스하려면 6단계에서 반환된 URL(예: https://fuse-console.192.168.64.12.nip.io)을 사용합니다.

2.2.1.4. OpenShift 4.x에서 Fuse Console 업그레이드

Red Hat OpenShift 4.x는 Red Hat Fuse Operator를 포함하여 Operator에 대한 업데이트를 처리합니다. 자세한 내용은 Operator OpenShift 설명서 를 참조하십시오.

Operator 업데이트는 애플리케이션 구성 방법에 따라 애플리케이션 업그레이드를 트리거할 수 있습니다.

Fuse Console 애플리케이션의 경우 애플리케이션 사용자 정의 리소스 정의의 .spec.version 필드를 편집하여 애플리케이션으로의 업그레이드를 트리거할 수도 있습니다.

사전 요구 사항

- OpenShift 클러스터 관리자 권한이 있어야 합니다.

절차

Fuse Console 애플리케이션을 업그레이드하려면 다음을 수행합니다.

1. 터미널 창에서 다음 명령을 사용하여 애플리케이션 사용자 정의 리소스 정의의 .spec.version 필드를 변경합니다.

```
oc patch <project-name> <custom-resource-name> --type='merge' -p '{"spec":
{"version":"1.7.1"}}'
```

예를 들면 다음과 같습니다.

```
oc patch myproject example-fuseconsole --type='merge' -p '{"spec":{"version":"1.7.1"}}'
```

2. 애플리케이션 상태가 업데이트되었는지 확인합니다.

```
oc get myproject
```

응답에는 버전 번호를 포함하여 애플리케이션에 대한 정보가 표시됩니다.

```
NAME                AGE   URL                                IMAGE
example-fuseconsole   1m    https://fuseconsole.192.168.64.38.nip.io
docker.io/fuseconsole/online:1.7.1
```

.spec.version 필드의 값을 변경하면 **OpenShift**에서 애플리케이션을 자동으로 재배포합니다.

3. 버전 변경으로 트리거되는 재배포 상태를 확인하려면 다음을 수행합니다.

```
oc rollout status deployment.v1.apps/example-fuseconsole
```

성공적인 배포에는 다음 응답이 표시됩니다.

```
deployment "example-fuseconsole" successfully rolled out
```

2.2.2. OpenShift 4.x에 API Cryostat 설치

Red Hat Fuse on OpenShift는 **REST API**를 설계하는 데 사용할 수 있는 웹 기반 **API Creator** 툴인 **API Creator**를 제공합니다. **API Splunk Operator**는 **OpenShift Container Platform 4.x**에서 **API Builder**의 설치 및 업그레이드를 간소화합니다.

OpenShift 관리자는 **API Creator Operator**를 **OpenShift** 프로젝트(네임스페이스)에 설치합니다. **Operator**가 설치되면 선택한 네임스페이스에서 **Operator**가 실행됩니다. 그러나 **API window**를 **OpenShift** 관리자로 사용하여 서비스로 사용 가능하게 하려면 개발자가 **API Cryostat**의 인스턴스를 생성해야 합니다. **API Cryostat** 서비스는 **API window web console**에 액세스할 수 있는 **URL**을 제공합니다.

사전 요구 사항

- **OpenShift** 클러스터에 대한 관리자 액세스 권한이 있어야 합니다.
- **Red Hat Container Registry**에 대한 인증을 구성했습니다.

절차

1.

1. OpenShift 4.x 서버를 시작합니다.

2. 웹 브라우저에서 브라우저에서 OpenShift 콘솔로 이동합니다. 인증 정보를 사용하여 콘솔에 로그인합니다.

3. Catalog 를 클릭한 다음 OperatorHub 를 클릭합니다.

4. 검색 필드에 API window를 입력합니다.

   오른쪽 패널에 있는 API Builder 카드를 볼 수 있습니다.

5. API desktop을 클릭합니다. API Cryostat Operator 설치 창이 표시됩니다.

6. 설치를 클릭합니다. Create Operator Subscription 양식이 열립니다.

   a. 설치 모드 의 경우 클러스터의 네임스페이스 목록에서 네임스페이스(프로젝트)를 선택합니다.

   b. 승인 전략 의 경우 자동 또는 수동 을 선택하여 OpenShift에서 API Cryostat Operator에 대한 업데이트를 처리하는 방법을 구성합니다.

   - 자동 업데이트를 선택하면 새 버전의 API Builder Operator를 사용할 수 있는 경우 OLM(Operator Lifecycle Manager)은 개입 없이 API Cryostat의 실행 중인 인스턴스를 자동으로 업그레이드합니다.

   - 수동 업데이트를 선택하면 최신 버전의 Operator가 사용 가능할 때 OLM에서 업데이트 요청을 생성합니다. 클러스터 관리자는 API Policy Operator를 새 버전으로 업데이트하려면 해당 업데이트 요청을 수동으로 승인해야 합니다.

7. Subscribe 를 클릭하여 지정된 네임스페이스(프로젝트)에서 API Creator Operator를 사용할 수 있도록 합니다.

8. API window가 프로젝트에 설치되어 있는지 확인하려면 Operators 를 클릭한 다음

Installed Operators 를 클릭하여 목록의 **APIFilter**를 확인합니다.

다음 단계

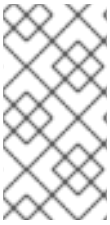
API Builder Operator가 설치되면 **API Cryostat**의 인스턴스를 생성하여 **API Builder**를 **OpenShift** 프로젝트에 서비스로 추가해야 합니다. 이 작업은 다음 두 가지 방법으로 수행할 수 있습니다.

- **OpenShift** 관리자는 **2.2.2.1**절. **"OpenShift 4.x** 프로젝트에 **API Cryostat**를 서비스로 추가" 의 단계를 수행할 수 있습니다.
- **OpenShift** 개발자는 **API** 설계에 설명된 단계를 따를 수 있습니다.

  API Cryostat 서비스는 **API window web console**에 액세스할 수 있는 **URL**을 제공합니다.

2.2.2.1. OpenShift 4.x 프로젝트에 API Cryostat를 서비스로 추가

OpenShift 4.x 프로젝트에 **API Creator Operator**를 설치한 후 **OpenShift** 개발자(또는 **OpenShift** 개발자)에서 **OpenShift** 프로젝트에 서비스로 추가할 수 있습니다. **API Splunk** 서비스는 개발자가 **API users users** 웹 콘솔에 액세스하는 데 사용하는 **URL**을 제공합니다.



참고

OpenShift 개발자가 **OpenShift 4.x** 프로젝트에 **API Creator**를 서비스로 추가하기 위해 수행하는 단계는 **API** 설계를 참조하십시오.

사전 요구 사항

- **OpenShift** 클러스터에 대한 관리자 액세스 권한이 있어야 합니다.
- **API Cryostat Operator**는 현재 **OpenShift** 프로젝트에 설치됩니다.

절차

1. **OpenShift** 웹 콘솔에서 **Operator**를 클릭한 다음 설치된 **Operator** 를 클릭합니다.

2. 이름 열에서 **API Creator**를 클릭합니다.

3. 제공된 **API** 에서 인스턴스 생성을 클릭합니다.

   API Builder 인스턴스에 대한 최소 시작 템플릿이 있는 기본 양식이 열립니다.

4. 기본값을 수락하거나 선택적으로 다음 값을 편집합니다.

   - **size: API Splunk** 인스턴스의 **Pod** 수입니다.

     기본값은 **API Cryostat** 관리와 관련된 논리가 포함된 세 가지 새 리소스 유형의 **Pod 3** 개입니다.

   - **image: APIDesign** 이미지입니다. 이 이미지를 변경하면 클러스터에서 **API Creator Operator**가 자동으로 업그레이드됩니다.

5. 생성 을 클릭하여 새 **apicurito-service** 를 생성합니다. **OpenShift**는 새 **API Creator** 서비스에 대한 **pod,** 서비스 및 기타 구성 요소를 시작합니다.

6. **API Creator** 서비스를 사용할 수 있는지 확인하려면 다음을 수행하십시오.

   a. **Operators** 를 클릭한 다음 **Installed Operators** 를 클릭합니다.

   b. **Provided APIs** 열에서 **Apicurito CRD** 를 클릭합니다.

      Operator 세부 정보 페이지에 **apicurito-service** 가 나열됩니다.

7. **API Cryostat**의 **URL**을 가져오려면 다음을 수행합니다.

   a. 네트워킹 **>** 경로를 클릭합니다.

b. 올바른 프로젝트가 선택되어 있는지 확인합니다.

c. **apicurito-service** 행에서 위치 열에서 **API Builder** 웹 콘솔의 **URL**을 찾습니다.

2.2.2.2. OpenShift 4.x에서 APIDesign 업그레이드

Red Hat OpenShift 4.x는 **Red Hat Fuse Operator**를 포함하여 **Operator**에 대한 업데이트를 처리합니다. 자세한 내용은 **Operator OpenShift** 설명서 를 참조하십시오.

결과적으로 **Operator** 업데이트는 애플리케이션 업그레이드를 트리거할 수 있습니다. 애플리케이션 업그레이드 방법은 애플리케이션 구성 방법에 따라 다릅니다.

API Builder 애플리케이션의 경우 **OpenShift**는 **API Creator Operator**를 업그레이드할 때 클러스터의 모든 **APIDesign** 애플리케이션도 자동으로 업그레이드합니다.

2.3. OPENSHIFT 3.X 서버에 FUSE 이미지 스트림 및 템플릿 설치

Red Hat 컨테이너 레지스트리에 대한 인증을 구성한 후 **OpenShift** 이미지 스트림 및 템플릿에서 **Red Hat Fuse**를 가져와서 사용합니다.

절차

1. **OpenShift Server**를 시작합니다.

2. 관리자로 **OpenShift** 서버에 로그인합니다.

```
oc login -u system:admin
```

3. **docker-registry** 시크릿을 생성한 프로젝트를 사용하고 있는지 확인합니다.

```
oc project openshift
```

4. **OpenShift** 이미지 스트림에 **Fuse**를 설치합니다.

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003

oc create -n openshift -f ${BASEURL}/fis-image-streams.json
```

5. **빠른 시작 템플릿을 설치합니다.**

```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 eap-camel-jpa-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json \
 spring-boot-camel-amq-template.json \
 spring-boot-camel-config-template.json \
 spring-boot-camel-drools-template.json \
 spring-boot-camel-infinispan-template.json \
 spring-boot-camel-rest-sql-template.json \
 spring-boot-camel-rest-3scale-template.json \
 spring-boot-camel-template.json \
 spring-boot-camel-xa-template.json \
 spring-boot-camel-xml-template.json \
 spring-boot-cxf-jaxrs-template.json \
 spring-boot-cxf-jaxws-template.json ;
 do
 oc create -n openshift -f \
 https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/quickstarts/${template}
 done
```

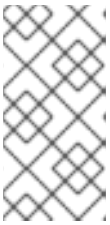
6. **Spring Boot 2 빠른 시작 템플릿을 설치합니다.**

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-rest-sql-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc create -n openshift -f \
```

```
https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-sb2-760039-redhat-00001/quickstarts/${template}
done
```

7. **Fuse Console용 템플릿을 설치합니다.**

```
oc create -n openshift -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fis-console-cluster-template.json
oc create -n openshift -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fis-console-namespace-template.json
```



참고

Fuse Console 배포에 대한 자세한 내용은 OpenShift에 Fuse Console 설정을 참조하십시오.

8. **Apicurito 템플릿을 설치합니다.**

```
oc create -n openshift -f ${BASEURL}/fuse-apicurito.yml
```

9. ***(선택 사항)* OpenShift 이미지 및 템플릿에서 설치된 Fuse를 확인합니다.**

```
oc get template -n openshift
```

2.3.1. OpenShift 3.11에서 Fuse 콘솔 설정

OpenShift 3.11에서는 다음 두 가지 방법으로 Fuse Console을 설정할 수 있습니다.

- 프로젝트에 *중앙 집중식* **Fuse Console** 카탈로그 항목을 추가하여 프로젝트에서 실행 중인 모든 **Fuse** 컨테이너를 모니터링할 수 있습니다.
- 특정 포드에서 실행 중인 단일 **Fuse** 컨테이너를 모니터링할 수 있습니다.

OpenShift 콘솔 또는 명령줄에서 Fuse 콘솔을 배포할 수 있습니다.

참고

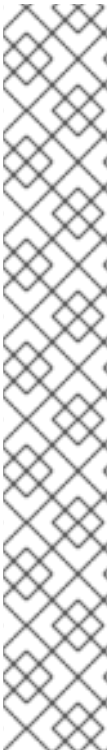
Minishift 또는 CDK 기반 환경에 Fuse 콘솔을 설치하려면 아래 KCS 문서에 설명된 단계를 따르십시오.

- Minishift 또는 CDK 기반 환경에 Fuse Console을 설치하려면 KCS 4998441 을 참조하십시오.
- Jolokia 인증을 비활성화해야 하는 경우 KCS 3988671 에 설명된 해결방법을 참조하십시오.

사전 요구 사항

- OpenShift의 Fuse에 설명된 대로 OpenShift 이미지 스트림에 Fuse 및 Fuse 콘솔 템플릿을 설치합니다.
- OpenShift 3.11의 클러스터 모드의 경우 클러스터 관리자 역할 및 클러스터 모드 템플릿이 필요합니다. 다음 명령을 실행합니다.

```
oc adm policy add-cluster-role-to-user cluster-admin system:serviceaccount:openshift-infra:template-instance-controller
```



참고

- 클러스터 모드 템플릿은 기본적으로 최신 버전의 OpenShift Container Platform에서만 사용할 수 있습니다. OpenShift Online 기본 카탈로그는 제공되지 않습니다.
- Fuse Console 템플릿은 브라우저에서 클러스터 내 서비스로의 보안 엔드 투 엔드 요청이 되도록 기본적으로 엔드 투 엔드 암호화를 구성합니다.
- Fuse Console의 사용자 관리는 OpenShift에서 처리합니다.
- 배포 후 Fuse Console에 액세스하는 사용자의 경우 OpenShift에서 Fuse를 사용할 수 없습니다.

2.3.1.1. OpenShift 3.11 콘솔에서 Fuse 콘솔 배포

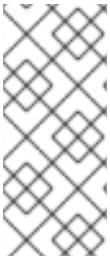
OpenShift 3.11 콘솔의 OpenShift 클러스터에 Fuse Console을 배포하려면 다음 단계를 따르십시오.

절차

1. OpenShift 콘솔에서 기존 프로젝트를 열거나 새 프로젝트를 생성합니다.
2. OpenShift 프로젝트에 Fuse 콘솔을 추가합니다.
   a. Add to Project → Browse Catalog 를 선택합니다.

      현재 프로젝트에 추가할 항목 선택 페이지가 열립니다.
   b. 검색 필드에 Fuse Console 을 입력합니다.

      Red Hat Fuse 7.x 콘솔 과 Red Hat Fuse 7.x 콘솔(cluster) 항목이 검색 결과에 표시됩니다.



참고

Red Hat Fuse Console 항목이 검색 결과로 나타나지 않거나 표시되는 항목이 최신 버전이 아닌 경우 OpenShift Guide의 "OpenShift 서버 전" 섹션에 설명된 대로 수동으로 Fuse Console 템플릿을 설치할 수 있습니다.

   a. Red Hat Fuse Console 항목 중 하나를 클릭합니다.

- **Red Hat Fuse 7.x 콘솔 -** 이 버전의 **Fuse Console**은 현재 **OpenShift** 프로젝트에 배포된 **Fuse** 애플리케이션을 검색하고 연결합니다.

- **Red Hat Fuse 7.x Console(**클러스터**) -** 이 버전의 **Fuse Console**은 **OpenShift** 클러스터의 여러 프로젝트에 배포된 **Fuse** 애플리케이션을 검색하고 연결할 수 있습니다.

b. **Red Hat Fuse Console** 마법사에서 다음을 클릭합니다. 마법사의 구성 페이지가 열립니다.

선택적으로 구성 매개변수의 기본값을 변경할 수 있습니다.

1. 생성을 클릭합니다.

마법사의 결과 페이지에는 **Red Hat Fuse Console**이 생성되었음을 나타냅니다.

2. **Continue to the project overview** 링크를 클릭하여 **Fuse Console** 애플리케이션이 프로젝트에 추가되었는지 확인합니다.

3. **Fuse** 콘솔을 열려면 제공된 **URL** 링크를 클릭한 다음 로그인합니다.

필요한 권한이 나열된 브라우저에서 권한 부여 페이지가 열립니다.

4. 선택한 권한 허용을 클릭합니다.

브라우저에서 **Fuse Console**이 열리고 프로젝트에서 실행 중인 **Fuse Pod**가 표시됩니다.

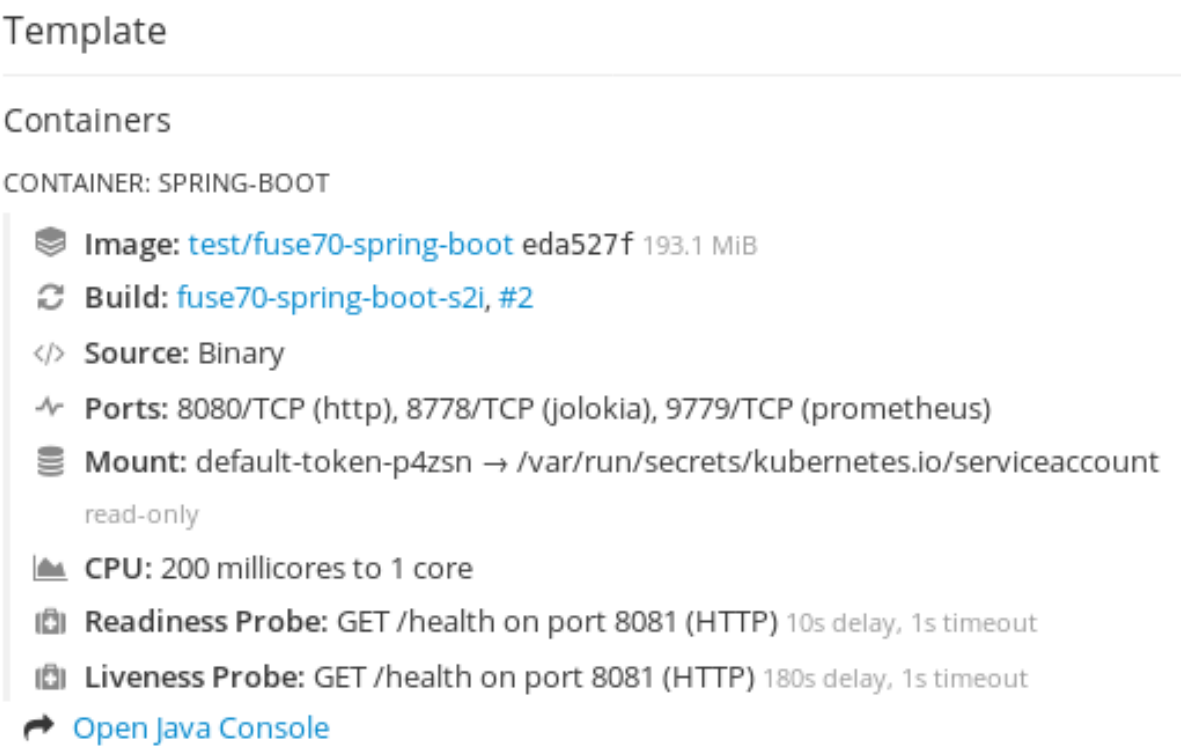
5. 확인할 애플리케이션에 대한 연결을 클릭합니다.

Fuse Console에 애플리케이션이 표시되는 새 브라우저 창이 열립니다.

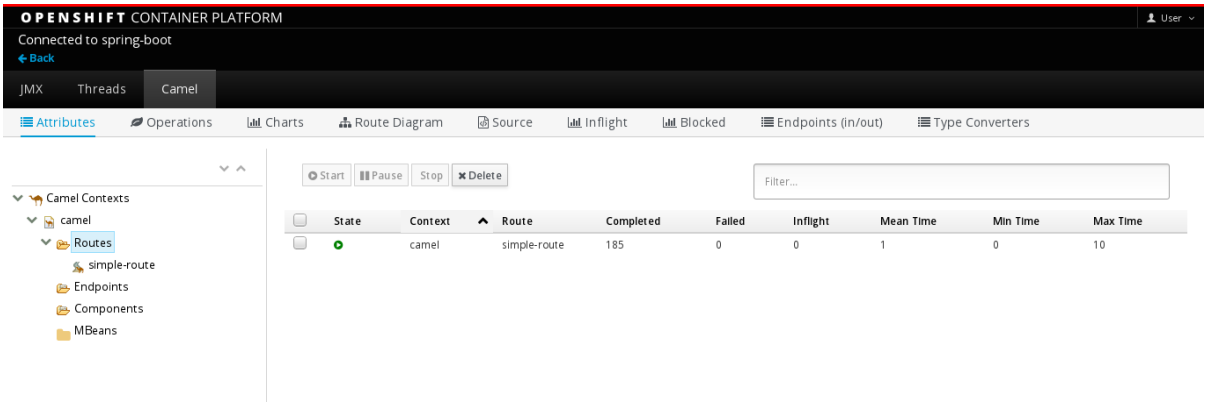
2.3.1.2. OpenShift 3.11의 Fuse Console에서 단일 Fuse Pod 모니터링

OpenShift 3.11에서 실행되는 **Fuse Pod**의 **Fuse Console**을 열 수 있습니다.

1. **OpenShift 프로젝트의 애플리케이션 → 포드 보기에서 Pod 이름을 클릭하여 실행 중인 Fuse Pod의 세부 정보를 확인합니다. 이 페이지 오른쪽에는 컨테이너 템플릿에 대한 요약이 표시됩니다.**



2. **이 보기에서 Open Java Console 링크를 클릭하여 Fuse 콘솔을 엽니다.**



참고

포드 뷰에 **Fuse Console**에 대한 링크를 표시하도록 **OpenShift**를 구성하려면 **OpenShift** 이미지에서 **Fuse**를 실행하는 **Pod**에서 **jolokia** 로 설정된 **name** 속성 내에서 **TCP** 포트를 선언해야 합니다.

```
{
  "kind": "Pod",
  [...]
  "spec": {
    "containers": [
      {
        [...]
        "ports": [
          {
            "name": "jolokia",
            "containerPort": 8778,
            "protocol": "TCP"
          }
```

2.3.1.3. 명령줄에서 Fuse 콘솔 배포

표 2.1. “Fuse 콘솔 템플릿” **Fuse** 애플리케이션 배포 유형에 따라 명령줄에서 **Fuse Console**을 배포하는 데 사용할 수 있는 **OpenShift 3.1** 템플릿을 설명합니다.

표 2.1. Fuse 콘솔 템플릿

유형	설명
fis-console-cluster-template.json	Fuse Console은 여러 네임스페이스 또는 프로젝트에 배포된 Fuse 애플리케이션을 검색하고 연결할 수 있습니다. 이 템플릿을 배포하려면 OpenShift cluster-admin 역할이 있어야 합니다.
fis-console-namespace-template.json	이 템플릿은 현재 OpenShift 프로젝트(네임스페이스)에 대한 Fuse Console 액세스를 제한하므로 단일 테넌트 배포 역할을 합니다. 이 템플릿을 배포하려면 현재 OpenShift 프로젝트에 대한 admin 역할이 있어야 합니다.

선택적으로 다음 명령을 실행하여 모든 템플릿에 대한 매개변수 목록을 볼 수 있습니다.

```
oc process --parameters -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fis-console-namespace-template.json
```

절차

명령줄에서 **Fuse** 콘솔을 배포하려면 다음을 수행합니다.

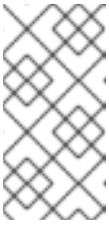
1. 다음 명령 중 하나를 실행하여 **Fuse Console** 템플릿을 기반으로 새 애플리케이션을 생성합니다(여기서 **myproject** 는 프로젝트 이름임).

   - **Fuse Console** 클러스터 템플릿의 경우 **myhost** 는 **Fuse Console**에 액세스할 수 있는 호스트 이름입니다.

   ```
   oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fis-console-cluster-template.json -p ROUTE_HOSTNAME=myhost
   ```

   - **Fuse Console** 네임스페이스 템플릿의 경우:

   ```
   oc new-app -n myproject -f https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003/fis-console-namespace-template.json
   ```

   참고

   OpenShift가 자동으로 생성되므로 네임스페이스 템플릿의 **route_hostname** 매개변수를 생략할 수 있습니다.

2. 다음 명령을 실행하여 **Fuse Console** 배포의 상태 및 **URL**을 가져옵니다.

   ```
   oc status
   ```

3. 브라우저에서 **Fuse** 콘솔에 액세스하려면 제공된 **URL**(예: **https://fuse-console.192.168.64.12.nip.io**)을 사용합니다.

3장. 관리자가 아닌 사용자로 OPENSHIFT에 FUSE 설치

OpenShift에서 Fuse를 사용하여 애플리케이션을 생성하고 OpenShift에 배포할 수 있습니다. 먼저 OpenShift 이미지 및 템플릿에 Fuse를 설치해야 합니다.

3.1. 관리자가 아닌 사용자로 OPENSHIFT 이미지 및 템플릿에 FUSE 설치

사전 요구 사항

- OpenShift 서버에 액세스할 수 있습니다. CDK 또는 원격 OpenShift 서버별 가상 OpenShift 서버일 수 있습니다.
- Red Hat Container Registry에 대한 인증을 구성했습니다.

자세한 내용은 다음을 참조하십시오.

- Red Hat 컨테이너 레지스트리 인증 구성.
- Red Hat CDK 3.11 Getting Started Guide

절차

1. OpenShift 프로젝트에서 Fuse를 빌드하고 배포할 수 있도록 다음과 같이 OpenShift Server에 로그인합니다.

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

여기서 OPENSHIFT_IP_ADDR 은 이 IP 주소가 항상 동일하지 않기 때문에 OpenShift 서버의 IP 주소 자리 표시자입니다.

참고

developer 사용자(개발 암호 포함)는 CDK를 통해 가상 OpenShift Server에서 자동으로 생성되는 표준 계정입니다. 원격 서버에 액세스하는 경우 OpenShift 관리자가 제공한 URL 및 인증 정보를 사용합니다.

2. **test** 라는 새 프로젝트 네임스페이스를 생성합니다(아직 존재하지 않는다고 가정).

```
oc new-project test
```

test 프로젝트 네임스페이스가 이미 있는 경우 해당 네임스페이스로 전환합니다.

```
oc project test
```

3. **OpenShift** 이미지 스트림에 **Fuse**를 설치합니다.

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003

oc create -n test -f ${BASEURL}/fis-image-streams.json
```

명령 출력에는 **OpenShift** 프로젝트의 **Fuse**에서 사용할 수 있는 **Fuse** 이미지 스트림이 표시됩니다.

4. 빠른 시작 템플릿을 설치합니다.

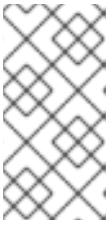
```
for template in eap-camel-amq-template.json \
 eap-camel-cdi-template.json \
 eap-camel-cxf-jaxrs-template.json \
 eap-camel-cxf-jaxws-template.json \
 eap-camel-jpa-template.json \
 karaf-camel-amq-template.json \
 karaf-camel-log-template.json \
 karaf-camel-rest-sql-template.json \
 karaf-cxf-rest-template.json \
 spring-boot-camel-amq-template.json \
 spring-boot-camel-config-template.json \
 spring-boot-camel-drools-template.json \
 spring-boot-camel-infinispan-template.json \
 spring-boot-camel-rest-sql-template.json \
 spring-boot-camel-rest-3scale-template.json \
 spring-boot-camel-template.json \
 spring-boot-camel-xa-template.json \
 spring-boot-camel-xml-template.json \
 spring-boot-cxf-jaxrs-template.json \
 spring-boot-cxf-jaxws-template.json ;
 do
 oc create -n test -f \
 ${BASEURL}/quickstarts/${template}
 done
```

5. **Spring Boot 2 빠른 시작 템플릿을 설치합니다.**

```
for template in spring-boot-2-camel-amq-template.json \
 spring-boot-2-camel-config-template.json \
 spring-boot-2-camel-drools-template.json \
 spring-boot-2-camel-infinispan-template.json \
 spring-boot-2-camel-rest-3scale-template.json \
 spring-boot-2-camel-template.json \
 spring-boot-2-camel-xa-template.json \
 spring-boot-2-camel-xml-template.json \
 spring-boot-2-cxf-jaxrs-template.json \
 spring-boot-2-cxf-jaxws-template.json \
 spring-boot-2-cxf-jaxrs-xml-template.json \
 spring-boot-2-cxf-jaxws-xml-template.json ;
 do oc create -n openshift -f \
 https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-sb2-760039-redhat-00001/quickstarts/${template}
 done
```

6. **Fuse Console용 템플릿을 설치합니다.**

```
oc create -n test -f ${BASEURL}/fis-console-cluster-template.json
oc create -n test -f ${BASEURL}/fis-console-namespace-template.json
```



참고

Fuse Console 배포에 대한 자세한 내용은 **OpenShift에 Fuse Console 설정**을 참조하십시오.

7. **(선택 사항) OpenShift 이미지 및 템플릿에서 설치된 Fuse를 확인합니다.**

```
oc get template -n test
```

8. **브라우저에서 OpenShift 콘솔로 이동합니다.**

a. **https://OPENSHIFT_IP_ADDR:8443** 을 사용하고 **OPENSHIFT_IP_ADDR** 을 **OpenShift** 서버의 **IP** 주소로 바꿉니다.

b. 인증 정보를 사용하여 **OpenShift** 콘솔에 로그인합니다(예: 사용자 이름 **developer** 및 암호 **developer**).

4장. 개발자용 시작하기

4.1. 개발 환경 준비

OpenShift 프로젝트에서 Fuse를 개발하고 테스트하는 데 필요한 기본적인 요구 사항은 OpenShift Server에 액세스하는 것입니다. 다음과 같은 기본 대안이 있습니다.

- Install Red Hat CDK
- 기존 OpenShift Server에 대한 원격 액세스 가져오기

4.1.1. 로컬 머신에 컨테이너 개발 키트(CDK) 설치

개발자로서 빠르게 시작하려는 경우 가장 실용적인 대안은 로컬 시스템에 Red Hat CDK 를 설치하는 것입니다. CDK를 사용하여 RHEL(Red Hat Enterprise Linux) 7에서 OpenShift 이미지를 실행하는 VM(가상 머신) 인스턴스를 부팅할 수 있습니다. CDK 설치는 다음 주요 구성 요소로 구성됩니다.

- 가상 머신(libvirt, Cryostat 또는 Hyper-V)
- Minishift: 컨테이너 개발 환경 시작 및 관리

중요

Red Hat CDK는 *개발 목적으로만 사용됩니다*. 프로덕션 환경과 같은 다른 용도로는 사용되지 않으며 알려진 보안 취약점을 해결하지 못할 수 있습니다. docker 형식의 컨테이너 내에서 미션 크리티컬 애플리케이션을 완전히 실행하려면 활성 RHEL 7 또는 RHEL Atomic 서브스크립션이 필요합니다. 자세한 내용은 Red Hat Container Development Kit (CDK) 지원을 참조하십시오.

사전 요구 사항

- Java 버전

  개발자 시스템에서 Fuse 7.6에서 지원하는 Java 버전을 설치했는지 확인합니다. 지원되는 Java 버전에 대한 자세한 내용은 지원되는 구성을 참조하십시오.

절차

로컬 시스템에 CDK를 설치하려면 다음을 수행합니다.

1. OpenShift의 Fuse의 경우 CDK 버전 3.11을 설치하는 것이 좋습니다. CDK 3.11 설치 및 사용에 대한 자세한 지침은 Red Hat CDK 3.11 시작 가이드에서 확인할 수 있습니다.

2. Red Hat Container Registry 인증 구성 지침에 따라 Red Hat 컨테이너 레지스트리에 액세스하도록 OpenShift 인증 정보를 구성합니다.

3. 2장. *관리자 시작하기* 에 설명된 대로 OpenShift 이미지 및 템플릿에 Fuse를 수동으로 설치합니다.

참고

CDK 버전에는 OpenShift 이미지 및 템플릿에 Fuse가 사전 설치되어 있을 수 있습니다. 그러나 OpenShift 인증 정보를 구성한 후 OpenShift 이미지 및 템플릿에 Fuse를 설치(또는 업데이트)해야 합니다.

4. 이 장의 예제를 진행하기 전에 Red Hat CDK 3.11 시작하기 가이드 의 내용을 읽고 이해해야 합니다.

4.1.2. 기존 OpenShift 서버에 대한 원격 액세스 가져오기

IT 부서에서 이미 일부 서버 시스템에 OpenShift 클러스터를 설정했을 수 있습니다. 이 경우 OpenShift에서 Fuse를 시작하려면 다음 요구 사항을 충족해야 합니다.

- 서버 머신은 지원되는 OpenShift Container Platform 버전을 실행해야 합니다(지원 구성 페이지에 설명되어 있음). 이 가이드의 예제는 3.11 버전에 대해 테스트되었습니다.

- OpenShift 관리자에게 OpenShift 컨테이너 기본 이미지에 최신 Fuse를 설치하고 OpenShift 서버의 OpenShift 템플릿에 Fuse를 설치하도록 요청합니다.

- 일반적인 개발자 권한이 있는 사용자 계정을 작성하도록 OpenShift 관리자에게 문의하십시오(OpenShift 프로젝트를 생성, 배포 및 실행할 수 있음).

-

관리자에게 **OpenShift Server**의 **URL**(사용 중 하나를 사용하여 **OpenShift** 콘솔로 이동하거나 **oc** 명령줄 클라이언트를 사용하여 **OpenShift**에 연결할 수 있음) 및 계정의 로그인 인증 정보를 요청합니다.

4.1.3. 클라이언트-Side 툴 설치

개발자 시스템에 다음 도구가 설치되어 있는 것이 좋습니다.

- **Apache Maven 3.6.x: OpenShift** 프로젝트의 로컬 빌드에 필요합니다. **Apache Maven** 다운로드 페이지에서 적절한 패키지를 다운로드합니다. **3.6.x** 이상 버전이 설치되어 있는지 확인합니다. 그러지 않으면 프로젝트를 빌드할 때 **Maven**에 종속성을 해결하는 데 문제가 있을 수 있습니다.

- **Git: OpenShift S2I** 소스 워크플로우에 필요하며 일반적으로 **OpenShift** 프로젝트에서 **Fuse**의 소스 제어에 권장됩니다. **Git** 다운로드 페이지에서 적절한 패키지를 다운로드합니다.

- **OpenShift** 클라이언트: **CDK**를 사용하는 경우 쉘에 입력하는 데 필요한 명령을 표시하는 **Mini shift oc -env** 를 사용하여 **PATH**에 **oc** 바이너리를 추가할 수 있습니다(**oc-env** 의 출력은 **OS** 및 쉘 유형에 따라 다릅니다).

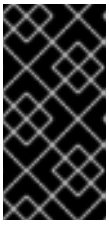
```
$ minishift oc-env
export PATH="/Users/john/.minishift/cache/oc/v1.5.0:$PATH"
# Run this command to configure your shell:
# eval $(minishift oc-env)
```

자세한 내용은 **CDK 3.11** *시작하기 가이드 의 OpenShift 클라이언트 바이너리 사용을 참조하십시오.*

CDK를 사용하지 않는 경우 **CLI** 참조 의 지침에 따라 **oc** 클라이언트 툴을 설치합니다.

- *(선택 사항)* **Docker** 클라이언트: 고급 사용자가 **Docker** 클라이언트 도구를 설치하는 것이 편리합니다(**OpenShift** 서버에서 실행되는 **docker** 데몬과 통신하기 위해). 운영 체제의 특정 바이너리 설치에 대한 자세한 내용은 **Docker** 설치 사이트를 참조하십시오.

자세한 내용은 **CDK 3.11** *시작 가이드 의 docker Daemon 재사용 을 참조하십시오.*



중요

OpenShift Server에서 실행 중인 OpenShift 버전과 호환되는 oc 툴 버전 및 docker 툴을 설치해야 합니다.

추가 리소스

(선택 사항) Red Hat JBoss CodeReady Studio: *Red Hat JBoss CodeReady Studio* 는 OpenShift 애플리케이션에서 Fuse 개발을 지원하는 Eclipse 기반 개발 환경입니다. 이 개발 환경을 설치하는 방법에 대한 자세한 내용은 Red Hat JBoss CodeReady Studio 설치를 참조하십시오.

4.1.4. Maven 리포지토리 구성

로컬 시스템에서 OpenShift 프로젝트에 Fuse를 빌드하는 데 필요한 archetypes 및 아티팩트를 보유하는 Maven 리포지토리를 구성합니다.

절차

1. 일반적으로 ~/.m2/ settings.xml (Linux 또는 macOS) 또는 Documents 및 Settings\<USER_NAME>\.m2\settings.xml(Windows의 경우)에 있는 Maven settings.xml 파일을 엽니다.

2. 다음 Maven 리포지토리를 추가합니다.

   - Maven 중앙: https://repo1.maven.org/maven2
   - Red Hat GA repository: https://maven.repository.redhat.com/ga
   - Red Hat EA repository: https://maven.repository.redhat.com/earlyaccess/all

   이전 리포지토리를 종속성 리포지토리 섹션과 settings.xml 파일의 플러그인 리포지토리 섹션에 추가해야 합니다.

4.2. OPENSHIFT에서 FUSE에서 애플리케이션 생성 및 배포

OpenShift에서 Fuse를 사용하여 애플리케이션을 생성하고 다음 OpenShift S2I(Source-to-Image) 애플리케이션 개발 워크플로우 중 하나를 사용하여 OpenShift에 배포할 수 있습니다.

S2I 바이너리 워크플로

*바이너리 소스*의 빌드 입력이 있는 **S2I**입니다. 이 워크플로는 빌드가 개발자의 자체 시스템에서 부분적으로 실행된다는 사실이 특징입니다. 바이너리 패키지를 로컬로 빌드한 후 이 워크플로는 바이너리 패키지를 **OpenShift**로 전달합니다. 자세한 내용은 ***OpenShift 3.5 개발자 가이드의 바이너리 소스 를 참조하십시오.***

S2I 소스 워크플로우

***Git** 소스*의 빌드 입력이 있는 **S2I** . 이 워크플로는 빌드가 전적으로 **OpenShift** 서버에서 실행된다는 사실이 특징입니다. 자세한 내용은 ***OpenShift 3.5** 개발자 가이드의 **Git** 소스 를 참조하십시오.*

4.2.1. S2I 바이너리 워크플로를 사용하여 애플리케이션 생성 및 배포

이 섹션에서는 **OpenShift S2I** 바이너리 워크플로우를 사용하여 **OpenShift** 프로젝트에서 **Fuse**를 생성, 빌드 및 배포합니다.

절차

1. **Maven archetype**을 사용하여 **OpenShift** 프로젝트에서 새 **Fuse**를 생성합니다. 이 예제에서는 샘플 **Spring Boot Camel** 프로젝트를 생성하는 **archetype**을 사용합니다. 새 쉘 프롬프트를 열고 다음 **Maven** 명령을 입력합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-760024-redhat-00001/archetypes-catalog-2.2.0.fuse-760024-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype \
  -DarchetypeVersion=2.2.0.fuse-760024-redhat-00001
```

archetype 플러그인은 대화형 모드로 전환되어 나머지 필드를 입력하라는 메시지를 표시합니다.

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-spring-boot
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

메시지가 표시되면 **groupId** 값으로 **org.example.fis** 를 입력하고 **artifactId** 값으로 **fuse76-spring-boot** 를 입력합니다. 나머지 필드의 기본값을 수락합니다.

2.

이전 명령이 **BUILD SUCCESS** 상태로 종료되면 이제 **fuse76-spring-boot** 하위 디렉터리에 **OpenShift** 프로젝트에서 새 **Fuse**가 있어야 합니다. **fuse76-spring-boot/src/main/resources/spring/camel-context.xml** 파일에서 **XML DSL** 코드를 검사할 수 있습니다. 데모 코드는 임의의 숫자가 포함된 메시지를 로그에 지속적으로 전송하는 간단한 **Camel** 경로를 정의합니다.

3.

OpenShift 프로젝트에서 **Fuse**를 빌드하고 배포할 수 있도록 다음과 같이 **OpenShift Server**에 로그인합니다.

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

여기서 **OPENSHIFT_IP_ADDR** 은 이 **IP** 주소가 항상 동일하지 않기 때문에 **OpenShift** 서버의 **IP** 주소 자리 표시자입니다.

참고

developer 사용자(개발 암호 포함)는 **CDK**를 통해 가상 **OpenShift Server**에서 자동으로 생성되는 표준 계정입니다. 원격 서버에 액세스하는 경우 **OpenShift** 관리자가 제공한 **URL** 및 인증 정보를 사용합니다.

4.

다음과 같이 **openshift** 프로젝트에 아직 없는 경우 **openshift** 프로젝트로 전환합니다.

```
oc project openshift
```

5.

다음 명령을 실행하여 **OpenShift** 이미지 및 템플릿의 **Fuse**가 이미 설치되어 있고 액세스할 수 있는지 확인합니다.

```
oc get template -n openshift
```

이미지 및 템플릿이 사전 설치되지 않았거나 제공된 버전이 최신 버전이 아닌 경우 **Fuse**를 **OpenShift** 이미지 및 템플릿에 수동으로 설치(또는 업데이트)합니다. **OpenShift** 이미지에 **Fuse**를 설치하는 방법에 대한 자세한 내용은 **2장. *관리자 시작하기*** 을 참조하십시오.

6.

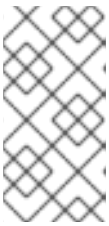
이제 **fuse76-spring-boot** 프로젝트를 빌드하고 배포할 준비가 되었습니다. **OpenShift**에 로

그인되어 있다고 가정하면 **fuse76-spring-boot** 프로젝트의 디렉터리로 변경한 다음 다음과 같이 프로젝트를 빌드하고 배포합니다.

```
cd fuse76-spring-boot
mvn fabric8:deploy -Popenshift
```

빌드가 성공적으로 완료되면 다음과 같은 일부 출력이 표시됩니다.

```
...
[INFO] OpenShift platform detected
[INFO] Using project: openshift
[INFO] Creating a Service from openshift.yml namespace openshift name fuse76-spring-boot
[INFO] Created Service: target/fabric8/applyJson/openshift/service-fuse76-spring-boot.json
[INFO] Using project: openshift
[INFO] Creating a DeploymentConfig from openshift.yml namespace openshift name fuse76-
spring-boot
[INFO] Created DeploymentConfig: target/fabric8/applyJson/openshift/deploymentconfig-
fuse76-spring-boot.json
[INFO] Creating Route openshift:fuse76-spring-boot host: null
[INFO] F8: HINT: Use the command `oc get pods -w` to watch your pods start up
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 05:38 min
[INFO] Finished at: 2019-10-24T12:15:06+05:30
[INFO] Final Memory: 63M/688M
[INFO] ------------------------------------------------------------------------
```



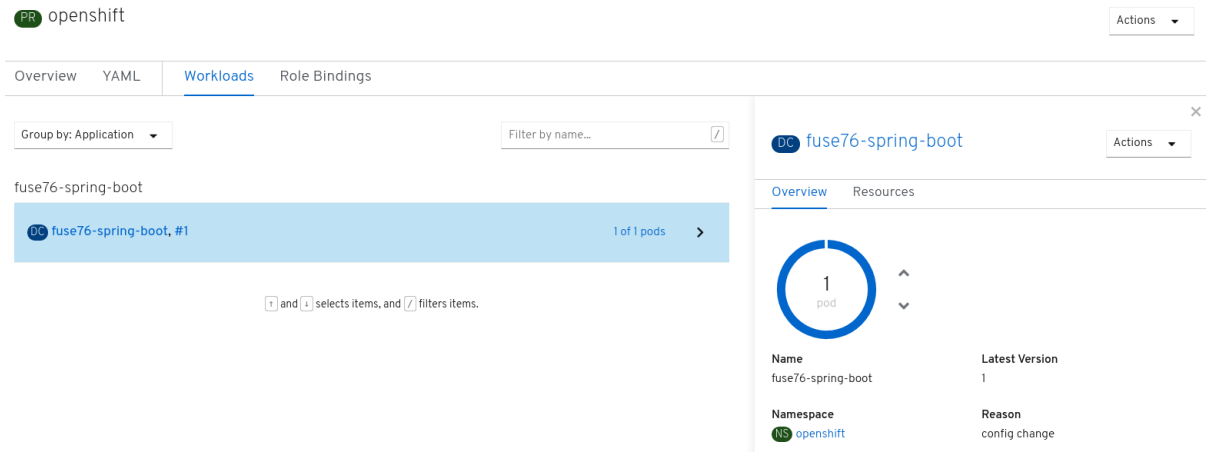
참고

이 명령을 처음 실행하는 경우 **Maven**은 많은 종속 항목을 다운로드해야 하며 몇 분 정도 걸립니다. 후속 빌드가 더 빨라집니다.

7. 브라우저에서 **OpenShift** 콘솔로 이동하여 인증 정보를 사용하여 콘솔에 로그인합니다(예: 사용자 이름 **developer** 및 암호 **developer**).

8. 왼쪽 패널에서 홈을 확장합니다. **Projects** 를 클릭한 다음 **openshift** 프로젝트를 선택하여 프로젝트 대시보드를 확인합니다.

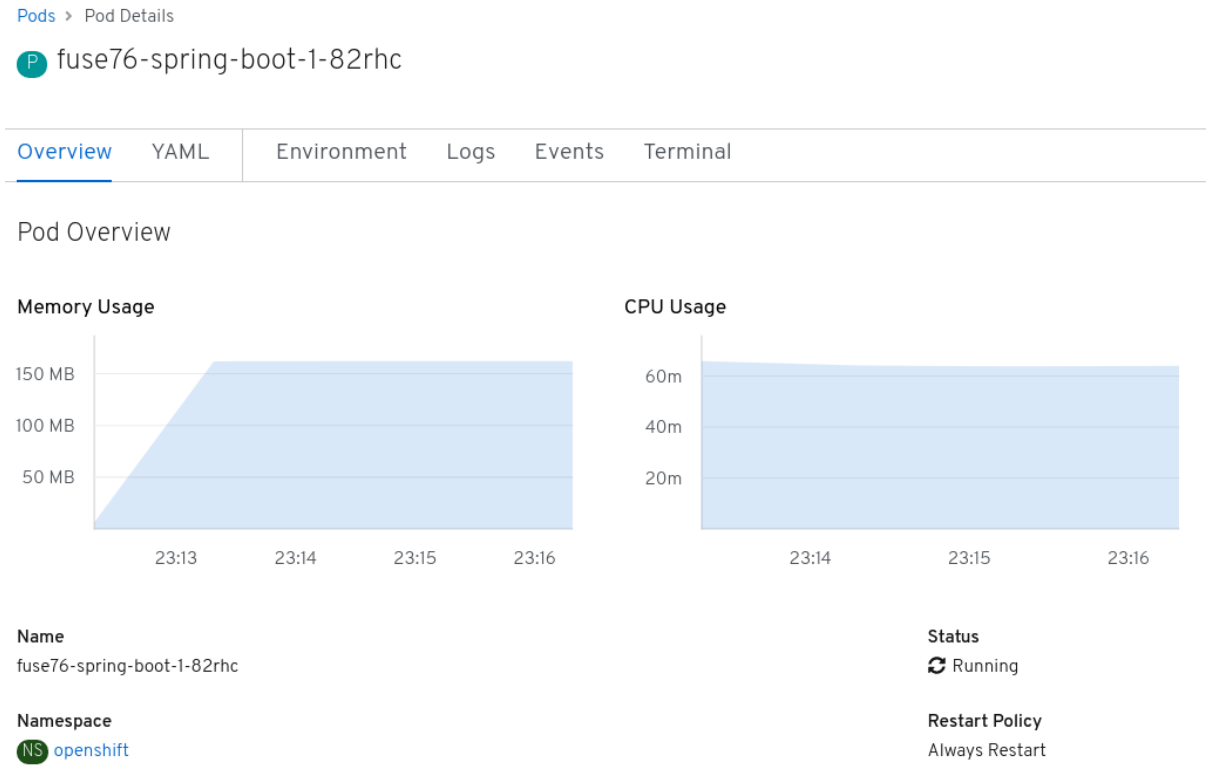
9. **fuse76-spring-boot** 를 클릭하여 **fuse76-spring-boot** 애플리케이션의 개요 정보 페이지를 확인합니다.



10. **왼쪽 패널에서 워크로드를 확장합니다.**

11. **포드 를 클릭합니다. openshift 프로젝트에서 실행 중인 모든 Pod가 표시됩니다.**

12. **Pod 이름 (이 예에서 fuse76-spring-boot-xxxxx)을 클릭하여 실행 중인 Pod의 세부 정보를 확인합니다.**



13.

로그 탭을 클릭하여 애플리케이션 로그를 보고 로그를 아래로 스크롤하여 **Camel** 애플리케이션에서 생성한 임의의 수 로그 메시지를 찾습니다.

```
...
06:45:54.311 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 130
06:45:56.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 898
06:45:58.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 414
06:46:00.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 486
06:46:02.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 093
06:46:04.265 [Camel (MyCamel) thread #1 - timer://foo] INFO simple-route - >>> 080
```

14. 실행 중인 **Pod**를 종료하려면 다음을 수행합니다.

a. **openshift** 프로젝트의 프로젝트 상태 페이지에서 워크로드 탭을 클릭한 다음 **fuse76-spring-boot** 애플리케이션을 클릭합니다.

b. 개요 탭을 클릭하여 애플리케이션의 개요 정보 페이지를 확인합니다.

c. **Pod** 아이콘 옆에 있는 아래쪽 화살표

를 클릭하여 **Pod**를 중지하도록 **0**으로 축소합니다.

4.2.2. 프로젝트 배포 취소 및 재배포

다음과 같이 프로젝트 배포를 취소하거나 재배포할 수 있습니다.

절차

- 프로젝트 배포를 취소하려면 다음 명령을 입력합니다.

```
mvn fabric8:undeploy
```

- 프로젝트를 재배포하려면 명령을 입력합니다.

```
mvn fabric8:undeploy
mvn fabric8:deploy -Popenshift
```

4.2.3. S2I 소스 워크플로우를 사용하여 애플리케이션 생성 및 배포

이 섹션에서는 **OpenShift S2I** 소스 워크플로를 사용하여 템플릿을 기반으로 **OpenShift** 애플리케이션에 **Fuse**를 빌드하고 배포합니다. 이 데모의 시작점은 원격 **Git** 리포지토리에 저장된 빠른 시작 프로젝트입니다. **OpenShift** 콘솔을 사용하면 **OpenShift** 서버에서 이 빠른 시작 프로젝트를 다운로드, 빌드 및 배포합니다.

절차

1. 다음과 같이 **OpenShift** 서버에 로그인합니다.

```
oc login -u developer -p developer https://OPENSHIFT_IP_ADDR:8443
```

여기서 **OPENSHIFT_IP_ADDR** 은 이 **IP** 주소가 항상 동일하지 않기 때문에 **OpenShift** 서버의 **IP** 주소 자리 표시자입니다.

참고

developer 사용자(개발 암호 포함)는 **CDK**를 통해 가상 **OpenShift Server**에서 자동으로 생성되는 표준 계정입니다. 원격 서버에 액세스하는 경우 **OpenShift** 관리자가 제공한 **URL** 및 인증 정보를 사용합니다.

2. 다음과 같이 **openshift** 프로젝트에 아직 없는 경우 **openshift** 프로젝트로 전환합니다.

```
oc project openshift
```

3. 다음 명령을 실행하여 **OpenShift** 템플릿의 **Fuse**가 이미 설치되어 있고 액세스할 수 있는지 확인합니다.

```
oc get template -n openshift
```

이미지 및 템플릿이 사전 설치되지 않았거나 제공된 버전이 최신 버전이 아닌 경우 **Fuse**를 **OpenShift** 이미지 및 템플릿에 수동으로 설치(또는 업데이트)합니다. **OpenShift** 이미지에 **Fuse**를 설치하는 방법에 대한 자세한 내용은 **2장. *관리자 시작하기*** 을 참조하십시오.

4. 다음 명령을 입력하여 **Spring Boot** 빠른 시작 템플릿으로 **Red Hat Fuse 7.6 Camel XML DSL** 을 실행하는 데 필요한 리소스를 생성합니다. 그러면 빠른 시작에 대한 배포 구성 및 빌드 구

성이 생성됩니다. 퀵 스타트의 기본 매개변수 및 생성된 리소스에 대한 정보가 터미널에 표시됩니다.

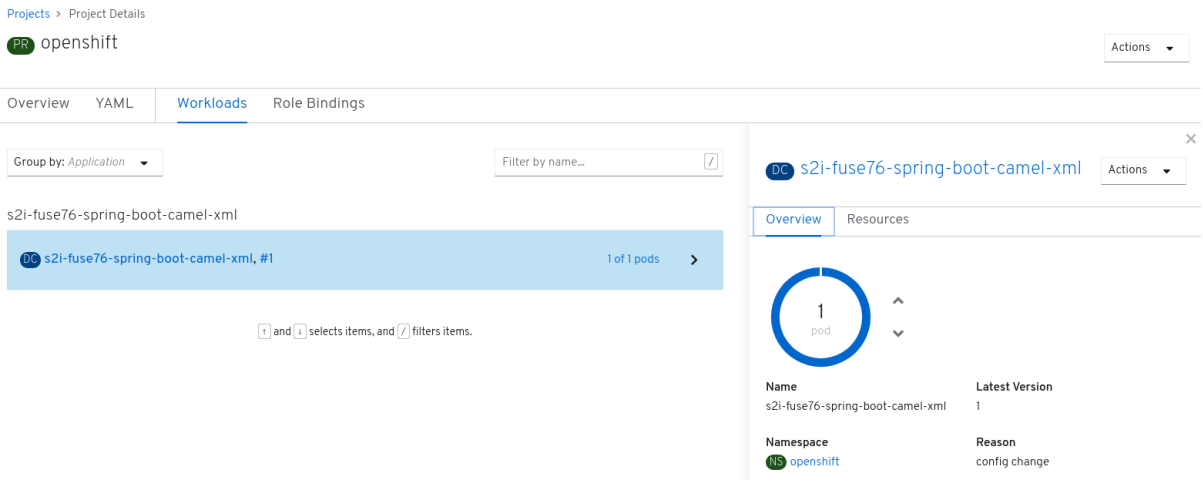
```
oc new-app s2i-fuse76-spring-boot-camel-xml

--> Deploying template "openshift/s2i-fuse76-spring-boot-camel-xml" to project openshift
...
--> Creating resources ...
    imagestream.image.openshift.io "s2i-fuse76-spring-boot-camel-xml" created
    buildconfig.build.openshift.io "s2i-fuse76-spring-boot-camel-xml" created
    deploymentconfig.apps.openshift.io "s2i-fuse76-spring-boot-camel-xml" created
--> Success
    Build scheduled, use 'oc logs -f bc/s2i-fuse76-spring-boot-camel-xml' to track its progress.
    Run 'oc status' to view your app.
```

5. 브라우저에서 **OpenShift** 웹 콘솔로 이동하고(**https://OPENSHIFT_IP_ADDR**, **OPENSHIFT_IP_ADDR** 을 클러스터의 **IP** 주소로 교체) 인증 정보로 로그인합니다(예: 사용자 이름 **developer** 및 암호, 개발자).

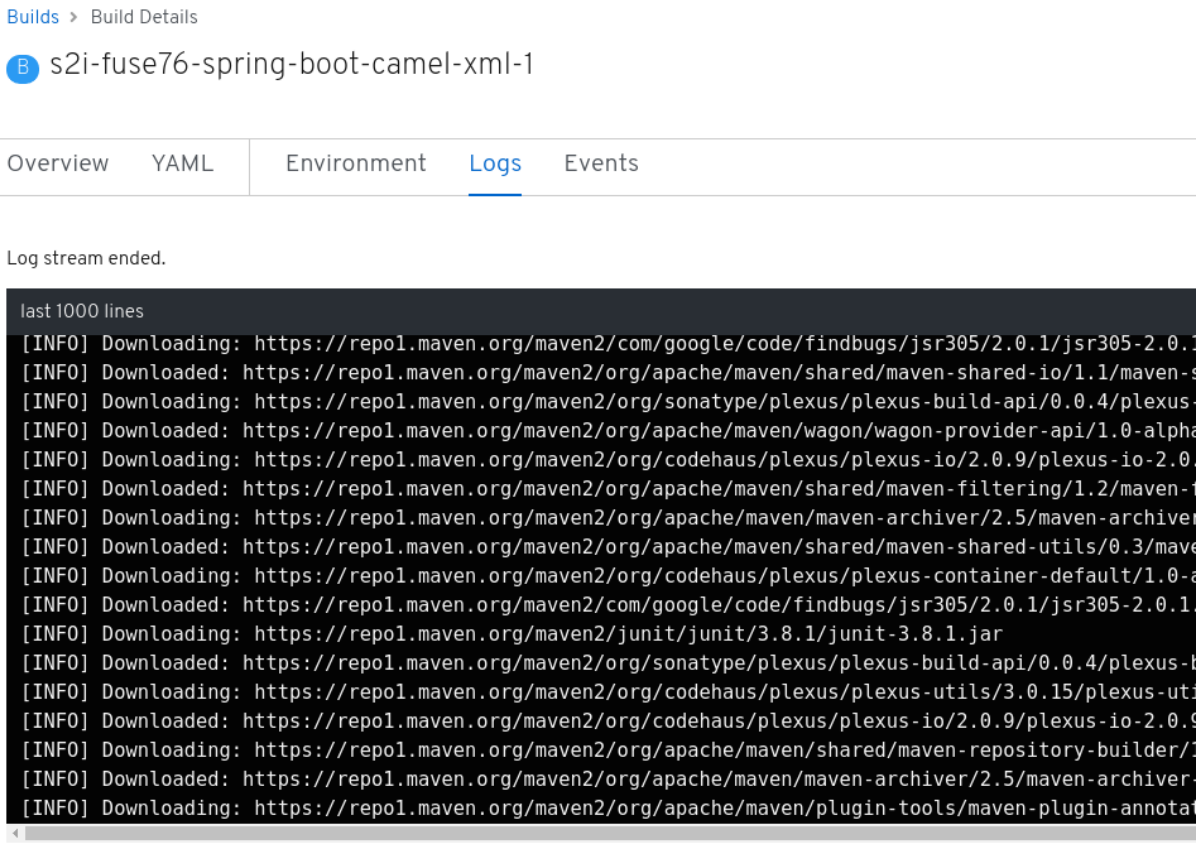
6. 왼쪽 패널에서 홈을 확장합니다. **Projects** 를 클릭한 다음 **openshift** 프로젝트를 선택하여 프로젝트 대시보드를 확인합니다.

7. 워크로드 탭을 클릭합니다. 선택한 네임스페이스(예: **openshift**)의 기존 애플리케이션이 모두 표시됩니다.

8. **s2i-fuse76-spring-boot-camel-xml** 을 클릭하여 퀵 스타트에 대한 개요 정보 페이지를 확인합니다.

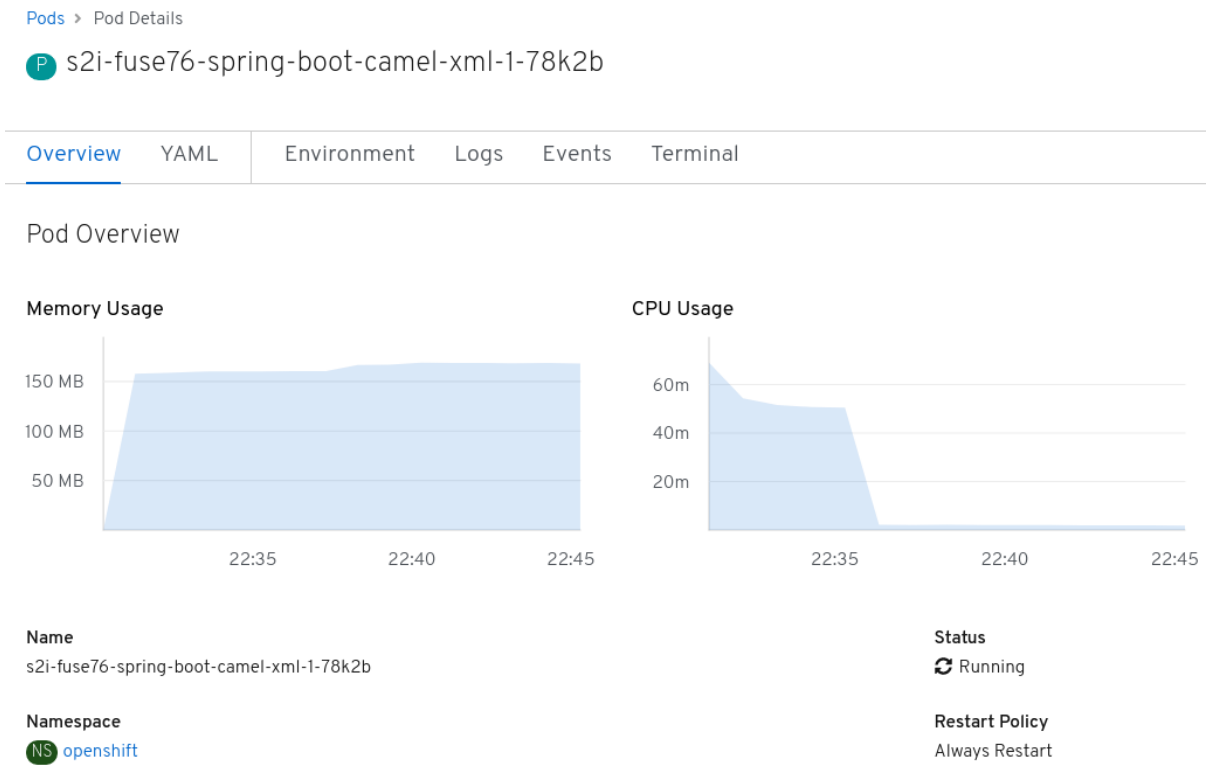


9. 리소스 탭을 클릭한 다음 로그 보기를 클릭하여 애플리케이션의 빌드 로그를 확인합니다.



10. 왼쪽 패널에서 워크로드를 확장합니다.

11. **Pod** 를 클릭한 다음 **s2i-fuse76-spring-boot-camel-xml-xxxx** 를 클릭합니다. 애플리케이션의 **Pod** 세부 정보가 표시됩니다.



12. **실행 중인 Pod를 종료하려면 다음을 수행합니다.**

a. **openshift 프로젝트의 프로젝트 세부 정보 페이지에서 워크로드 탭을 클릭한 다음 s2i-fuse76-spring-boot-camel-xml-xxxx 애플리케이션을 클릭합니다.**

b. **개요 탭을 클릭하여 애플리케이션의 개요 정보 페이지를 확인합니다.**

c. **Pod 아이콘 옆에 있는 아래쪽 화살표**

를 클릭하여 Pod를 중지하도록 0으로 축소합니다.

d. **아래쪽 화살표를 사용하여 Pod를 중지하려면 0으로 축소합니다.**

5장. SPRING BOOT 이미지용 애플리케이션 개발

이 장에서는 Spring Boot 이미지용 애플리케이션을 개발하는 방법을 설명합니다.

5.1. MAVEN ARCHETYPE을 사용하여 SPRING BOOT 프로젝트 생성

Maven archetypes를 사용하여 Spring Boot 프로젝트를 생성하려면 다음 단계를 따르십시오.

절차

1. 시스템의 적절한 디렉터리로 이동합니다.

2. 쉘 프롬프트에서 다음 mvn 명령을 입력하여 Spring Boot 프로젝트를 생성합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
 -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-760024-redhat-00001/archetypes-catalog-2.2.0.fuse-760024-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype \
  -DarchetypeVersion=2.2.0.fuse-760024-redhat-00001
```

archetype 플러그인은 대화형 모드로 전환되어 나머지 필드를 입력하라는 메시지를 표시합니다.

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-spring-boot
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

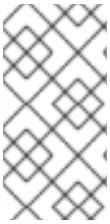
메시지가 표시되면 groupId 값으로 org.example.fis 를 입력하고 artifactId 값으로 fuse76-spring-boot 를 입력합니다. 나머지 필드의 기본값을 수락합니다.

3. 위의 명령이 BUILD SUCCESS 상태로 종료되면 fuse76-spring-boot 하위 디렉터리에

OpenShift 프로젝트에서 새 **Fuse**가 있어야 합니다.

4. 이제 **fuse76-spring-boot** 프로젝트를 빌드하고 배포할 준비가 되었습니다. **OpenShift**에 로그인되어 있다고 가정하면 **fuse76-spring-boot** 프로젝트의 디렉터리로 변경한 다음 다음과 같이 프로젝트를 빌드하고 배포합니다.

```
cd fuse76-spring-boot
mvn fabric8:deploy -Popenshift
```



참고

사용 가능한 **Spring Boot archetypes**의 전체 목록은 Spring Boot Archetype Catalog 를 참조하십시오.

5.2. MAVEN ARCHETYPE을 사용하여 SPRING BOOT 2 프로젝트 생성

이 빠른 시작에서는 **Maven archetypes**를 사용하여 **Spring Boot 2** 프로젝트를 생성하는 방법을 보여줍니다.

절차

1. 시스템의 적절한 디렉터리로 이동합니다.

2. 쉘 프롬프트에서 다음 **mvn** 명령을 입력하여 **Spring Boot 2** 프로젝트를 생성합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-
catalog/2.2.0.fuse-sb2-760038-redhat-00001/archetypes-catalog-2.2.0.fuse-sb2-760038-
redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-xml-archetype \
  -DarchetypeVersion=2.2.0.fuse-sb2-760038-redhat-00001
```

archetype 플러그인은 대화형 모드로 전환되어 나머지 필드를 입력하라는 메시지를 표시합니다.

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-spring-boot
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
```

```
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-spring-boot
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

메시지가 표시되면 **groupId** 값으로 **org.example.fis** 를 입력하고 **artifactId** 값으로 **fuse76-spring-boot** 를 입력합니다. 나머지 필드의 기본값을 수락합니다.

3. 위의 명령이 **BUILD SUCCESS** 상태로 종료되면 **fuse76-spring-boot** 하위 디렉터리에 **OpenShift** 프로젝트에서 새 **Fuse**가 있어야 합니다.

4. 이제 **fuse76-spring-boot** 프로젝트를 빌드하고 배포할 준비가 되었습니다. **OpenShift**에 로그인되어 있다고 가정하면 **fuse76-spring-boot** 프로젝트의 디렉터리로 변경한 다음 다음과 같이 프로젝트를 빌드하고 배포합니다.

```
cd fuse76-spring-boot
mvn fabric8:deploy -Popenshift
```

참고

사용 가능한 **Spring Boot 2 archetypes**의 전체 목록은 **Spring Boot 2 Archetype 카탈로그** 를 참조하십시오.

5.3. CAMEL SPRING BOOT 애플리케이션 구조

Camel Spring Boot 애플리케이션의 디렉터리 구조는 다음과 같습니다.

```
├── LICENSE.md
├── pom.xml
├── README.md
├── configuration
│   └── settings.xml
└── src
    ├── main
    │   ├── fabric8
    │   │   └── deployment.yml
    │   ├── java
    │   │   └── org
    │   │       └── example
    │   │           └── fis
    │   │               ├── Application.java
    │   │               └── MyTransformer.java
```

```
    │   └── resources
    │       ├── application.properties
    │       ├── logback.xml
    │       └── spring
    │           └── camel-context.xml
    └── test
        └── java
            └── org
                └── example
                    └── fis
```

다음 파일이 애플리케이션을 개발하는 데 중요한 위치:

pom.xml

추가 종속 항목을 포함합니다. **Spring Boot**와 호환되는 **Camel** 구성 요소는 시작 버전(예: **camel-jdbc-starter** 또는 **camel-infinispan-starter**)에서 사용할 수 있습니다. 시작자가 **pom.xml** 에 포함되면 부팅 시 **Camel** 컨텐츠에 자동으로 구성 및 등록됩니다. 사용자는 **application.properties** 파일을 사용하여 구성 요소의 속성을 구성할 수 있습니다.

application.properties

구성을 외부화하고 다른 환경에서 동일한 애플리케이션 코드로 작업할 수 있습니다. 자세한 내용은 **Externalized Configuration**을 참조하십시오.

예를 들어 이 **Camel** 애플리케이션에서는 애플리케이션 이름 또는 **IP** 주소와 같은 특정 속성을 구성할 수 있습니다.

application.properties

```
#spring.main.sources=org.example.fos

logging.config=classpath:logback.xml

# the options from org.apache.camel.spring.boot.CamelConfigurationProperties can be configured here
camel.springboot.name=MyCamel

# lets listen on all ports to ensure we can be invoked from the pod IP
server.address=0.0.0.0
management.address=0.0.0.0

# lets use a different management port in case you need to listen to HTTP requests on 8080
management.port=8081

# disable all management endpoints except health
endpoints.enabled = false
endpoints.health.enabled = true
```

Application.java

애플리케이션을 실행하는 데 중요한 파일입니다. 사용자는 여기에서 **Spring DSL**을 사용하여 경로를 구성하기 위해 **camel-context.xml** 파일을 가져옵니다.

Application.java 파일은 **@Configuration,@EnableAutoConfiguration** 및 **@ComponentScan**과 동일한 **@SpringBootApplication** 주석을 지정합니다.

Application.java

```
@SpringBootApplication
// load regular Blueprint file from the classpath that contains the Camel XML DSL
@ImportResource({"classpath:blueprint/camel-context.xml"})
```

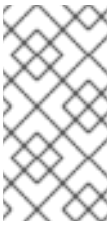
Spring Boot 애플리케이션을 실행하는 기본 방법이 있어야 합니다.

Application.java

```
public class Application {
    /**
     * A main method to start this application.
     */
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

camel-context.xml

src/main/resources/spring/camel-context.xml 은 **Camel** 경로가 포함되어 있으므로 애플리케이션을 개발하는 데 중요한 파일입니다.

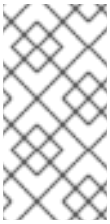


참고

Spring-Boot 애플리케이션 개발에 대한 자세한 내용은 첫 번째 Spring Boot Application 개발에서 확인할 수 있습니다.

src/main/fabric8/deployment.yml

fabric8-maven-plugin에서 생성된 기본 OpenShift 구성 파일과 병합되는 추가 구성을 제공합니다.



참고

이 파일은 Spring Boot 애플리케이션의 일부를 사용하지 않지만 CPU 및 메모리 사용과 같은 리소스를 제한하는 데 모든 빠른 시작에서 사용됩니다.

5.4. SPRING BOOT ARCHETYPE 카탈로그

Spring Boot Archetype 카탈로그에는 다음 예제가 포함되어 있습니다.

표 5.1. Spring Boot Maven Archetypes

이름	설명
spring-boot-camel-archetype	fabric8 Java 기본 이미지를 기반으로 Apache Camel을 Spring Boot와 함께 사용하는 방법을 보여줍니다.
spring-boot-camel-amq-archetype	Spring Boot 애플리케이션을 ActiveMQ 브로커에 연결하고 Kubernetes 또는 OpenShift를 사용하여 두 Camel 경로 간에 JMS 메시징을 사용하는 방법을 보여줍니다.
spring-boot-camel-config-archetype	Kubernetes ConfigMaps 및 Secrets를 사용하여 Spring Boot 애플리케이션을 구성하는 방법을 보여줍니다.
spring-boot-camel-drools-archetype	Apache Camel을 사용하여 Kubernetes 또는 OpenShift에서 실행되는 Spring Boot 애플리케이션을 원격 Kie 서버와 통합하는 방법을 보여줍니다.
spring-boot-camel-infinispan-archetype	Hot Rod 프로토콜을 사용하여 Spring Boot 애플리케이션을 JBoss Data Grid 또는 Infinispan 서버에 연결하는 방법을 보여줍니다.
spring-boot-camel-rest-3scale-archetype	Camel의 REST DSL을 사용하여 RESTful API를 노출하고 3scale에 노출하는 방법을 보여줍니다.

이름	설명
spring-boot-camel-rest-sql-archetype	Camel의 REST DSL과 함께 JDBC를 통한 SQL을 사용하여 RESTful API를 노출하는 방법을 보여줍니다.
spring-boot-camel-xa-archetype	Spring Boot, Camel 및 XA 트랜잭션. 이 예제에서는 두 개의 외부 트랜잭션 리소스에서 XA 트랜잭션을 지원하는 Spring Boot에서 Camel 서비스를 실행하는 방법(A-MQ) 및 데이터베이스(PostgreSQL)를 보여줍니다. 이 빠른 시작에는 PostgreSQL 데이터베이스가 필요하며 A-MQ 브로커가 먼저 배포 및 실행 중 하나는 Openshift 서비스 카탈로그에 제공된 템플릿을 사용하는 것입니다.
spring-boot-camel-xml-archetype	블루프린트 구성 파일을 통해 Spring Boot에서 Camel 경로를 구성하는 방법을 보여줍니다.
spring-boot-cxf-jaxrs-archetype	fabric8 Java 기본 이미지를 기반으로 Apache CXF를 Spring Boot와 함께 사용하는 방법을 보여줍니다. 빠른 시작에서는 Spring Boot를 사용하여 OpenAPI가 활성화된 CXF CryostatRS 끝점을 포함하는 애플리케이션을 구성합니다.
spring-boot-cxf-jaxws-archetype	fabric8 Java 기본 이미지를 기반으로 Apache CXF를 Spring Boot와 함께 사용하는 방법을 보여줍니다. 빠른 시작에서는 Spring Boot를 사용하여 CXF CryostatWS 엔드포인트가 포함된 애플리케이션을 구성합니다.

중요

기술 프리뷰 빠른 시작도 사용할 수 있습니다. Spring Boot Camel XA 트랜잭션 빠른 시작은 Spring Boot를 사용하여 XA 트랜잭션을 지원하는 Camel 서비스를 실행하는 방법을 보여줍니다. 이 빠른 시작에서는 JMS(AMQ) 브로커와 데이터베이스(PostgreSQL)의 두 가지 외부 트랜잭션 리소스를 사용하는 방법을 보여줍니다. 이 빠른 시작은 https://github.com/jboss-fuse/spring-boot-camel-xa 에서 확인할 수 있습니다.

기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있으며 프로덕션 환경에서 사용하는 것이 좋습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다. 자세한 내용은 Red Hat 기술 프리뷰 기능 지원 범위를 참조하십시오.

5.5. SPRING BOOT 2 ARCHETYPE 카탈로그

Spring Boot 2 Archetype 카탈로그에는 다음 예제가 포함되어 있습니다.

표 5.2. Spring Boot 2 Maven Archetypes

이름	설명
spring-boot-camel-archetype	fabric8 Java 기본 이미지를 기반으로 Apache Camel을 Spring Boot와 함께 사용하는 방법을 보여줍니다.
spring-boot-camel-amq-archetype	Spring Boot 애플리케이션을 ActiveMQ 브로커에 연결하고 Kubernetes 또는 OpenShift를 사용하여 두 Camel 경로 간에 JMS 메시징을 사용하는 방법을 보여줍니다.
spring-boot-camel-drools-archetype	Apache Camel을 사용하여 Kubernetes 또는 OpenShift에서 실행되는 Spring Boot 애플리케이션을 원격 Kie 서버와 통합하는 방법을 보여줍니다.
spring-boot-camel-infinispan-archetype	Hot Rod 프로토콜을 사용하여 Spring Boot 애플리케이션을 JBoss Data Grid 또는 Infinispan 서버에 연결하는 방법을 보여줍니다.
spring-boot-camel-rest-3scale-archetype	Camel의 REST DSL을 사용하여 RESTful API를 노출하고 3scale에 노출하는 방법을 보여줍니다.
spring-boot-camel-rest-sql-archetype	Camel의 REST DSL과 함께 JDBC를 통한 SQL을 사용하여 RESTful API를 노출하는 방법을 보여줍니다.
spring-boot-camel-xml-archetype	블루프린트 구성 파일을 통해 Spring Boot에서 Camel 경로를 구성하는 방법을 보여줍니다.
spring-boot-cxf-jaxrs-archetype	fabric8 Java 기본 이미지를 기반으로 Apache CXF를 Spring Boot와 함께 사용하는 방법을 보여줍니다. 빠른 시작에서는 Spring Boot를 사용하여 Swagger가 활성화된 CXF CryostatRS 끝점이 포함된 애플리케이션을 구성합니다.
spring-boot-cxf-jaxws-archetype	fabric8 Java 기본 이미지를 기반으로 Apache CXF를 Spring Boot와 함께 사용하는 방법을 보여줍니다. 빠른 시작에서는 Spring Boot를 사용하여 CXF CryostatWS 엔드포인트가 포함된 애플리케이션을 구성합니다.
spring-boot-cxf-jaxrs-xml-archetype	OpenShift에서 Apache CXF Cryostat-RS를 Spring Boot 2와 함께 사용하는 방법을 보여줍니다. 이 빠른 시작에서는 Spring Boot2를 사용하여 Swagger가 활성화된 CXF CryostatRS 끝점이 포함된 Spring 구성 파일 기반 CXF 애플리케이션을 시작합니다.
spring-boot-cxf-jaxws-xml-archetype	OpenShift에서 Apache CXF Cryostat-WS를 Spring Boot 2와 함께 사용하는 방법을 보여줍니다. 빠른 시작은 Spring Boot2를 사용하여 CXF CryostatWS 엔드포인트를 포함하는 CXF 애플리케이션 기반 CXF 파일 기반 CXF를 시작합니다.

참고

다음 **Spring Boot 2 Maven archetypes**가 **OpenShift**에 빌드 및 배포되지 않습니다. 자세한 내용은 릴리스 정보를 참조하십시오.

- **spring-boot-camel-archetype**
- **spring-boot-camel-infinspan-archetype**
- **spring-boot-cxf-jaxrs-archetype**
- **spring-boot-cxf-jaxws-archetype**

이 문제를 해결하려면 다음 빠른 시작에 사용할 **Maven** 프로젝트를 생성한 후 프로젝트의 **Maven pom.xml** 파일을 편집하여 다음 종속성을 추가합니다.

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>2.4.1</version>
  <scope>test</scope>
</dependency>
```

5.6. BOM 파일 FOR SPRING BOOT

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.

중요

사용 중인 **Spring Boot** 버전에 따라 올바른 **Fuse BOM**을 사용하고 있는지 확인합니다 (부팅 1 또는 **Spring Boot 2**).

Fuse BOM for Spring Boot는 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.

5.6.1. BOM 파일 통합

BOM 파일을 Maven 프로젝트에 통합하려면 아래 Spring Boot 2 및 Spring Boot 1의 예와 같이 프로젝트의 pom.xml 파일(또는 상위 POM 파일에서)에 dependencyManagement 요소를 지정합니다.

- Spring Boot 2 BOM
- Spring Boot 1 BOM

```
<?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>7.6.0.fuse-sb2-760028-redhat-00001</fuse.version>
  </properties>

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

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

```
<?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>7.6.0.fuse-760027-redhat-00001</fuse.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-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-hystrix** 구성 요소에 대한 종속성을 추가하려면 **POM**의 종속성 요소에 다음 **XML** 조각을 추가합니다.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hystrix-starter</artifactId>
</dependency>
```

Camel 아티팩트 **ID**를 **-starter suffix Cryostat- Cryostat**로 지정하는 방법에 유의하십시오. 즉 **Camel Hystrix** 구성 요소를 **camel-hystrix -starter** 로 지정합니다. **Camel** 시작 구성 요소는 **Spring Boot** 환경에 최적화된 방식으로 패키지됩니다.

5.7. SPRING BOOT MAVEN 플러그인

Spring Boot Maven 플러그인은 **Spring Boot**에서 제공하며 **Spring Boot** 프로젝트를 빌드하고 실행

하기 위한 개발자 유틸리티입니다.

- 프로젝트 디렉터리에 **mvn** 패키지를 입력하여 **Spring Boot** 애플리케이션에 대한 실행 가능한 **Jar** 패키지를 *빌드* 합니다. 빌드 출력은 **Maven** 프로젝트의 **target/** 하위 디렉터리에 배치됩니다.

- 편의를 위해 **mvn spring-boot:start** 명령을 사용하여 새로 빌드된 애플리케이션을 실행할 수 있습니다.

***Spring Boot Maven** 플러그인을 프로젝트 **POM** 파일에 통합하려면 다음 예와 같이 **pom.xml** 파일의 **project/build/plugins** 섹션에 플러그인 구성을 추가합니다.*

```
<?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>7.6.0.fuse-760027-redhat-00001</fuse.version>

  </properties>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${fuse.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>repackage</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

6장. SPRING BOOT에서 APACHE CAMEL 애플리케이션 실행

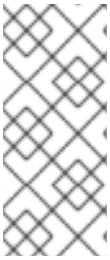
Apache Camel Spring Boot 구성 요소는 Spring Boot에 대한 Camel 컨텍스트를 자동으로 구성합니다. Camel 컨텍스트의 자동 구성에서는 Spring 컨텍스트에서 사용할 수 있는 Camel 경로를 자동으로 감지하고 생산자 템플릿, 소비자 템플릿 및 유형 변환기와 같은 주요 Camel 유틸리티를 빈으로 등록합니다. Apache Camel 구성 요소에는 시작자를 사용하여 Spring Boot 애플리케이션을 개발할 수 있는 Spring Boot Starter 모듈이 포함되어 있습니다.

6.1. CAMEL SPRING BOOT 구성 요소 소개

모든 Camel Spring Boot 애플리케이션에서는 프로젝트의 pom.xml 의 dependencyManagement 요소를 사용하여 종속 항목의 제품화된 버전을 지정해야 합니다. 이러한 종속 항목은 Red Hat Fuse BOM에 정의되어 있으며 특정 Red Hat Fuse 버전에서 지원됩니다. BOM에서 버전을 재정의하지 않도록 추가 시작자의 버전 번호 속성을 생략할 수 있습니다. 자세한 내용은 퀵 스타트 pom 을 참조하십시오.

예제

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



참고

camel-spring-boot param에는 Spring Boot가 Camel 컨텍스트를 자동으로 구성할 수 있도록 해당 종속성을 클래스 경로에 추가할 수 있는 spring.factories 파일이 포함되어 있습니다.

6.2. CAMEL SPRING BOOT 시작 모듈 소개

시작자는 Spring Boot 애플리케이션에서 사용하기 위한 Apache Camel 모듈입니다. 각 Camel 구성 요소에는 camel-xxx-starter 모듈이 있습니다(6.3절. “시작자가 없는 Camel 구성 요소 목록” 섹션에 몇

가지 예외가 나열되어 있음).

시작자는 다음 요구 사항을 충족합니다.

- IDE 툴링과 호환되는 네이티브 Spring Boot 구성 시스템을 사용하여 구성 요소의 자동 구성을 허용합니다.
- 데이터 형식 및 언어의 자동 구성을 허용합니다.
- Spring Boot 로깅 시스템과 통합하기 위해 전송형 로깅 종속성을 관리합니다.
- 추가 종속성을 포함하고 전송 종속성을 정렬하여 작동하는 Spring Boot 애플리케이션 생성 작업을 최소화합니다.

각 시작자는 현재 Spring Boot 릴리스와의 호환성을 확인하는 tests/camel-itest-spring-boot 에서 자체 통합 테스트를 갖습니다.

참고

자세한 내용은 link: Apache Camel Spring-Boot 예제 를 참조하십시오.

6.3. 시작자가 없는 CAMEL 구성 요소 목록

다음 구성 요소에는 호환성 문제로 인해 시작 가능한 모듈이 없습니다.

- Camel-blueprint (OSGi에만 통합)
- Camel-cdi (CDB에만 권장)
- Camel-core-osgi (OSGi에만 해당)

- *Camel- Cryostat (Jee에 대해서만 권장)*
- *Camel-eventadmin (OSGi 전용 통합)*
- *Camel-ibatis (camel-my topologyis-starter 포함)*
- *camel-jclouds*
- *Camel-mina (camel-mina2-starter 포함)*
- *Camel-paxlogging (OSGi 전용 통합)*
- *Camel-quartz (camel-quartz2-starter 포함)*
- *camel-spark-rest*
- *Camel-openapi-java (camel-openapi-java-starter 포함)*

6.4. CAMEL SPRING BOOT 시작 프로그램 사용

Apache Camel은 Spring Boot 애플리케이션 개발을 빠르게 시작할 수 있는 시작 모듈을 제공합니다.

절차

1. *Spring Boot pom.xml 파일에 다음 종속성을 추가합니다.*

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-boot-starter</artifactId>
</dependency>
```

2. *아래 코드 조각에 표시된 대로 Camel 경로가 포함된 클래스를 추가합니다. 이러한 경로가 클*

레스 경로에 추가되면 경로가 자동으로 시작됩니다.

```
package com.example;

import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

   @Override
   public void configure() throws Exception {
      from("timer:foo")
       .to("log:bar");
   }
}
```

3. *선택 사항: Camel이 유지되도록 기본 스레드를 차단하여 유지하려면 다음 중 하나를 수행하십시오.*

   a. *spring-boot-starter-web 종속성을 포함합니다.*

   b. *또는 application.properties 또는 application.yml 파일에 camel.springboot.main-run-controller=true 를 추가합니다.*

   camel.springboot. 속성을 사용하여 application.properties 또는 application.yml 파일에서 Camel 애플리케이션을 사용자 지정할 수 있습니다.*

4. *선택 사항: Quarkus의 ID 이름을 사용하여 사용자 지정 useful을 참조하려면 src/main/resources/application.properties (또는 application.yml) 파일에서 옵션을 구성합니다. 다음 예제에서는ans ID를 사용하여 xslt 구성 요소가 사용자 지정 8080을 참조하는 방법을 보여줍니다.*

   a. *myExtensionFactory ID로 사용자 지정 8080을 참조하십시오.*

   ```
   camel.component.xslt.saxon-extension-functions=myExtensionFactory
   ```

   b. *그런 다음 Spring Boot @Bean 주석을 사용하여 사용자 지정 8080을 만듭니다.*

   ```
   @Bean(name = "myExtensionFactory")
   public ExtensionFunctionDefinition myExtensionFactory() {
      }
   ```

-

또는 jackson ObjectMapper의 경우 camel-jackson data-format에서 다음을 수행하십시오.

```
camel.dataformat.json-jackson.object-mapper=myJacksonMapper
```

6.5. SPRING BOOT의 CAMEL 컨텍스트 자동 구성 정보

Camel Spring Boot 자동 구성에서는 CamelContext 인스턴스를 제공하고 SpringCamelContext 를 생성합니다. 또한 해당 컨텍스트의 종료를 초기화하고 수행합니다. 이 Camel 컨텍스트는 camelContext 8080 이름 아래에 Spring 애플리케이션 컨텍스트에 등록되며 다른 Spring 8080과 마찬가지로 액세스할 수 있습니다. 아래 표시된 대로 camelContext 에 액세스할 수 있습니다.

예제

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

  @Bean
  MyService myService() {
    return new DefaultMyService(camelContext);
  }

}
```

6.6. SPRING BOOT APPLICATIONS에서 CAMEL 경로 자동 감지

Camel 자동 구성은 Spring 컨텍스트에서 모든 RouteBuilder 인스턴스를 수집하고 CamelContext 에 자동으로 삽입합니다. 이렇게 하면 Spring Boot 시작기를 사용하여 새 Camel 경로를 생성하는 프로세스가 간소화됩니다. 다음과 같이 경로를 생성할 수 있습니다.

예제

@Component 주석이 달린 클래스를 classpath에 추가합니다.

```
@Component
```

```
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("jms:invoices").to("file:/invoices");
  }

}
```

또는 @Configuration 클래스에 새 RouteBuilder 8080을 만듭니다.

```
@Configuration
public class MyRouterConfiguration {

  @Bean
  RoutesBuilder myRouter() {
    return new RouteBuilder() {

      @Override
      public void configure() throws Exception {
        from("jms:invoices").to("file:/invoices");
      }

    };
  }

}
```

6.7. CAMEL SPRING BOOT 자동 구성의 CAMEL 속성 구성

Spring Boot 자동 구성은 속성 자리 표시자, OS 환경 변수 또는 Camel 속성을 지원하는 시스템 속성과 같은 Spring Boot 외부 구성에 연결됩니다.

절차

1. application.properties 파일에서 속성을 정의합니다.

```
route.from = jms:invoices
```

또는 Camel 적절한 항목을 시스템 속성으로 설정합니다. 예를 들면 다음과 같습니다.

```
java -Droute.to=jms:processed.invoices -jar mySpringApp.jar
```

2. 다음과 같이 구성된 속성을 Camel 경로에서 자리 표시자로 사용합니다.

```
@Component
public class MyRouter extends RouteBuilder {

 @Override
 public void configure() throws Exception {
  from("{{route.from}}").to("{{route.to}}");
 }

}
```

6.8. 사용자 정의 CAMEL 컨텍스트 구성

Camel Spring Boot 자동 구성으로 생성된 CamelContext 8080에서 작업을 수행하려면 Spring 컨텍스트에 CamelContextConfiguration 인스턴스를 등록합니다.

절차

- 다음과 같이 Spring 컨텍스트에 CamelContextConfiguration 인스턴스를 등록합니다.

```
@Configuration
public class MyAppConfig {

 ...

 @Bean
 CamelContextConfiguration contextConfiguration() {
  return new CamelContextConfiguration() {
   @Override
   void beforeApplicationStart(CamelContext context) {
    // your custom configuration goes here
   }
  };
 }

}
```

CamelContextConfiguration 및 beforeApplicationStart(CamelContext) 방법은 Spring 컨텍스트를 시작하기 전에 호출되므로 이 콜백에 전달되는 CamelContext 인스턴스는 완전히 자동으로 구성됩니다. CamelContextConfiguration 의 많은 인스턴스를 Spring 컨텍스트에 추가할 수 있으며 모든 인스턴스가 실행됩니다.

6.9. 자동 구성된 CAMELCONTEXT에서 CRYOSTAT 비활성화

자동 구성된 CamelContext 에서 Cryostat를 비활성화하려면 기본적으로 Cryostat가 활성화되어 있으므로 camel.springboot.jmxEnabled 속성을 사용할 수 있습니다.

절차

- *application.properties 파일에 다음 속성을 추가하고 false 로 설정합니다.*

```
camel.springboot.jmxEnabled = false
```

6.10. SPRING 관리 빈에 자동 구성된 소비자 및 생산자 템플릿 삽입

Camel 자동 구성은 사전 구성된 ConsumerTemplate 및 ProducerTemplate 인스턴스를 제공합니다. 이를 Spring 관리 빈에 삽입할 수 있습니다.

예제

```
@Component
public class InvoiceProcessor {

 @Autowired
 private ProducerTemplate producerTemplate;

 @Autowired
 private ConsumerTemplate consumerTemplate;
 public void processNextInvoice() {
  Invoice invoice = consumerTemplate.receiveBody("jms:invoices", Invoice.class);
  ...
  producerTemplate.sendBody("netty-http:http://invoicing.com/received/" + invoice.id());
 }

}
```

기본적으로 소비자 템플릿 및 생산자 템플릿에는 엔드포인트 캐시 크기가 1000으로 설정되어 있습니다. 예를 들어 다음 Spring 속성을 원하는 캐시 크기로 설정하여 이러한 값을 변경할 수 있습니다.

```
camel.springboot.consumerTemplateCacheSize = 100
camel.springboot.producerTemplateCacheSize = 200
```

6.11. SPRING 컨텍스트에서 자동 구성된 TYPECONVERTER 정보

Camel 자동 구성에서는 Spring 컨텍스트에서 typeConverter 라는 TypeConverter 인스턴스를 등록합니다.

예제

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public long parseInvoiceValue(Invoice invoice) {
    String invoiceValue = invoice.grossValue();
    return typeConverter.convertTo(Long.class, invoiceValue);
  }

}
```

6.12. SPRING 유형 변환 API 브리지

Spring은 강력한 유형 변환 API 로 구성됩니다. Spring API는 Camel 유형 변환기 API 와 유사합니다. 두 API Camel Spring Boot 간의 유사점으로 인해 Spring 변환 API에 위임하는 브리지 변환기 (SpringTypeConverter)가 자동으로 등록됩니다. 즉, 즉시 사용 가능한 Camel은 Camel과 유사한 Spring Cryostat를 처리합니다.

이를 통해 다음과 같이 Camel TypeConverter API를 사용하여 Camel 및 Spring 컨버터에 모두 액세스할 수 있습니다.

예제

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public UUID parseInvoiceId(Invoice invoice) {
    // Using Spring's StringToUUIDConverter
    UUID id = invoice.typeConverter.convertTo(UUID.class, invoice.getId());
  }

}
```

여기에서 **Spring Boot**는 애플리케이션 컨텍스트에서 사용할 수 있는 **Spring**의 **ConversionService** 인스턴스로 변환을 위임합니다. **ConversionService** 인스턴스를 사용할 수 없는 경우 **Camel Spring Boot** 자동 구성으로 **ConversionService** 인스턴스가 생성됩니다.

6.13. 유형 변환 기능 비활성화

Camel Spring Boot 유형 변환 기능을 비활성화하려면 **camel.springboot.typeConversion** 속성을 **false** 로 설정합니다. 이 속성을 **false** 로 설정하면 자동 구성에서 유형 변환기 인스턴스를 등록하지 않고 **Spring Boot** 유형 변환 **API**로 형식 변환 위임을 활성화하지 않습니다.

절차

- **Camel Spring Boot** 구성 요소의 유형 변환 기능을 비활성화하려면 다음과 같이 **camel.springboot.typeConversion** 속성을 **false** 로 설정합니다.

```
camel.springboot.typeConversion = false
```

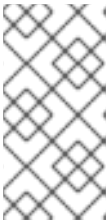
6.14. 자동 구성을 위해 CLASSPATH에 XML 경로 추가

기본적으로 **Camel Spring Boot** 구성 요소는 자동으로 감지되며 **camel** 디렉터리의 **classpath**에 있는 **Camel XML** 경로를 포함합니다. 구성 옵션을 사용하여 디렉터리 이름을 구성하거나 이 기능을 비활성화할 수 있습니다.

절차

- 다음과 같이 **classpath**에서 **Camel Spring Boot XML** 경로를 구성합니다.

```
// turn off
camel.springboot.xmlRoutes = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRoutes = classpath:com/foo/routes/*.xml
```



참고

XML 파일은 **CamelContext** 요소가 아닌 **Camel XML** 경로 요소를 정의해야 합니다. 예를 들면 다음과 같습니다.

-

```
<routes xmlns="http://camel.apache.org/schema/spring">
    <route id="test">
        <from uri="timer://trigger"/>
        <transform>
            <simple>ref:myBean</simple>
        </transform>
        <to uri="log:out"/>
    </route>
</routes>
```

Spring XML 파일 사용

<camelContext>와 함께 Spring XML 파일을 사용하려면 Spring XML 파일 또는 application.properties 파일에서 Camel 컨텍스트를 구성할 수 있습니다. Camel 컨텍스트의 이름을 설정하고 스트림 캐싱을 설정하려면 application.properties 파일에 다음을 추가합니다.

```
camel.springboot.name = MyCamel
camel.springboot.stream-caching-enabled=true
```

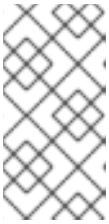
6.15. 자동 구성을 위한 XML REST-DSL 경로 추가

Camel Spring Boot 구성 요소는 camel-rest 디렉터리 아래의 classpath에 추가된 Camel Rest-DSL 경로를 자동으로 탐지하고 포함합니다. 구성 옵션을 사용하여 디렉터리 이름을 구성하거나 이 기능을 비활성화할 수 있습니다.

절차

- **다음과 같이 classpath에서 Camel Spring Boot Rest-DSL XML 경로를 구성합니다.**

```
// turn off
camel.springboot.xmlRests = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRests = classpath:com/foo/rests/*.xml
```



참고

Rest-DSL XML 파일은 CamelContext 요소가 아닌 Camel XML REST 요소를 정의해야 합니다. 예를 들면 다음과 같습니다.

```
<rests xmlns="http://camel.apache.org/schema/spring">
  <rest>
    <post uri="/persons">
      <to uri="direct:postPersons"/>
    </post>
    <get uri="/persons">
```

```
        <to uri="direct:getPersons"/>
      </get>
      <get uri="/persons/{personId}">
        <to uri="direct:getPersionId"/>
      </get>
      <put uri="/persons/{personId}">
        <to uri="direct:putPersionId"/>
      </put>
      <delete uri="/persons/{personId}">
        <to uri="direct:deletePersionId"/>
      </delete>
    </rest>
  </rests>
```

6.16. CAMEL SPRING BOOT로 테스트

Camel이 **Spring Boot**에서 실행되면 **Spring Boot**에 **@Component** 로 주석이 추가된 **Camel** 및 모든 경로가 자동으로 포함됩니다. **Spring Boot**로 테스트할 때 **@ContextConfiguration** 대신 **@SpringBootTest** 를 사용하여 사용할 구성 클래스를 지정합니다.

다른 **RouteBuilder** 클래스에 **Camel** 경로가 여러 개인 경우 **Camel Spring Boot** 구성 요소는 애플리케이션을 실행할 때 이러한 모든 경로를 자동으로 포함합니다. 따라서 하나의 **RouteBuilder** 클래스에서만 경로를 테스트하려면 다음 패턴을 사용하여 활성화할 **RouteBuilders**를 포함하거나 제외할 수 있습니다.

- **java-routes-include-pattern:** 패턴과 일치하는 **RouteBuilder** 클래스를 포함하는 데 사용됩니다.
- **java-routes-exclude-pattern:** 패턴과 일치하는 **RouteBuilder** 클래스를 제외하는 데 사용됩니다. **exclude**는 **include**보다 우선합니다.

절차

1. 다음과 같이 **@SpringBootTest** 주석에 대한 속성으로 단위 테스트 클래스에 포함 또는 제외 패턴을 지정합니다.

```
@RunWith(CamelSpringBootRunner.class)
@SpringBootTest(classes = {MyApplication.class);
  properties = {"camel.springboot.java-routes-include-pattern=**/Foo*"})
public class FooTest {
```

FooTest 클래스에서 포함 패턴은 **/**Foo*** 이며, 이는 **Cryostat** 스타일 패턴을 나타냅니다. 여기에서 패턴은 선행 패키지 이름과 일치하는 이중 별표로 시작됩니다. **/foo*** 는 클래스 이름이

FooRoute로 시작해야 함을 의미합니다.

2. *다음 maven 명령을 사용하여 테스트를 실행합니다.*

```
mvn test -Dtest=FooTest
```

추가 리소스

- *Camel 구성*
- *구성 요소*
- *endpoint*
- *시작하기*

7장. XA 트랜잭션을 사용하여 SPRING BOOT에서 CAMEL 서비스 실행

Spring Boot Camel XA 트랜잭션 빠른 시작은 두 개의 외부 트랜잭션 리소스, A-MQ(A-MQ) 및 데이터베이스(PostgreSQL)에서 XA 트랜잭션을 지원하는 Spring-Boot에서 Camel 서비스를 실행하는 방법을 보여줍니다. 이러한 외부 리소스는 OpenShift에서 제공하며 이 빠른 시작을 실행하기 전에 시작해야 합니다.

7.1. STATEFULSET 리소스

이 빠른 시작에서는 OpenShift StatefulSet 리소스를 사용하여 트랜잭션 관리자의 고유성을 보장하고 트랜잭션 로그를 저장하려면 PersistentVolume이 필요합니다. 애플리케이션은 StatefulSet 리소스에서 스케일링을 지원합니다. 각 인스턴스에는 자체 프로세스 내 복구 관리자가 있습니다. 특수 컨트롤러를 사용하면 애플리케이션이 축소될 때 보류 중인 트랜잭션을 종료하지 않고도 모든 인스턴스를 올바르게 완료할 수 있습니다. 복구 관리자가 종료하기 전에 보류 중인 모든 작업을 플러시할 수 없는 경우 컨트롤러에서 축소 작업을 롤백합니다. 이 빠른 시작에서는 Spring Bootnaana 복구 컨트롤러를 사용합니다.

7.2. SPRING BOOT NARAYANA 복구 컨트롤러

Spring Boot Narayana 복구 컨트롤러를 사용하면 종료 전에 보류 중인 트랜잭션을 정리하여 StatefulSet의 축소 단계를 정상적으로 처리할 수 있습니다. 축소 작업이 실행되고 종료 후 Pod가 정리되지 않으면 이전 복제본 수가 복원되므로 축소 작업을 효과적으로 취소합니다.

StatefulSet의 모든 Pod는 StatefulSet에 속하는 각 Pod의 종료 상태를 저장하는 데 사용되는 공유 볼륨에 액세스해야 합니다. StatefulSet의 pod-0은 주기적으로 상태를 확인하고 불일치가 있는 경우 StatefulSet을 올바른 크기로 스케일링합니다.

복구 컨트롤러가 작동하려면 현재 네임스페이스에 대한 권한을 편집해야 합니다(OpenShift에 게시된 리소스 세트에 역할 바인딩이 포함되어 있음). CLUSTER_RECOVERY_ENABLED 환경 변수를 사용하여 복구 컨트롤러를 비활성화할 수 있습니다. 이 경우 서비스 계정에 특별한 권한이 필요하지 않지만 축소 작업에서는 사전 통지 없이 종료된 Pod에 보류 중인 트랜잭션이 남길 수 있습니다.

7.3. SPRING BOOT NARAYANA 복구 컨트롤러 구성

다음 예제에서는 복구 컨트롤러를 사용하여 OpenShift에서 작동하도록narayana를 구성하는 방법을 보여줍니다.

절차

1. 샘플 application.properties 파일입니다. Kubernetes yaml 설명자에서 다음 옵션을 교체합니다.

```
# Cluster
cluster.nodename=1
cluster.base-dir=./target/tx

# Transaction Data
spring.jta.transaction-manager-id=${cluster.nodename}
spring.jta.log-dir=${cluster.base-dir}/store/${cluster.nodename}

# Narayana recovery settings
snowdrop.narayana.openshift.recovery.enabled=true
snowdrop.narayana.openshift.recovery.current-pod-name=${cluster.nodename}
# You must enable resource filtering in order to inject the Maven artifactId
snowdrop.narayana.openshift.recovery.statefulset=${project.artifactId}
snowdrop.narayana.openshift.recovery.status-dir=${cluster.base-dir}/status
```

2. *종료와 관련된 트랜잭션 및 정보를 모두 저장하려면 공유 볼륨이 필요합니다. 다음과 같이* ***StatefulSet yaml*** *설명자에 마운트할 수 있습니다.*

```
apiVersion: apps/v1beta1
kind: StatefulSet
#...
spec:
#...
  template:
#...
    spec:
      containers:
      - env:
        - name: CLUSTER_BASE_DIR
          value: /var/transaction/data
          # Override CLUSTER_NODENAME with Kubernetes Downward API (to use `pod-0`,
`pod-1` etc. as tx manager id)
        - name: CLUSTER_NODENAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
#...
        volumeMounts:
        - mountPath: /var/transaction/data
          name: the-name-of-the-shared-volume
#...
```

Camel Extension for Spring Bootnaana Recovery Controller

Camel*이* ***Spring Boot*** *애플리케이션 컨텍스트에서 발견되면 보류 중인 모든 트랜잭션을 플러시하기 전에* ***Camel*** *컨텍스트가 자동으로 중지됩니다.*

7.4. OPENSHIFT에서 CAMEL SPRING BOOT XA 빠른 시작 실행

*다음 절차에서는 실행 중인 단일 노드 **OpenShift** 클러스터에서 퀵스타트를 실행하는 방법을 보여줍니다.*

절차

1. ***Camel Spring Boot XA** 프로젝트를 다운로드합니다.*

```
git clone --branch spring-boot-camel-xa-1.0.0.fuse-760022-redhat-00001
https://github.com/jboss-fuse/spring-boot-camel-xa
```

2. ***spring-boot-camel-xa** 디렉토리로 이동하여 다음 명령을 실행합니다.*

```
mvn clean install
```

3. ***OpenShift** 서버에 로그인합니다.*

```
oc login -u developer -p developer
```

4. ***test** 라는 새 프로젝트 네임스페이스를 생성합니다(아직 존재하지 않는다고 가정).*

```
oc new-project test
```

   ***test** 프로젝트 네임스페이스가 이미 있는 경우 해당 네임스페이스로 전환합니다.*

```
oc project test
```

5. *종속 항목을 설치합니다.*

   - *사용자 이름으로 사용자 이름 및 암호 **Thepassword1!** 를 사용하여 **postgresql** 를 설치합니다.*

```
oc new-app --param=POSTGRESQL_USER=theuser --
param=POSTGRESQL_PASSWORD='Thepassword1!' --
env=POSTGRESQL_MAX_PREPARED_TRANSACTIONS=100 --template=postgresql-
persistent
```

   - *사용자 이름으로 사용자 이름 및 암호 **Thepassword1!** 를 사용하여 **A-MQ** 브로커를 설*

치합니다.

```
oc new-app --param=MQ_USERNAME=theuser --
param=MQ_PASSWORD='Thepassword1!' --template=amq63-persistent
```

6. *트랜잭션 로그에 대한 영구 볼륨 클레임을 생성합니다.*

```
oc create -f persistent-volume-claim.yml
```

7. *빠른 시작을 빌드하고 배포합니다.*

```
mvn fabric8:deploy -P openshift
```

8. *원하는 복제본 수까지 확장합니다.*

```
oc scale statefulset spring-boot-camel-xa --replicas 3
```

*참고: **Pod** 이름은 트랜잭션 관리자 **ID(spring.jta. Cryostat-manager-id** 속성)로 사용됩니다. 현재 구현은 트랜잭션 관리자 **ID**의 길이도 제한합니다. 다음 사항에 유의하십시오.*

- ***StatefulSet**의 이름은 트랜잭션 시스템의 식별자이므로 변경할 수 없습니다.*
- *모든 **Pod** 이름의 길이가 **23**자보다 작거나 같도록 **StatefulSet**의 이름을 지정해야 합니다. **Pod** 이름은 **<statefulset-name>-0, <statefulset-name>-1** 등을 사용하여 **OpenShift**에 의해 생성됩니다. **Narayana**는 동일한 **ID**를 가진 여러 복구 관리자가 없도록하기 위해 최선을 다하므로 **Pod** 이름이 제한보다 길면 마지막 **23** 바이트가 트랜잭션 관리자 **ID**로 사용됩니다 (와 같은 일부 문자를 제거 한 후).*

9. *빠른 시작이 실행되면 다음 명령을 사용하여 기본 서비스 **URL**을 가져옵니다.*

```
NARAYANA_HOST=$(oc get route spring-boot-camel-xa -o jsonpath={.spec.host})
```

*7.5. 성공적인 **XA** 트랜잭션 테스트*

*다음 워크플로는 성공적인 **XA** 트랜잭션을 테스트하는 방법을 보여줍니다.*

절차

1. ***audit_log** 테이블의 메시지 목록을 가져옵니다.*

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

2. *처음에는 목록이 비어 있습니다. 이제 첫 번째 요소를 삽입할 수 있습니다.*

```
curl -w "\n" -X POST http://$NARAYANA_HOST/api/?entry=hello
```

잠시 기다린 후 새 목록을 받으십시오.

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

3. *새 목록에는 **hello** 및 **hello-ok** 이라는 두 개의 메시지가 포함되어 있습니다. **hello-ok** 은 메시지가 발신 큐로 전송된 다음 기록되었음을 확인합니다. 여러 메시지를 추가하고 로그를 볼 수 있습니다.*

*7.6. 실패한 **XA** 트랜잭션 테스트*

*다음 워크플로우에서는 실패한 **XA** 트랜잭션을 테스트하는 방법을 보여줍니다.*

절차

1. ***fail** 이라는 메시지를 보냅니다.*

```
curl -w "\n" -X POST http://$NARAYANA_HOST/api/?entry=fail
```

2. *잠시 기다린 후 새 목록을 받으십시오.*

```
curl -w "\n" http://$NARAYANA_HOST/api/
```

3. *이 메시지는 경로 끝에 예외를 생성하므로 트랜잭션이 항상 롤백됩니다. **audit_log** 표에서 메시지의 추적을 찾을 수 없습니다.*

8장. CAMEL 애플리케이션과 A-MQ 브로커 통합

이 튜토리얼에서는 A-MQ 이미지를 사용하여 빠른 시작을 배포하는 방법을 보여줍니다.

8.1. SPRING BOOT CAMEL A-MQ 빠른 시작 빌드 및 배포

이 빠른 시작에서는 Spring Boot 애플리케이션을 AMQ Online에 연결하고 OpenShift에서 Fuse를 사용하여 두 Camel 경로 간에 JMS 메시징을 사용하는 방법을 보여줍니다.

사전 요구 사항

- AMQ Online이 배포 및 실행 중인지 확인합니다. OpenShift에 AMQ Online을 설치하려면 OpenShift에 AMQ Online 설치 및 구성을 참조하십시오.
- OpenShift가 올바르게 실행 중이고 Fuse 이미지 스트림이 이미 OpenShift에 설치되어 있는지 확인합니다. 관리자용 시작하기를 참조하십시오.
- Maven 리포지토리가 fuse용으로 구성되었는지 확인합니다. Maven 리포지토리 구성을 참조하십시오.

절차

1. OpenShift 서버에 개발자로 로그인합니다.

```
oc login -u developer -p developer
```

2. openshift 프로젝트에 로그인합니다.

```
oc project openshift
```

3. Maven 워크플로를 사용하여 빠른 시작 프로젝트를 생성합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
-
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-760024-redhat-00001/archetypes-catalog-2.2.0.fuse-760024-redhat-00001-archetype-catalog.xml \
```

```
-DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
-DarchetypeArtifactId=spring-boot-camel-amq-archetype \
-DarchetypeVersion=2.2.0.fuse-760024-redhat-00001
```

4. ***archetype** 플러그인은 대화형 모드로 전환되어 나머지 필드를 입력하라는 메시지를 표시합니다.*

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-spring-boot-camel-amq
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-spring-boot-camel-amq
version: 1.0-SNAPSHOT
package: org.example.fis
Y: :Y
```

*메시지가 표시되면 **groupId** 값으로 **org.example.fis** 를 입력하고 **artifactId** 값으로 **fuse76-spring-boot-camel-amq** 를 입력합니다. 나머지 필드의 기본값을 수락합니다.*

5. *빠른 시작 디렉터리 **fuse76-spring-boot-camel-amq** 로 이동합니다.*

```
cd fuse76-spring-boot-camel-amq
```

6. *다음 명령을 실행하여 구성 파일을 **AMQ Online**에 적용합니다. 이러한 구성 파일은 관리자 권한이 있는 **AMQ Online** 사용자와 큐를 생성합니다.*

```
oc login -u system:admin
oc apply -f src/main/resources/k8s
```

7. ***mvn** 명령을 실행하여 **OpenShift** 서버에 퀵스타트를 배포합니다.*

```
mvn fabric8:deploy -Popenshift
```

8. *빠른 시작이 성공적으로 실행 중인지 확인하려면 다음을 수행하십시오.*

   a. *브라우저에서 **OpenShift** 웹 콘솔(**https://OPENSHIFT_IP_ADDR**, **OPENSHIFT_IP_ADDR**을 클러스터의 **IP** 주소로 교체)로 이동하고 인증 정보(예: 사용자 이름 **developer** 및 암호, **developer** 사용)로 콘솔에 로그인합니다.*

b.

*왼쪽 패널에서 홈을 확장한 다음 상태를 클릭하여 **openshift** 프로젝트의 **Project Status** 페이지를 확인합니다.*

c.

***fuse76-spring-boot-camel-amq** 를 클릭하여 퀵 스타트에 대한 개요 정보 페이지를 확인합니다.*

d.

왼쪽 패널에서 워크로드를 확장합니다.

e.

*포드 를 클릭한 다음 **fuse76-spring-boot-camel-amq-xxxxx** 를 클릭합니다. 빠른 시작에 대한 **Pod** 세부 정보가 표시됩니다.*

f.

로그를 클릭하여 애플리케이션의 로그를 확인합니다.

출력에 메시지가 성공적으로 전송됨이 표시됩니다.

```
10:17:59.825 [Camel (camel) thread #10 - timer://order] INFO  generate-order-route - Generating order order1379.xml
10:17:59.829 [Camel (camel) thread #8 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Sending order order1379.xml to the UK
10:17:59.829 [Camel (camel) thread #8 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Done processing order1379.xml
10:18:02.825 [Camel (camel) thread #10 - timer://order] INFO  generate-order-route - Generating order order1380.xml
10:18:02.829 [Camel (camel) thread #7 - JmsConsumer[incomingOrders]] INFO  jms-cbr-route - Sending order order1380.xml to another country
10:18:02.829 [Camel (camel) thread #7 - JmsConsumer[incomingOrders]] INFO jms-cbr-route - Done processing order1380.xml
```

9.

*웹 인터페이스에서 경로를 보려면 **Open Java Console** 을 클릭하고 **AMQ** 큐에서 메시지를 확인합니다.*

9장. SPRING BOOT와 KUBERNETES 통합

Spring Cloud Kubernetes 플러그인을 사용하면 현재 Spring Boot 및 Kubernetes의 다음 기능을 통합할 수 있습니다.

- Spring Boot Externalized Configuration
- Kubernetes ConfigMap
- Kubernetes 시크릿

9.1. SPRING 부팅 외부 구성

Spring Boot에서 외부화된 구성은 외부 소스의 구성 값을 Java 코드에 삽입할 수 있는 메커니즘입니다. Java 코드에서는 일반적으로 @Value 주석(단일 필드에 삽입) 또는 @ConfigurationProperties 주석(Java Cryostat 클래스의 여러 속성에 삽입)에 주석을 달아 삽입할 수 있습니다.

구성 데이터는 다양한 소스(또는 속성소스)에서 가져올 수 있습니다. 특히 구성 속성은 프로젝트의 application.properties 파일(또는 원하는 경우 application.yaml 파일)에 설정되는 경우가 많습니다.

9.1.1. Kubernetes ConfigMap

Kubernetes ConfigMap 은 배포된 애플리케이션에 구성 데이터를 제공할 수 있는 메커니즘입니다. ConfigMap 오브젝트는 일반적으로 YAML 파일에 정의되어 있으며, 그런 다음 Kubernetes 클러스터에 업로드되어 배포된 애플리케이션에서 구성 데이터를 사용할 수 있습니다.

9.1.2. Kubernetes 시크릿

Kubernetes 시크릿 은 배포된 애플리케이션에 중요한 데이터(암호, 인증서 등)를 제공하는 메커니즘입니다.

9.1.3. Spring Cloud Kubernetes 플러그인

Spring Cloud Kubernetes 플러그인은 Kubernetes와 Spring Boot 간의 통합을 구현합니다. 기본적으로 Kubernetes API를 사용하여 ConfigMap의 구성 데이터에 액세스할 수 있습니다. 그러나 Kubernetes ConfigMap을 Spring Boot 외부화된 구성 메커니즘과 직접 통합하여 Kubernetes

ConfigMap이 Spring Boot 구성의 대체 속성 소스로 작동하는 것이 훨씬 더 편리합니다. 기본적으로 Spring Cloud Kubernetes 플러그인이 제공하는 것입니다.

9.1.4. Kubernetes 통합을 사용하여 Spring Boot 활성화

pom.xml 파일에 Maven 종속성으로 추가하여 Kubernetes 통합을 활성화할 수 있습니다.

절차

1. *Spring Boot Maven 프로젝트의 pom.xml 파일에 다음 Maven 종속성을 추가하여 Kubernetes 통합을 활성화합니다.*

```
<project ...>
 ...
 <dependencies>
  ...
  <dependency>
   <groupId>io.fabric8</groupId>
   <artifactId>spring-cloud-kubernetes-core</artifactId>
  </dependency>
  ...
 </dependencies>
 ...
</project>
```

2. *통합을 완료하려면,*

- *Java 소스 코드에 몇 가지 주석을 추가합니다.*
- *Kubernetes ConfigMap 오브젝트 생성*
- *애플리케이션이 ConfigMap 오브젝트를 읽을 수 있도록 OpenShift 서비스 계정 권한을 수정합니다.*

추가 리소스

- *자세한 내용은 ConfigMap 속성 소스에 대한 튜토리얼 실행을 참조하십시오.*

9.2. CONFIGMAP 속성 소스에 대한 튜토리얼 실행

다음 튜토리얼을 사용하면 **Kubernetes** 시크릿 및 **ConfigMap** 설정을 실험할 수 있습니다. **Kubernetes** 통합 활성화에 설명된 대로 **Spring Cloud Kubernetes** 플러그인을 활성화하여 **Kubernetes** 구성 오브젝트를 **Spring Boot Externalized Configuration**과 통합합니다.

9.2.1. Spring Boot Camel Config 빠른 시작 실행

다음 튜토리얼은 **Kubernetes Secrets** 및 **ConfigMap**을 설정할 수 있는 **Spring -boot-camel-config-archetype Maven archetype**을 기반으로 합니다.

절차

1. 새 쉘 프롬프트를 열고 다음 **Maven** 명령을 입력하여 간단한 **Camel Spring Boot** 프로젝트를 생성합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-760024-redhat-00001/archetypes-catalog-2.2.0.fuse-760024-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=spring-boot-camel-config-archetype \
  -DarchetypeVersion=2.2.0.fuse-760024-redhat-00001
```

archetype 플러그인은 대화형 모드로 전환되어 나머지 필드를 입력하라는 메시지를 표시합니다.

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-configmap
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-configmap
version: 1.0-SNAPSHOT
package: org.example.fis
Y: : Y
```

메시지가 표시되면 **groupId** 값으로 **org.example.fis** 를 입력하고 **artifactId** 값으로 **fuse76-configmap** 을 입력합니다. 나머지 필드의 기본값을 수락합니다.

2. **OpenShift**에 로그인하고 애플리케이션을 배포할 **OpenShift** 프로젝트로 전환합니다. 예를 들어 **developer** 사용자로 로그인하고 **openshift** 프로젝트에 배포하려면 다음 명령을 입력합니다.

```
oc login -u developer -p developer
oc project openshift
```

3. 명령줄에서 새 **fuse76-configmap** 프로젝트의 디렉터리로 변경하고 이 애플리케이션에 대한 **Secret** 오브젝트를 생성합니다.

```
cd fuse76-configmap
oc create -f sample-secret.yml
```

참고

애플리케이션을 배포하기 전에 **Secret** 오브젝트를 생성해야 합니다. 그러지 않으면 배포된 컨테이너가 **Secret**을 사용할 수 있을 때까지 대기 상태가 됩니다. 나중에 **Secret**을 생성하면 컨테이너가 대기 상태가 됩니다. **Secret Object** 설정 방법에 대한 자세한 내용은 시크릿 설정을 참조하십시오.

4. 빠른 시작 애플리케이션을 빌드하고 배포합니다. **fuse76-configmap** 프로젝트의 최상위 수준에서 다음을 입력합니다.

```
mvn fabric8:deploy -Popenshift
```

5. 다음과 같이 애플리케이션 로그를 확인합니다.

a. 브라우저에서 **OpenShift** 웹 콘솔로 이동하고(**https://OPENSHIFT_IP_ADDR**, **OPENSHIFT_IP_ADDR** 을 클러스터의 **IP** 주소로 교체) 인증 정보로 로그인합니다(예: 사용자 이름 **developer** 및 암호, 개발자).

b. 왼쪽 패널에서 홈을 확장합니다. **Status** (상태)를 클릭하여 프로젝트 상태 페이지를 확인합니다. 선택한 네임스페이스(예: **openshift**)의 기존 애플리케이션이 모두 표시됩니다.

c. **fuse76-configmap** 을 클릭하여 퀵 스타트에 대한 개요 정보 페이지를 확인합니다.

d. 왼쪽 패널에서 워크로드를 확장합니다.

e. 포드 를 클릭한 다음 **fuse76-configmap-xxxx** 를 클릭합니다. 애플리케이션의 **Pod** 세부 정보가 표시됩니다.

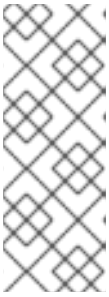
f. *로그 탭을 클릭하여 애플리케이션 로그를 확인합니다.*

6. ***src/main/resources/application.properties** 에 구성된 기본 수신자 목록은 생성된 메시지를 **direct:async-queue** 및 **direct:file** 이라는 두 개의 더미 엔드포인트로 보냅니다. 이로 인해 다음과 같은 메시지가 애플리케이션 로그에 기록됩니다.*

```
5:44:57.376 [Camel (camel) thread #0 - timer://order] INFO  generate-order-route -
Generating message message-44, sending to the recipient list
15:44:57.378 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ---->
message-44 pushed to an async queue (simulation)
15:44:57.379 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> Using
username 'myuser' for the async queue
15:44:57.380 [Camel (camel) thread #0 - timer://order] INFO  target-route--file - ---->
message-44 written to a file
```

7. ***ConfigMap** 오브젝트를 사용하여 **fuse76-configmap** 애플리케이션의 구성을 업데이트하려면 **fuse76-configmap** 애플리케이션 권한을 부여하여 **OpenShift ApiServer**에서 데이터를 볼 수 있는 권한을 부여해야 합니다. 다음 명령을 입력하여 **fuse76-configmap** 애플리케이션의 서비스 계정에 대한 보기 권한을 부여합니다.*

```
oc policy add-role-to-user view system:serviceaccount:openshift:qs-camel-config
```



참고

*서비스 계정은 **system:serviceaccount:PROJECT_NAME:SERVICE_ACCOUNT_NAME** 구문을 사용하여 지정됩니다. **fis-config** 배포 설명자는 **qs-camel-config** 로 **SERVICE_ACCOUNT_NAME** 을 정의합니다.*

8. *작동 중인 실시간 다시 로드 기능을 보려면 다음과 같이 **ConfigMap** 오브젝트를 생성합니다.*

```
oc create -f sample-configmap.yml
```

*새 **ConfigMap**은 실행 중인 애플리케이션에서 **Camel** 경로의 수신자 목록을 재정의하여 생성된 메시지를 **direct:async-queue,direct:file, direct:mail. ConfigMap** 오브젝트에 대한 자세한 내용은 **ConfigMap** 설정을 참조하십시오. 이로 인해 다음과 같은 메시지가 애플리케이션 로그에 기록됩니다.*

```
16:25:24.121 [Camel (camel) thread #0 - timer://order] INFO  generate-order-route -
Generating message message-9, sending to the recipient list
16:25:24.124 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ---->
message-9 pushed to an async queue (simulation)
```

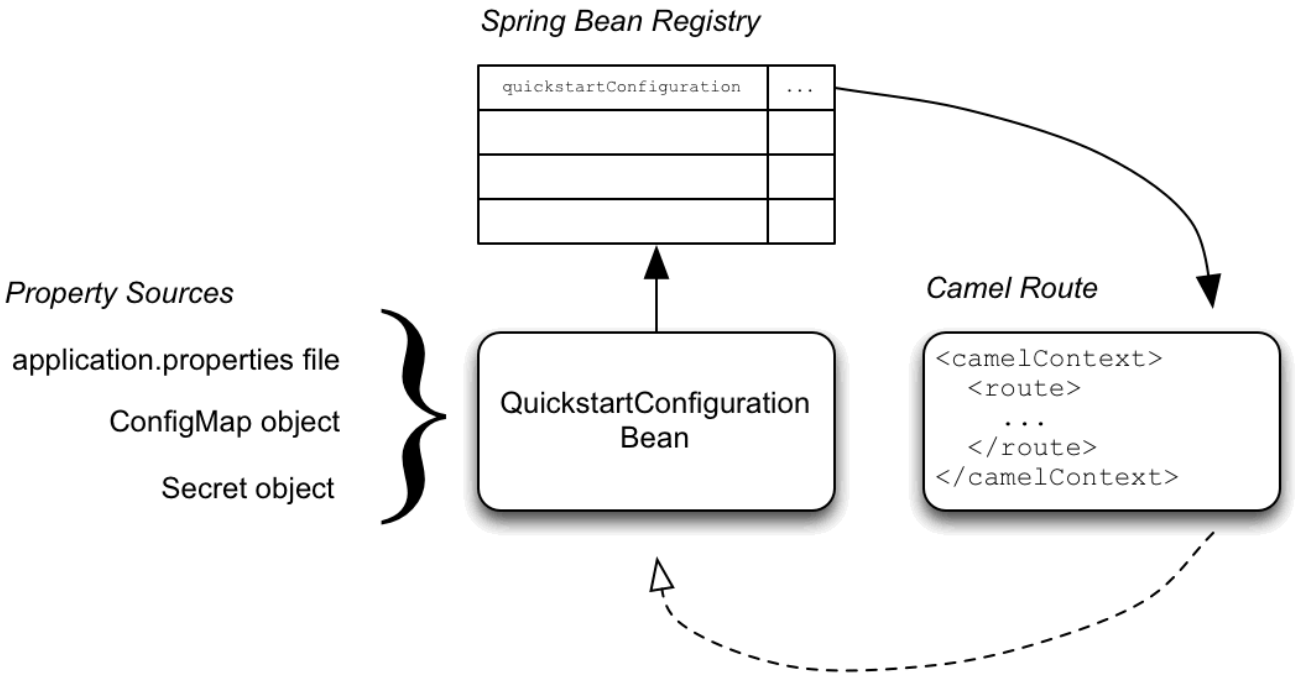
```
16:25:24.125 [Camel (camel) thread #0 - timer://order] INFO  target-route-queue - ----> Using username 'myuser' for the async queue
16:25:24.125 [Camel (camel) thread #0 - timer://order] INFO  target-route--file - ----> message-9 written to a file (simulation)
16:25:24.126 [Camel (camel) thread #0 - timer://order] INFO  target-route--mail - ----> message-9 sent via mail
```

9.2.2. 구성 속성 metrics

구성 속성^n은 삽입을 통해 구성 설정을 수신할 수 있는 일반 **Java 8080**입니다. **Java** 코드와 외부 구성 메커니즘 간의 기본 인터페이스를 제공합니다.

외부 구성 및 **Cryostat** 레지스트리

다음 이미지는 **Spring Boot Externalized Configuration**이 **spring-boot-camel-config quickstart**에서 작동하는 방법을 보여줍니다.



구성 메커니즘에는 다음과 같은 주요 부분이 있습니다.

속성 소스

구성에 삽입하기 위한 속성 설정을 제공합니다. 기본 속성 소스는 애플리케이션의 **application.properties** 파일이며, 선택적으로 **ConfigMap** 오브젝트 또는 **Secret** 오브젝트로 덮어쓸 수 있습니다.

구성 속성 metrics

속성 소스에서 configuraton 업데이트를 수신합니다. 구성 속성 8080은 @Configuration 및 @ConfigurationProperties 주석으로 데코레이팅된 Java 8080입니다.

Spring 8080 레지스트리

필수 주석을 사용하면 구성 속성 metrics이 Spring 8080 레지스트리에 등록됩니다.

Camel 8080 레지스트리와 통합

Camel 8080 레지스트리는 Spring 8080 레지스트리와 자동으로 통합되어 Camel 경로에서 registered Spring 빈을 참조할 수 있습니다.

빠른 시작Configuration 클래스

fuse76-configmap 프로젝트의 구성 속성 sum은 다음과 같이 QuickstartConfiguration Java 클래스(src/main/java/org/example/fis/ 디렉터리에 있음)로 정의됩니다.

```
package org.example.fis;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration (1)
@ConfigurationProperties(prefix = "quickstart") (2)
public class QuickstartConfiguration {

    /**
     * A comma-separated list of routes to use as recipients for messages.
     */
    private String recipients; (3)

    /**
     * The username to use when connecting to the async queue (simulation)
     */
    private String queueUsername; (4)

    /**
     * The password to use when connecting to the async queue (simulation)
     */
    private String queuePassword; (5)

    // Setters and Getters for Bean properties
    // NOT SHOWN
    ...
}
```

(1)

@Configuration 주석을 사용하면 빠른 시작 클래스가 ID가 있는 빈으로 Spring에 인스턴스화되고 등록되며, quickstartConfiguration 클래스가 생성됩니다. 이렇게 하면 Camel에서 빈에 자동으로 액세스할 수 있습니다. 예를 들어 target-route-queue 경로는 Camel 구문 ${bean:quickstartConfiguration?method=getQueueUsername} 을 사용하여 queueUserName 속성에 액세스할 수 있습니다.

2

@ConfigurationProperties 주석은 속성 소스에서 속성 값을 정의할 때 사용해야 하는 접두사인 quickstart 를 정의합니다. 예를 들어 속성 파일은 recipient 속성을 quickstart.recipients 로 참조합니다.For example, a properties file would reference the recipient property as quickstart.recipients.

3

recipient 속성은 속성 소스에서 삽입할 수 있습니다.

4

queueUsername 속성은 속성 소스에서 삽입할 수 있습니다.

5

queuePassword 속성은 속성 소스에서 삽입할 수 있습니다.

9.2.3. 보안 설정

이 빠른 시작의 Kubernetes 시크릿은 추가 필수 단계 중 하나와 별도로 표준 방식으로 설정됩니다. Spring Cloud Kubernetes 플러그인은 시크릿의 마운트 경로로 구성되어야 런타임에 보안을 읽을 수 있습니다. 보안을 설정하려면 다음을 수행합니다.

1. *샘플 시크릿 오브젝트 생성*
2. *시크릿의 볼륨 마운트 구성*
3. *Secret 속성을 읽을 수 있도록 spring-cloud-kubernetes 구성*

샘플 Secret 오브젝트

빠른 시작 프로젝트는 다음과 같이 샘플 Secret, sample-secret.yml 을 제공합니다. Secret 오브젝트의 속성 값은 항상 base64로 인코딩됩니다(base64 명령줄 유틸리티 사용). Pod의 파일 시스템에 보안이

마운트되면 값이 자동으로 일반 텍스트로 다시 디코딩됩니다.

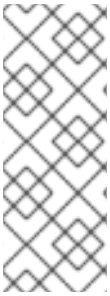
sample-secret.yml 파일

```
apiVersion: v1
kind: Secret
metadata: (1)
  name: camel-config
type: Opaque
data:
  # The username is 'myuser'
  quickstart.queue-username: bXI1c2VyCg== (2)
  quickstart.queue-password: MWYyZDFlMmU2N2Rm (3)
```

(1) **metadata.name:** 시크릿을 식별합니다. **OpenShift** 시스템의 다른 부분에서는 이 **ID**를 사용하여 보안을 참조합니다.

(2) **Quickstart.queue-username: quickstartConfiguration VLAN**의 **queueUsername** 속성에 삽입해야 합니다. 값은 **base64**로 인코딩 되어야 합니다.

(3) **Quickstart.queue-password: quickstartConfiguration 8080**의 **queuePassword** 속성에 삽입해야 합니다. 값은 **base64**로 인코딩 되어야 합니다.



참고

Kubernetes에서는 **CamelCase**에서 속성 이름을 정의할 수 없습니다(속성 이름이 모두 소문자여야 함). 이 제한을 해결하려면 **Spring Boot**가 **queueUsername** 과 일치하는 하이픈이 지정된 형식의 **queue-username** 을 사용합니다. 이는 외부화된 구성에 대한 **Spring Boot**의 완화된 바인딩 규칙을 활용합니다.

시크릿의 볼륨 마운트 구성

보안을 볼륨 마운트로 구성하여 런타임에 보안을 로드하도록 애플리케이션을 구성해야 합니다. 애플리케이션이 시작되면 **Secret** 속성이 파일 시스템의 지정된 위치에서 사용할 수 있게 됩니다. 애플리케이

션의 deployment.yml 파일은 Secret의 볼륨 마운트를 정의하는 src/main/fabric8/ 디렉터리에 있습니다.

deployment.yml 파일

```
spec:
  template:
    spec:
      serviceAccountName: "qs-camel-config"
      volumes: (1)
        - name: "camel-config"
          secret:
            # The secret must be created before deploying this application
            secretName: "camel-config"
      containers:
        -
          volumeMounts: (2)
            - name: "camel-config"
              readOnly: true
              # Mount the secret where spring-cloud-kubernetes is configured to read it
              # see src/main/resources/bootstrap.yml
              mountPath: "/etc/secrets/camel-config"
          resources:
#           requests:
#             cpu: "0.2"
#             memory: 256Mi
#           limits:
#             cpu: "1.0"
#             memory: 256Mi
          env:
            - name: SPRING_APPLICATION_JSON
              value: '{"server":{"undertow":{"io-threads":1, "worker-threads":2 }}}'
```

(1) *volumes 섹션에서 배포는 camel-config 라는 시크릿을 참조하는 camel-config 라는 새 볼륨을 선언합니다.*

(2) *volumeMounts 섹션에서 배포는 camel-config 볼륨을 참조하는 새 볼륨 마운트를 선언하고 Secret 볼륨을 Pod 파일 시스템의 경로 /etc/secrets/camel-config 에 마운트하도록 지정합니다.*

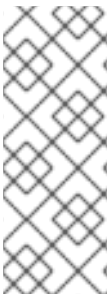
Secret 속성을 읽을 수 있도록 spring-cloud-kubernetes 구성

시크릿을 Spring Boot 외부화된 구성과 통합하려면 Spring Cloud Kubernetes 플러그인을 시크릿의 마운트 경로로 구성해야 합니다. Spring Cloud Kubernetes는 지정된 위치에서 시크릿을 읽고 속성 소스로 Spring Boot에서 사용할 수 있도록 합니다. Spring Cloud Kubernetes 플러그인은 quickstart 프로젝트의 src/main/resources 아래에 있는 bootstrap.yml 파일의 설정으로 구성됩니다.

bootstrap.yml 파일

```
# Startup configuration of Spring-cloud-kubernetes
spring:
  application:
    name: camel-config
  cloud:
    kubernetes:
      reload:
        # Enable live reload on ConfigMap change (disabled for Secrets by default)
        enabled: true
      secrets:
        paths: /etc/secrets/camel-config
```

spring.cloud.kubernetes.secrets.paths 속성은 Pod의 보안 볼륨 마운트 경로 목록을 지정합니다.



참고

bootstrap.properties 파일(또는 bootstrap.yml 파일)은 application.properties 파일과 유사하게 작동하지만 애플리케이션 시작의 이전 단계에서 로드됩니다. bootstrap.properties 파일에서 Spring Cloud Kubernetes 플러그인과 관련된 속성을 설정하는 것이 더 안정적입니다.

9.2.4. ConfigMap 설정

ConfigMap 오브젝트를 생성하고 뷰 권한을 적절하게 설정하는 것 외에도 Spring Cloud Kubernetes와의 통합을 위해서는 ConfigMap의 metadata.name 값과 프로젝트의 bootstrap.yml 파일에 구성된 spring.application.name 속성 값과 일치해야 합니다. ConfigMap을 설정하려면 다음을 수행합니다.

- 샘플 ConfigMap 오브젝트 생성

- 보기 권한 설정
- Spring Cloud Kubernetes 플러그인 구성

샘플 ConfigMap 오브젝트

빠른 시작 프로젝트는 샘플 ConfigMap인 sample-configmap.yml 을 제공합니다.

```
kind: ConfigMap
apiVersion: v1
metadata: (1)
  # Must match the 'spring.application.name' property of the application
  name: camel-config
data:
  application.properties: | (2)
    # Override the configuration properties here
    quickstart.recipients=direct:async-queue,direct:file,direct:mail (3)
```

(1) metadata.name: ConfigMap을 식별합니다. OpenShift 시스템의 다른 부분에서는 이 ID를 사용하여 ConfigMap을 참조합니다.

(2) data.application.properties: 이 섹션에는 애플리케이션과 함께 배포된 원래 application.properties 파일의 설정을 재정의할 수 있는 속성 설정이 나열됩니다.

(3) Quickstart.recipients: quickstartConfiguration VLAN의 받는 사람 속성에 삽입해야 합니다.

보기 권한 설정

Secret의 deployment.yml 파일에 표시된 대로 프로젝트의 deployment.yml 파일에서 serviceAccountName 이 qs-camel-config 로 설정됩니다. 따라서 빠른 시작 애플리케이션에 대한 보기 권한을 활성화하려면 다음 명령을 입력해야 합니다(테스트 프로젝트 네임스페이스에 배포된다고 가정).

```
oc policy add-role-to-user view system:serviceaccount:test:qs-camel-config
```

Spring Cloud Kubernetes 플러그인 구성

Spring Cloud Kubernetes 플러그인은 bootstrap.yml 파일의 다음 설정으로 구성됩니다.

spring.application.name

이 값은 ConfigMap 오브젝트의 metadata.name 과 일치해야 합니다(예: quickstart 프로젝트의 sample-configmap.yml 에 정의된 대로). 기본값은 application 입니다.

spring.cloud.kubernetes.reload.enabled

이를 true 로 설정하면 ConfigMap 오브젝트의 동적 재로드가 활성화됩니다.

지원되는 속성에 대한 자세한 내용은 PropertySource Reload Configuration Properties 을 참조하십시오.

9.3. CONFIGMAP PROPERTYSOURCE 사용

Kubernetes에는 구성을 애플리케이션에 전달하기 위한 ConfigMap 의 개념이 있습니다. Spring 클라우드 Kubernetes 플러그인은 ConfigMap 과 통합을 제공하여 Spring Boot에서 구성 맵에 액세스할 수 있도록 합니다.

활성화된 ConfigMap PropertySource 는 애플리케이션 다음에 이름이 지정된 ConfigMap 에 대해 Kubernetes를 조회합니다(spring.application.name참조). 맵이 발견되면 해당 데이터를 읽고 다음을 수행합니다.

- 개별 속성 적용
- 속성 이름 application.yaml 적용
- 속성 이름 적용 application.properties

9.3.1. 개별 속성 적용

속성을 사용하여 스레드 풀 구성을 읽는 demo 라는 Spring Boot 애플리케이션이 있다고 가정하겠습니다.

- *pool.size.core*
- *pool.size.max*

YAML 형식의 구성 맵에 외부화할 수 있습니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  pool.size.core: 1
  pool.size.max: 16
```

9.3.2. application.yaml ConfigMap 속성 적용

개별 속성은 대부분의 경우에 적합하지만 YAML이 더 편리합니다. 이 경우 application.yaml 이라는 단일 속성을 사용하고 YAML을 여기에 삽입합니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  application.yaml: |-
    pool:
      size:
        core: 1
        max:16
```

9.3.3. application.properties ConfigMap 속성 적용

Spring Boot application.properties 파일의 스타일에 ConfigMap 속성을 정의할 수도 있습니다. 이 경우 application.properties 라는 단일 속성을 사용하고 그 안에 속성 설정을 나열합니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: demo
data:
  application.properties: |-
    pool.size.core: 1
    pool.size.max: 16
```

9.3.4. ConfigMap 배포

ConfigMap을 배포하고 Spring Boot 애플리케이션에 액세스하려면 다음 단계를 수행합니다.

절차

1. Spring Boot 애플리케이션에서 외부화된 구성 메커니즘을 사용하여 ConfigMap 속성 소스에 액세스합니다. 예를 들어 Java 8080에 @Configuration 주석에 주석을 달면 ConfigMap을 통해 빈의 속성 값을 삽입할 수 있습니다.

2. 프로젝트의 bootstrap.properties 파일(또는 bootstrap.yaml 파일)에서 ConfigMap의 이름과 일치하도록 spring.application.name 속성을 설정합니다.

3. 애플리케이션과 연결된 서비스 계정에 대한 보기 권한을 활성화합니다(기본적으로 기본이라는 서비스 계정임). 예를 들어 기본 서비스 계정에 보기 권한을 추가하려면 다음을 수행합니다.

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project -q)
```

9.4. SECRETS PROPERTYSOURCE 사용

Kubernetes에는 암호, OAuth 토큰 등과 같은 중요한 데이터를 저장하기 위한 시크릿 개념이 있습니다. Spring 클라우드 Kubernetes 플러그인은 시크릿과 통합을 제공하여 Spring Boot에서 시크릿에 액세스할 수 있도록 합니다.

활성화된 Secrets 속성 소스는 다음 소스에서 Kubernetes for Secrets 를 찾습니다. 시크릿이 발견되면 해당 데이터를 애플리케이션에서 사용할 수 있습니다.

1. 보안 마운트에서 재귀적으로 읽기

2. 애플리케이션 이름(spring.application.name참조)

3. 일부 라벨 일치

기본적으로 API(위 2 및 3)를 통해 Secrets를 사용하는 것은 활성화되지 않습니다.

9.4.1. 시크릿 설정 예

속성을 사용하여 **ActiveMQ** 및 **PostreSQL** 구성을 읽는 **demo** 라는 **Spring Boot** 애플리케이션이 있다고 가정하겠습니다.

```
amq.username
amq.password
pg.username
pg.password
```

이러한 보안은 **YAML** 형식의 시크릿 으로 외부화할 수 있습니다.

ActiveMQ Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: activemq-secrets
  labels:
    broker: activemq
type: Opaque
data:
  amq.username: bXl1c2VyCg==
  amq.password: MWYyZDFlMmU2N2Rm
```

PostreSQL 시크릿

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secrets
  labels:
    db: postgres
type: Opaque
data:
  pg.username: dXNlcgo=
  pg.password: cGdhZG1pbgo=
```

9.4.2. 보안 사용

다음과 같은 여러 가지 방법으로 사용할 보안을 선택할 수 있습니다.

- 보안이 매핑되는 디렉터리를 나열하여 다음을 수행하십시오.

```
-Dspring.cloud.kubernetes.secrets.paths=/etc/secrets/activemq,etc/secrets/postgres
```

공통 루트에 매핑된 모든 보안이 있는 경우 다음과 같이 설정할 수 있습니다.

```
-Dspring.cloud.kubernetes.secrets.paths=/etc/secrets
```

- 이름이 지정된 보안을 설정하면 다음을 수행합니다.

```
-Dspring.cloud.kubernetes.secrets.name=postgres-secrets
```

- 라벨 목록을 정의하여 다음을 수행합니다.

```
-Dspring.cloud.kubernetes.secrets.labels.broker=activemq
-Dspring.cloud.kubernetes.secrets.labels.db=postgres
```

9.4.3. Secrets PropertySource의 구성 속성

다음 속성을 사용하여 **Secrets** 속성 소스를 구성할 수 있습니다.

spring.cloud.kubernetes.secrets.enabled

Secrets 속성 소스를 활성화합니다. **type**은 부울 이며 기본값은 **true** 입니다.

spring.cloud.kubernetes.secrets.name

조회할 시크릿 이름을 설정합니다. **type**은 문자열 이며 기본값은 **${spring.application.name}** 입니다.

spring.cloud.kubernetes.secrets.labels

보안을 조회하는 데 사용되는 레이블을 설정합니다. 이 속성은 **Map** 기반 바인딩에 의해 정의된 대로 작동합니다. **type**은 **java.util.Map** 이며 기본값은 **null** 입니다.

spring.cloud.kubernetes.secrets.paths

보안이 마운트된 경로를 설정합니다. 이 속성은 컬렉션 기반 바인딩에 의해 정의된 대로 작동합니다. **type**은 **java.util.List** 이며 기본값은 **null** 입니다.

spring.cloud.kubernetes.secrets.enableApi

API를 통해 보안 사용을 활성화/비활성화합니다. **type**은 부울 이며 기본값은 **false** 입니다.

참고

보안상의 이유로 **API**를 통해 시크릿에 대한 액세스가 제한될 수 있습니다. 즉, 권장되는 방법은 **POD**에 시크릿을 마운트하는 것입니다.

9.5. PROPERTYSOURCE RELOAD 사용

일부 애플리케이션은 외부 속성 소스의 변경 사항을 감지하고 새 구성을 반영하도록 내부 상태를 업데이트해야 할 수 있습니다. **Spring Cloud Kubernetes**의 다시 로드 기능은 관련 **ConfigMap** 또는 **Secret**이 변경되면 애플리케이션을 다시 로드할 수 있습니다.

9.5.1. PropertySource Reload 활성화

Spring Cloud Kubernetes의 **PropertySource** 다시 로드 기능은 기본적으로 비활성화되어 있습니다.

절차

1. **faststart** 프로젝트의 **src/main/resources** 디렉터리로 이동하여 **bootstrap.yml** 파일을 엽니다.

2. 설정 속성 **spring.cloud.kubernetes.reload.enabled=true** 를 변경합니다.

9.5.2. PropertySource Reload의 수준

속성 **spring.cloud.kubernetes.reload.strategy:**에 대해 다음 수준의 재로드가 지원됩니다.

새로고침

(기본값) **@ConfigurationProperties** 또는 **@RefreshScope** 주석이 추가된 구성 빈만 다시 로드됩니다. 이 다시 로드 수준은 **Spring Cloud Context**의 새로 고침 기능을 활용합니다.

참고

PropertySource 다시 로드 기능은 다시 로드 전략이 새로 고침 되도록 설정된 경우 간단한 속성(즉, 컬렉션이 아님)에만 사용할 수 있습니다. 컬렉션에서 지원하는 속성은 런타임 시 변경할 수 없습니다.

restart_context

전체 Spring ApplicationContext 가 정상적으로 다시 시작됩니다. 빈은 새 구성으로 다시 생성됩니다.

shutdown

Spring ApplicationContext 가 종료되어 컨테이너를 다시 시작합니다. 이 수준을 사용하는 경우 모든 비daemon 스레드의 라이프사이클이 ApplicationContext에 바인딩되고 복제 컨트롤러 또는 복제본 세트가 Pod를 재시작하도록 구성되어 있는지 확인합니다.

9.5.3. PropertySource Reload의 예

다음 예제에서는 다시 로드 기능이 활성화된 경우 어떤 일이 발생하는지 설명합니다.

절차

1. 다시 로드 기능이 기본 설정(새로 고침 모드)으로 활성화되어 있다고 가정합니다. 구성 맵이 변경되면 다음 8080이 새로 고쳐집니다.

```
@Configuration
@ConfigurationProperties(prefix = "bean")
public class MyConfig {

    private String message = "a message that can be changed live";

    // getter and setters

}
```

2. 변경 사항을 확인하려면 다음과 같이 메시지를 주기적으로 출력하는 다른 8080을 만듭니다.

```
@Component
public class MyBean {

    @Autowired
    private MyConfig config;

    @Scheduled(fixedDelay = 5000)
    public void hello() {
        System.out.println("The message is: " + config.getMessage());
    }
}
```

3. 다음과 같이 ConfigMap을 사용하여 애플리케이션에서 인쇄한 메시지를 변경할 수 있습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: reload-example
data:
  application.properties: |-
    bean.message=Hello World!
```

Pod와 연결된 구성 맵에서 metrics.message 라는 속성의 변경 사항은 프로그램 출력에 반영됩니다.

9.5.4. PropertySource 다시 로드 운영 모드

다시 로드 기능은 다음 두 가지 작동 모드를 지원합니다.

event

(기본값) Kubernetes API(웹 소켓)를 사용하여 ConfigMap 또는 시크릿의 변경 사항을 감시합니다. 모든 이벤트는 구성을 다시 확인하고 변경 시 다시 로드됩니다. 구성 맵 변경 사항을 수신하려면 서비스 계정의 view 역할이 필요합니다. 상위 수준 역할(예:). 보안에는 편집해야 합니다(기본적으로 시크릿은 모니터링되지 않음).

폴링

구성 맵 및 시크릿에서 정기적으로 구성을 다시 생성하여 변경되었는지 확인합니다. 폴링 기간은 spring.cloud.kubernetes.reload.period 속성을 사용하여 구성할 수 있으며 기본값은 15초입니다. 모니터링된 속성 소스와 동일한 역할이 필요합니다. 예를 들어, 파일 마운트된 시크릿 소스에서 폴링을 사용하면 특정 권한이 필요하지 않습니다.

9.5.5. PropertySource 구성 속성 다시 로드

다음 속성을 사용하여 다시 로드 기능을 구성할 수 있습니다.

spring.cloud.kubernetes.reload.enabled

속성 소스 및 구성 다시 로드를 모니터링할 수 있습니다. type은 부울 이며 기본값은 false 입니다.

spring.cloud.kubernetes.reload.monitoring-config-maps

구성 맵의 변경 사항을 모니터링할 수 있습니다. type은 부울 이며 기본값은 true 입니다.

spring.cloud.kubernetes.reload.monitoring-secrets

*보안 변경 사항을 모니터링할 수 있습니다. **type**은 부울 이며 기본값은 **false** 입니다.*

spring.cloud.kubernetes.reload.strategy

*다시 로드를 실행할 때 사용할 전략입니다(새로 고침,**restart_context**,**shutdown**). **type**은 **Enum** 이고 기본값은 **refresh** 입니다.*

spring.cloud.kubernetes.reload.mode

*속성 소스의 변경 사항을 청취하는 방법(이벤트,폴링)을 지정합니다. **type**은 **Enum** 이고 기본값은 **event** 입니다.*

spring.cloud.kubernetes.reload.period

*폴링 전략을 사용할 때 변경 사항을 확인하는 데 사용되는 시간(밀리초)입니다. **type**은 **Long** 이고 기본값은 **15000** 입니다.*

다음 사항에 유의하십시오.

- *__spring.cloud.kubernetes.reload.*__ 속성은 **ConfigMaps** 또는 **Secrets**에서 사용해서는 안 됩니다. 런타임에 이러한 속성을 변경하면 예기치 않은 결과가 발생할 수 있습니다.*
- *속성 또는 전체 구성 맵을 삭제해도 새로 고침 수준을 사용할 때 빈의 원래 상태가 복원되지 않습니다.*

10장. KARAF 이미지에 대한 애플리케이션 개발

이 튜토리얼에서는 Karaf 이미지에 대한 애플리케이션을 만들고 배포하는 방법을 보여줍니다.

10.1. MAVEN ARCHETYPE을 사용하여 KARAF 프로젝트 생성

Maven archetype을 사용하여 Karaf 프로젝트를 생성하려면 다음 단계를 따르십시오.

절차

1. 시스템의 적절한 디렉터리로 이동합니다.

2. Maven 명령을 시작하여 Karaf 프로젝트를 생성합니다.

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \
  -
DarchetypeCatalog=https://maven.repository.redhat.com/ga/io/fabric8/archetypes/archetypes-catalog/2.2.0.fuse-760024-redhat-00001/archetypes-catalog-2.2.0.fuse-760024-redhat-00001-archetype-catalog.xml \
  -DarchetypeGroupId=org.jboss.fuse.fis.archetypes \
  -DarchetypeArtifactId=karaf-camel-log-archetype \
  -DarchetypeVersion=2.2.0.fuse-760024-redhat-00001
```

3. archetype 플러그인이 대화형 모드로 전환하여 나머지 필드를 입력하라는 메시지를 표시합니다.

```
Define value for property 'groupId': : org.example.fis
Define value for property 'artifactId': : fuse76-karaf-camel-log
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  org.example.fis: :
Confirm properties configuration:
groupId: org.example.fis
artifactId: fuse76-karaf-camel-log
version: 1.0-SNAPSHOT
package: org.example.fis
 Y: : Y
```

메시지가 표시되면 groupId 값으로 org.example.fis 를 입력하고 artifactId 값에 대해 fuse76-karaf-camel-log 를 입력합니다. 나머지 필드의 기본값을 수락합니다.

4. 위의 명령이 BUILD SUCCESS 상태로 종료되면 fuse76-karaf-camel-log 하위 디렉터리에

OpenShift 프로젝트에서 새 Fuse가 있어야 합니다.

5. *이제 fuse76-karaf-camel-log 프로젝트를 빌드하고 배포할 준비가 되었습니다. OpenShift에 로그인되어 있다고 가정하면 fuse76-karaf-camel-log 프로젝트의 디렉터리로 변경한 다음 다음과 같이 프로젝트를 빌드하고 배포합니다.*

```
cd fuse76-karaf-camel-log
mvn fabric8:deploy -Popenshift
```

참고

사용 가능한 Karaf archetypes의 전체 목록은 Karaf Archetype Catalog 를 참조하십시오.

10.2. CAMEL KARAF 애플리케이션 구조

Camel Karaf 애플리케이션의 디렉터리 구조는 다음과 같습니다.

```
├── pom.xml (1)
├── README.md
├── configuration
│   └── settings.xml
└── src
    ├── main
    │   ├── fabric8
    │   │   └── deployment.yml (2)
    │   ├── java
    │   │   └── org
    │   │       └── example
    │   │           └── fis
    │   └── resources
    │       ├── assembly
    │       │   └── etc
    │       │       └── org.ops4j.pax.logging.cfg (3)
    │       └── OSGI-INF
    │           └── blueprint
    │               └── camel-log.xml (4)
    └── test
        └── java
            └── org
                └── example
                    └── fis
```

Karaf 애플리케이션을 개발하는 데 다음 파일이 중요한 경우:

1

POM.xml: 추가 종속 항목을 포함합니다. pom.xml 파일에 종속 항목을 추가할 수 있습니다(예: 로깅의 경우 SLF4J를 사용할 수 있습니다.

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
</dependency>
```

2

src/main/fabric8/deployment.yml: fabric8-maven-plugin에서 생성한 기본 OpenShift 구성 파일과 병합되는 추가 구성을 제공합니다.

참고

이 파일은 Karaf 애플리케이션의 일부로 사용되지 않지만 CPU 및 메모리 사용과 같은 리소스를 제한하는 데 모든 퀵스타트에서 사용됩니다.

3

org.ops4j.pax.logging.cfg: 로그 수준을 사용자 지정하는 방법을 시연하고, 로깅 수준을 기본값 INFO 대신 DEBUG로 설정합니다.

4

Camel-log.xml: 애플리케이션의 소스 코드를 포함합니다.

10.3. KARAF ARCHETYPE 카탈로그

Karaf archetype 카탈로그에는 다음 예제가 포함되어 있습니다.

표 10.1. Karaf Maven Archetypes

이름	설명
karaf-camel-amq-archetype	Camel amq 구성 요소를 사용하여 Apache ActiveMQ 메시지 브로커에 메시지를 보내고 수신하는 방법을 보여줍니다.
karaf-camel-log-archetype	5초마다 서버 로그에 메시지를 기록하는 간단한 Apache Camel 애플리케이션을 보여줍니다.

이름	설명
karaf-camel-rest-sql-archetype	Camel의 REST DSL과 함께 JDBC를 통한 SQL을 사용하여 RESTful API를 노출하는 방법을 보여줍니다.
karaf-cxf-rest-archetype	CXF를 사용하여 RESTful(JAX-RS) 웹 서비스를 생성하고 OSGi HTTP Service를 통해 노출하는 방법을 보여줍니다.

10.4. FABRIC8 KARAF 기능 사용

Fabric8은 Apache Karaf를 지원하여 Kubernetes용 OSGi 앱을 더 쉽게 개발할 수 있도록 지원합니다.

Fabric8의 중요한 기능은 다음과 같습니다.

- *블루프린트 **XML** 파일에서 자리 표시자를 해결하기 위한 다양한 전략입니다.*
- *환경 변수*
- *시스템 속성*
- *서비스*
- ***Kubernetes ConfigMap***
- ***Kubernetes** 시크릿*
- ***Kubernetes** 구성 맵을 사용하여 **OSGi** 구성 관리를 동적으로 업데이트합니다.*
- ***OSGi** 서비스에 대한 **Kubernetes heath** 검사를 제공합니다.*

10.4.1. Fabric8 Karaf 기능 추가

이 기능을 사용하려면 프로젝트 POM 파일에 fabric8-karaf-features 종속성을 추가합니다.

절차

1. 프로젝트의 pom.xml 파일을 열고 fabric8-karaf-features 종속성을 추가합니다.

```
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-features</artifactId>
  <version>${fabric8.version}</version>
  <classifier>features</classifier>
  <type>xml</type>
</dependency>
```

fabric8 karaf 기능은 Karaf 서버에 설치됩니다.

10.4.2. Fabric8 Karaf Core 번들 기능 추가

bundle fabric8-karaf-core 는 블루프린트 및 ConfigAdmin 확장에서 사용하는 기능을 제공합니다.

절차

1. 프로젝트의 pom.xml 을 열고 fabric8-karaf-core 를 startupFeatures 섹션에 추가합니다.

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-core</feature>
  ...
</startupFeatures>
```

그러면 사용자 지정 Karaf 배포에 fabric8-karaf-core 기능이 추가됩니다.

10.4.3. Property Placeholder 서비스 옵션 설정

bundle fabric8-karaf-core 는 다음 인터페이스로 서비스 PlaceholderResolver 를 내보냅니다.

```
public interface PlaceholderResolver {
    /**
     * Resolve a placeholder using the strategy indicated by the prefix
     *
```

```
     * @param value the placeholder to resolve
     * @return the resolved value or null if not resolved
     */
    String resolve(String value);

    /**
     * Replaces all the occurrences of variables with their matching values from the resolver
using the given source string as a template.
     *
     * @param source the string to replace in
     * @return the result of the replace operation
     */
    String replace(String value);

    /**
     * Replaces all the occurrences of variables within the given source builder with their
matching values from the resolver.
     *
     * @param value the builder to replace in
     * @rerurn true if altered
     */
    boolean replaceIn(StringBuilder value);

    /**
     * Replaces all the occurrences of variables within the given dictionary
     *
     * @param dictionary the dictionary to replace in
     * @rerurn true if altered
     */
    boolean replaceAll(Dictionary<String, Object> dictionary);

    /**
     * Replaces all the occurrences of variables within the given dictionary
     *
     * @param dictionary the dictionary to replace in
     * @rerurn true if altered
     */
    boolean replaceAll(Map<String, Object> dictionary);
}
```

PlaceholderResolver 서비스는 다양한 속성 자리 표시자 확인 전략에 대해 수집기 역할을 합니다. 기본적으로 제공되는 해결 전략은 표 **Resolution Strategies** 에 나열되어 있습니다. 속성 자리 표시자 서비스 옵션을 설정하려면 시스템 속성 또는 환경 변수 또는 둘 다를 사용할 수 있습니다.

절차

1. **OpenShift의 ConfigMap에 액세스하려면 서비스 계정에 보기 권한이 필요합니다. 서비스 계정에 보기 권한을 추가합니다.**

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project
-q)
```

2. *API를 통해 보안에 대한 액세스가 제한될 수 있으므로 Pod에 보안을 마운트합니다.*

3. *Pod에서 볼륨 마운트로 사용 가능한 시크릿은 다음과 같이 시크릿이라는 디렉터리에 매핑됩니다.*

```
containers:
 -
  env:
  - name: FABRIC8_K8S_SECRETS_PATH
    value: /etc/secrets
    volumeMounts:
  - name: activemq-secret-volume
    mountPath: /etc/secrets/activemq
    readOnly: true
  - name: postgres-secret-volume
    mountPath: /etc/secrets/postgres
    readOnly: true

volumes:
  - name: activemq-secret-volume
  secret:
  secretName: activemq
  - name: postgres-secret-volume
   secret:
  secretName: postgres
```

10.4.4. 사용자 정의 속성 자리 표시자 확인기 추가

사용자 지정 암호화와 같은 특정 요구 사항을 지원하기 위해 사용자 지정 자리 표시자 확인자를 추가할 수 있습니다. 또한 PlaceholderResolver 서비스를 사용하여 블루프린트 및 ConfigAdmin에서 해결자를 사용할 수 있습니다.

절차

1. *pom.xml 프로젝트에 다음 mvn 종속성을 추가합니다.*

pom.xml

```
---
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-core</artifactId>
</dependency>
---
```

2. *PropertiesFunction 인터페이스를 구현하고 SCR을 사용하여 OSGi 서비스로 등록합니다.*

```
import io.fabric8.karaf.core.properties.function.PropertiesFunction;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.ConfigurationPolicy;
import org.apache.felix.scr.annotations.Service;

@Component(
    immediate = true,
    policy = ConfigurationPolicy.IGNORE,
    createPid = false
)
@Service(PropertiesFunction.class)
public class MyPropertiesFunction implements PropertiesFunction {
    @Override
    public String getName() {
        return "myResolver";
    }

    @Override
    public String apply(String remainder) {
        // Parse and resolve remainder
        return remainder;
    }
}
```

3. *다음과 같이 구성 관리에서 해결자를 참조할 수 있습니다.*

속성

```
my.property = $[myResolver:value-to-resolve]
```

10.4.5. 해결 전략 목록

PlaceholderResolver 서비스는 다양한 속성 자리 표시자 확인 전략에 대해 수집기 역할을 합니다. 기본적으로 제공되는 해결 전략은 표에 나열되어 있습니다.

1.
해결 전략 목록

접두사	예제	설명
env	**env:JAVA_HOME**	OS 환경 변수에서 속성을 조회합니다.
`sys	**sys:java.version**	Java JVM 시스템 속성에서 속성을 조회합니다.
`service	**service:amq**	서비스 이름 지정 규칙을 사용하여 OS 환경 변수에서 속성을 조회합니다.
service.host	**service.host:amq**	hostname 부분만 반환하는 서비스 이름 지정 규칙을 사용하여 OS 환경 변수에서 속성을 조회합니다.
service.port	**service.port:amq**	포트 부분만 반환하는 서비스 이름 지정 규칙을 사용하여 OS 환경 변수에서 속성을 조회합니다.
k8s:map	**k8s:map:myMap/myKey**	Kubernetes ConfigMap에서 속성 검색(API를 통해)
k8s:secret	**k8s:secret:amq/password**	Kubernetes Secrets에서 속성 검색(API 또는 볼륨 마운트를 통해)

10.4.6. Property Placeholder 서비스 옵션 목록

속성 자리 표시자 서비스는 다음 옵션을 지원합니다.

1.
속성 자리 표시자 서비스 옵션 목록

이름	기본	설명
fabric8.placeholder.prefix	$[	자리 표시자의 접두사
fabric8.placeholder.suffix	]	자리 표시자의 접미사
fabric8.k8s.secrets.path	null	보안이 매핑되는 쉼표로 구분된 경로 목록

이름	기본	설명
fabric8.k8s.secrets.api.enabled	false	API를 통해 시크릿 활성화/비활성화

10.5. FABRIC8 KARAF CONFIG 관리자 지원 추가

10.5.1. Fabric8 Karaf Config 관리자 지원 추가

*사용자 지정 **Karaf** 배포에 **Fabric8 Karaf Config** 관리자 지원을 추가할 수 있습니다.*

절차

- *프로젝트의 **pom.xml** 을 열고 **startupFeatures** 섹션에 **fabric8-karaf-cm** 를 추가합니다.*

pom.xml

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-cm</feature>
  ...
</startupFeatures>
```

10.5.2. ConfigMap 삽입 추가

***fabric8-karaf-cm** 는 **Karaf**의 **ConfigAdmin** 에 **ConfigMap** 값을 삽입하는 **ConfigAdmin** 브리지를 제공합니다.*

절차

1. ***ConfigAdmin** 브리지에서 추가하려면 **ConfigMap**에 **karaf.pid** 라는 레이블이 지정되어야 합니다. **karaf.pid** 값은 구성 요소의 **pid**에 해당합니다. 예를 들면 다음과 같습니다.*

```
kind: ConfigMap
apiVersion: v1
```

```
metadata:
  name: myconfig
  labels:
    karaf.pid: com.mycompany.bundle
data:
  example.property.1: my property one
  example.property.2: my property two
```

2. 구성을 정의하려면 단일 속성 이름을 사용할 수 있습니다. 개별 속성은 대부분의 경우 작동합니다. karaf/etc 의 pid 파일과 동일합니다. 예를 들면 다음과 같습니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: myconfig
  labels:
    karaf.pid: com.mycompany.bundle
data:
  com.mycompany.bundle.cfg: |
    example.property.1: my property one
    example.property.2: my property two
```

10.5.3. 구성 플러그인

fabric8-karaf-cm 는 구성 속성 자리 표시자를 확인하는 ConfigurationPlugin 을 제공합니다.

fabric8-karaf-cm 플러그인으로 속성 대체를 활성화하려면 Java 속성 fabric8.config.plugin.enabled 를 true 로 설정해야 합니다. 예를 들어 Karaf 이미지의 JAVA_OPTIONS 환경 변수를 사용하여 이 속성을 설정할 수 있습니다.

```
JAVA_OPTIONS=-Dfabric8.config.plugin.enabled=true
```

10.5.4. 구성 속성 위치 소유자

구성 속성 자리 표시자의 예는 다음과 같습니다.

my.service.cfg

```
amq.usr = $[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/username]
amq.pwd = $[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/password]
amq.url = tcp://$[env+service:ACTIVEMQ_SERVICE_NAME]
```

my-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
             https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
             http://camel.apache.org/schema/blueprint
             http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

  <cm:property-placeholder persistent-id="my.service" id="my.service" update-strategy="reload"/>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="userName"  value="${amq.usr}"/>
    <property name="password"  value="${amq.pwd}"/>
    <property name="brokerURL" value="${amq.url}"/>
  </bean>
</blueprint>
```

10.5.5. Fabric8 Karaf Config 관리자 옵션

Fabric8 Karaf Config Admin은 다음 옵션을 지원합니다.

이름	기본	설명
fabric8.config.plugin.enabled	false	ConfigurationPlugin 활성화
fabric8.cm.bridge.enabled	true	ConfigAdmin 브리지 활성화
fabric8.config.watch	true	ConfigMap 변경 감시 활성화
fabric8.config.merge	false	ConfigAdmin에서 병합 ConfigMap 값 활성화
fabric8.config.meta	true	ConfigAdmin 브리지에서 ConfigMap 메타 삽입 활성화

이름	기본	설명
fabric8.pid.label	karaf.pid	ConfigAdmin 브리지가 찾는 레이블을 정의합니다(즉, 선택해야 하는 ConfigMap에는 해당 레이블이 있어야 합니다. 해당 레이블의 값은 연결된 PID를 결정합니다.)
fabric8.pid.filters	빈	ConfigMap을 선택하려면 ConfigAdmin 브릿지에 대한 추가 조건을 정의합니다. 지원되는 구문은 다음과 같습니다. ● 서로 다른 라벨의 조건은 ""로 구분되며 서로 AND로 구분됩니다. ● 레이블 내에서 together together (;)는 OR로 간주되며 레이블 값의 조건에 대한 구분 기호로 사용할 수 있습니다. 예를 들어 - **Dfabric8.pid.filters=appNames=appNames=A;B,database.name=my.oracle.datasource** 와 같은 필터가 "give me all ConfigMaps that have a label appName with values A 또는 B and a label database.name equals to my.oracle.datasource"로 변환됩니다.

중요

ConfigurationPlugin 에는 Aries 블루프린트 CM 1.0.9 이상이 필요합니다.

10.6. FABRIC8 KARAF 블루프린트 지원 추가

fabric8-karaf-blueprint 는 Aries PropertyEvaluator 및 fabric8-karaf-core 의 속성 자리 표시자를 사용하여 블루프린트 XML 파일의 자리 표시자를 해결합니다.

절차

- ***사용자 정의 Karaf 배포에 블루프린트 지원 기능을 포함하려면 프로젝트 pom.xml 의 startupFeatures 섹션에 fabric8-karaf-blueprint 를 추가합니다.***

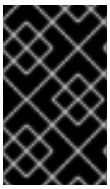
-

```
<startupFeatures>
  ...
  <feature>fabric8-karaf-blueprint</feature>
  ...
</startupFeatures>
```

예제

fabric8 평가기는 **${env+service:MY_ENV_VAR}** 과 같은 체인 평가기를 지원합니다. 환경 변수에 대해 **MY_ENV_VAR** 변수를 확인해야 합니다. 그런 다음 **service function**을 사용하여 결과를 해결합니다. 예를 들면 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.2.0"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
             https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
             http://camel.apache.org/schema/blueprint
             http://camel.apache.org/schema/blueprint/camel-blueprint.xsd
             http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.3.0
             http://aries.apache.org/schemas/blueprint-ext/blueprint-ext-1.3.xsd">

  <ext:property-placeholder evaluator="fabric8" placeholder-prefix="$[" placeholder-suffix="]"/>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="userName"
value="$[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/username]"/>
    <property name="password"
value="$[k8s:secret:$[env:ACTIVEMQ_SERVICE_NAME]/password]"/>
    <property name="brokerURL" value="tcp://$[env+service:ACTIVEMQ_SERVICE_NAME]"/>
  </bean>
</blueprint>
```



중요

중첩된 속성 자리 표시자 대체에는 **Aries Blueprint Core 1.7.0** 이상이 필요합니다.

10.7. FABRIC8 KARAF 상태 점검 활성화

fabric8-karaf-check 를 시작 기능으로 설치하는 것이 좋습니다. 활성화되면 **Karaf** 서버에서 준비 및 활성 프로브에 **Kubernetes**에서 사용할 수 있는 **http://0.0.0.0:8181/readiness-check** 및 **http://0.0.0.0:8181/health-check** **URL**을 노출할 수 있습니다.

참고

이러한 **URL**은 다음 사항이 사실인 경우에만 **HTTP 200** 상태 코드를 사용하여 응답합니다.

- **OSGi Framework**가 시작되었습니다.
- 모든 **OSGi** 번들이 시작됩니다.
- 모든 부팅 기능이 설치됩니다.
- 배포된 모든 **Blue Cryostat** 번들은 생성된 상태에 있습니다.
- 배포된 모든 **SCR** 번들은 활성, 등록 또는 팩토리 상태에 있습니다.
- 모든 웹 번들은 웹 서버에 배포됩니다.
- 생성된 모든 **Camel** 컨텍스트는 **started** 상태에 있습니다.

절차

1. 프로젝트의 **pom.xml** 을 열고 **startupFeatures** 섹션에 **fabric8-karaf-checks** 기능을 추가합니다.

pom.xml

```
<startupFeatures>
 ...
 <feature>fabric8-karaf-checks</feature>
 ...
</startupFeatures>
```

fabric8-maven-plugin:resources 목표는 *fabric8-karaf-checks* 기능을 사용하는지 여부를 감지하고 컨테이너 구성에 준비 및 활성 프로브를 자동으로 추가합니다.

10.8. 사용자 정의 상태 점검 추가

추가 사용자 지정 *heath* 검사를 제공하여 *Karaf* 서버가 요청을 처리할 준비가 되기 전에 사용자 트래픽을 수신하지 못하도록 할 수 있습니다. 사용자 정의 상태 점검을 활성화하려면 *io.fabric8.karaf.checks.HealthChecker* 또는 *io.fabric8.karaf.checks.ReadinessChecker* 인터페이스를 구현하고 해당 오브젝트를 *OSGi* 레지스트리에 등록해야 합니다.

절차

- 다음 *mvn* 종속성을 프로젝트 *pom.xml* 파일에 추가합니다.

pom.xml

```
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-karaf-checks</artifactId>
</dependency>
```

참고

OSGi 레지스트리에서 오브젝트를 생성하고 등록하는 가장 간단한 방법은 *SCR*을 사용하는 것입니다.

예제

다음과 같이 상태 점검을 수행하여 사용 가능한 디스크 공간이 있는지 확인하는 예는 다음과 같습니다.

```
import io.fabric8.karaf.checks.*;
import org.apache.felix.scr.annotations.*;
import org.apache.commons.io.FileSystemUtils;
import java.util.Collections;
import java.util.List;

@Component(
```

```
    name = "example.DiskChecker",
    immediate = true,
    enabled = true,
    policy = ConfigurationPolicy.IGNORE,
    createPid = false
)
@Service({HealthChecker.class, ReadinessChecker.class})
public class DiskChecker implements HealthChecker, ReadinessChecker {

    public List<Check> getFailingReadinessChecks() {
        // lets just use the same checks for both readiness and health
        return getFailingHeathChecks();
    }

    public List<Check> getFailingHealthChecks() {
        long free = FileSystemUtils.freeSpaceKb("/");
        if (free < 1024 * 500) {
            return Collections.singletonList(new Check("disk-space-low", "Only " + free + "kb of
disk space left."));
        }
        return Collections.emptyList();
    }
}
```

11장. JBOSS EAP 이미지용 애플리케이션 개발

JBoss EAP에서 Fuse 애플리케이션을 개발하기 위해 대체 방법은 S2I 소스 워크플로우를 사용하여 EAP와 함께 Red Hat Camel CDI용 OpenShift 프로젝트를 생성하는 것입니다.

사전 요구 사항

- OpenShift가 올바르게 실행 중이고 Fuse 이미지 스트림이 이미 OpenShift에 설치되어 있는지 확인합니다. 관리자용 시작하기를 참조하십시오.
- Maven 리포지토리가 fuse용으로 구성되었는지 확인합니다. Maven 리포지토리 구성을 참조하십시오.

11.1. S2I 소스 워크플로우를 사용하여 JBOSS EAP 프로젝트 생성

JBoss EAP에서 Fuse 애플리케이션을 개발하기 위해 대체 방법은 S2I 소스 워크플로우를 사용하여 EAP와 함께 Red Hat Camel CDI용 OpenShift 프로젝트를 생성하는 것입니다.

절차

1. 기본 서비스 계정에 view 역할을 추가하여 클러스터링을 활성화합니다. 그러면 사용자에게 기본 서비스 계정에 대한 보기 액세스 권한이 부여됩니다. 빌드, 배포 및 기타 Pod를 실행하려면 각 프로젝트에 서비스 계정이 필요합니다. 쉘 프롬프트에서 다음 oc 클라이언트 명령을 입력합니다.

```
oc login -u developer -p developer
oc policy add-role-to-user view -z default
```

2. OpenShift 템플릿에 설치된 Fuse를 확인합니다.

```
oc get template -n openshift
```

3. 다음 명령을 입력하여 EAP 빠른 시작으로 Red Hat Fuse 7.6 Camel CDI를 실행하는 데 필요한 리소스를 생성합니다. 빠른 시작에 대한 배포 구성 및 빌드 구성을 생성합니다. 빠른 시작 및 생성된 리소스에 대한 정보가 터미널에 표시됩니다.

```
oc new-app s2i-fuse76-eap-camel-cdi

--> Creating resources ...
```

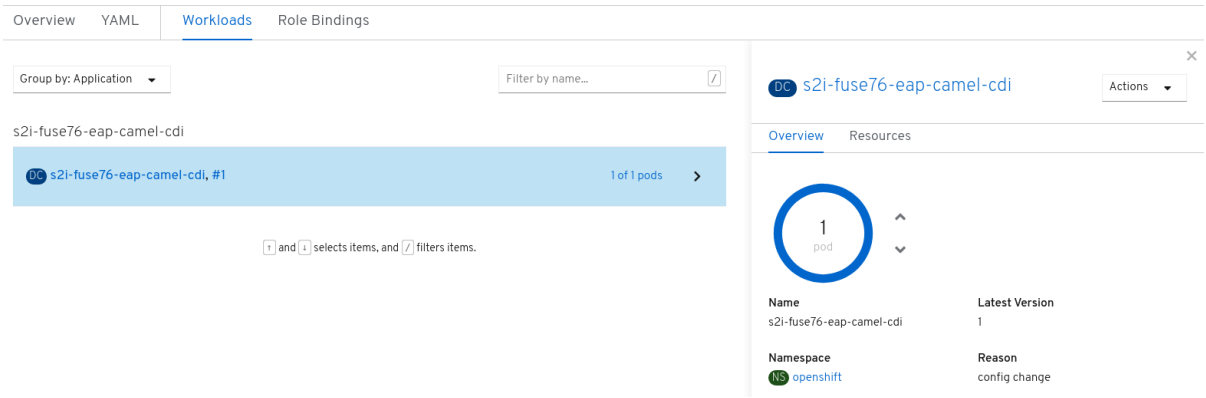
```
    service "s2i-fuse76-eap-camel-cdi" created
    service "s2i-fuse76-eap-camel-cdi-ping" created
    route.route.openshift.io "s2i-fuse76-eap-camel-cdi" created
    imagestream.image.openshift.io "s2i-fuse76-eap-camel-cdi" created
    buildconfig.build.openshift.io "s2i-fuse76-eap-camel-cdi" created
    deploymentconfig.apps.openshift.io "s2i-fuse76-eap-camel-cdi" created
--> Success
    Access your application via route 's2i-fuse76-eap-camel-cdi-OPENSHIFT_IP_ADDR'
    Build scheduled, use 'oc logs -f bc/s2i-fuse76-eap-camel-cdi' to track its progress.
    Run 'oc status' to view your app.
```

4. *브라우저에서 **OpenShift** 웹 콘솔로 이동하고(**https://OPENSHIFT_IP_ADDR**, **OPENSHIFT_IP_ADDR** 을 클러스터의 **IP** 주소로 교체) 인증 정보로 로그인합니다(예: 사용자 이름 **developer** 및 암호, 개발자).*

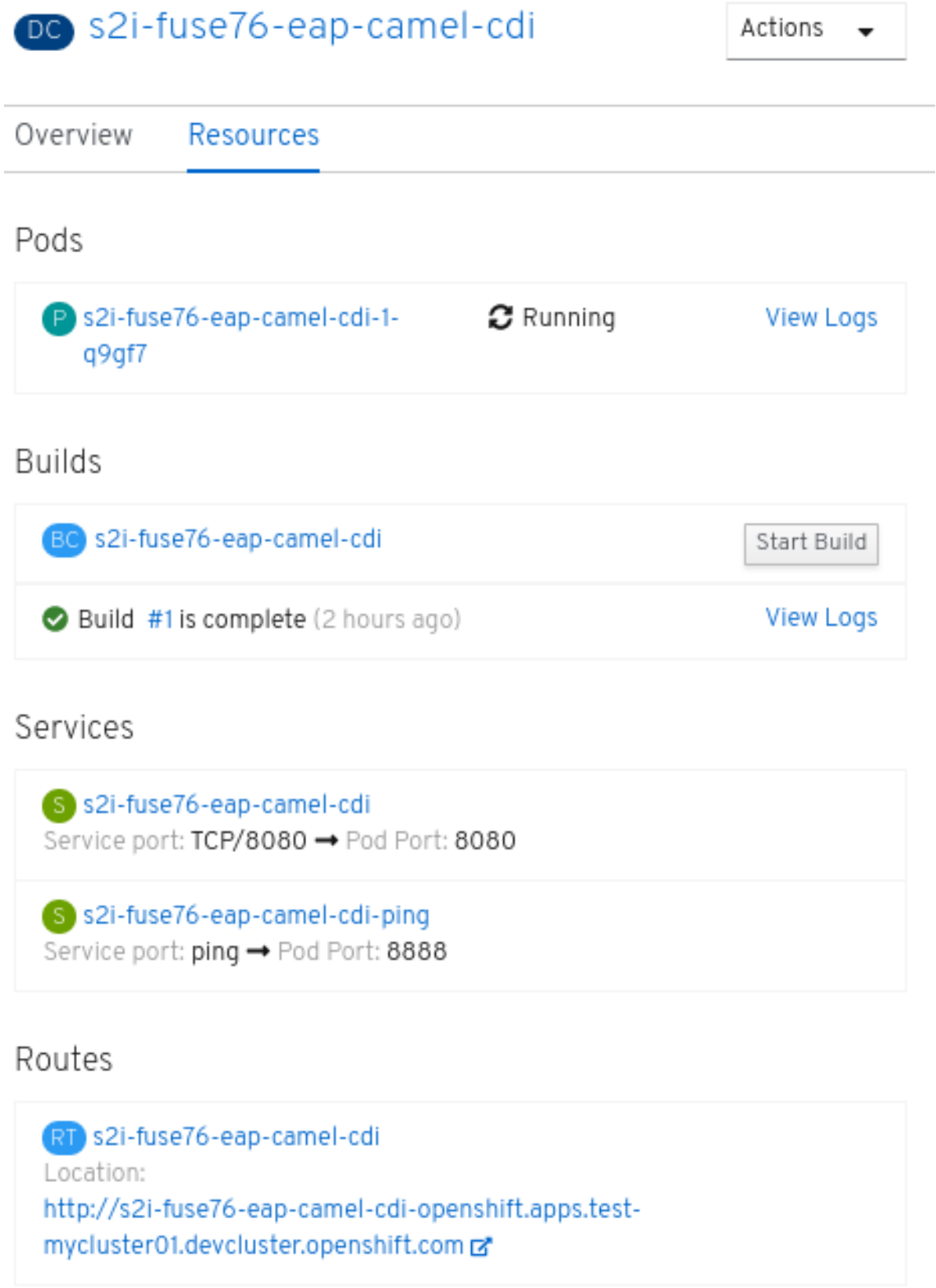
5. *왼쪽 패널에서 홈을 확장합니다. **Projects** 를 클릭한 다음 **openshift** 프로젝트를 선택하여 프로젝트 대시보드를 확인합니다.*

6. *워크로드 탭을 클릭합니다. 선택한 네임스페이스(예: **openshift**)의 기존 애플리케이션이 모두 표시됩니다.*

7. ***s2i-fuse76-eap-camel-cdi** 를 클릭하여 퀵 스타트에 대한 개요 정보 페이지를 확인합니다.*



8. *리소스 탭을 클릭한 다음 경로 섹션에 표시된 링크를 클릭하여 애플리케이션에 액세스합니다.*



링크의 형식은 http://s2i-fuse76-eap-camel-cdi-OPENSHIFT_IP_ADDR 입니다. 브라우저에 다음과 같은 메시지가 표시됩니다.

> *Hello world from 172.17.0.3*

9. **URL에서 name 매개변수를 사용하여 이름을 지정할 수도 있습니다. 예를 들어 브라우저에 URL http://s2i-fuse76-eap-camel-cdi-openshift.apps.cluster-name.openshift.com/?name=jdoe 을 입력하면 응답이 표시됩니다.**

> Hello jdoe from 172.17.0.3

10. **View Logs (로그 보기)를 클릭하여 애플리케이션의 로그를 확인합니다.**

11. **실행 중인 Pod를 종료하려면 다음을 수행합니다.**

    a. **개요 탭을 클릭하여 애플리케이션의 개요 정보 페이지로 돌아갑니다.**

    b. **Pod 아이콘 옆에 있는 아래쪽 화살표**

    를 클릭하여 Pod를 중지하도록 0으로 축소합니다.

11.2. JBOSS EAP 애플리케이션의 구조

다음 위치에서 EAP 예제를 사용하여 Red Hat Fuse 7.6 Camel CDI의 소스 코드를 찾을 수 있습니다.

> https://github.com/wildfly-extras/wildfly-camel-examples/tree/wildfly-camel-examples-5.2.0.fuse-720021/camel-cdi

EAP 애플리케이션에서 Camel의 디렉터리 구조는 다음과 같습니다.

```
├── pom.xml
├── README.md
├── configuration
│   └── settings.xml
└── src
    └── main
        ├── java
        │   └── org
        │       └── wildfly
        │           └── camel
        │               └── examples
        │                   └── cdi
        │                       └── camel
        │                           ├── MyRouteBuilder.java
```

```
        ├── SimpleServlet.java
        └── SomeBean.java
    └── webapp
        └── WEB-INF
            └── beans.xml
```

*다음 파일이 **JBoss EAP** 애플리케이션을 개발하는 데 중요한 위치:*

pom.xml

추가 종속 항목을 포함합니다.

11.3. JBOSS EAP 빠른 시작 템플릿

*다음 **S2I** 템플릿은 **JBoss EAP**의 **Fuse**에 제공됩니다.*

표 11.1. JBoss EAP S2I 템플릿

이름	설명
JBoss Fuse 7.6 Camel A-MQ 및 EAP(Cryo**stat-camel-amq-template**)	카l-activemq 구성 요소를 사용하여 OpenShift에서 실행되는 AMQ 메시지 브로커에 연결하는 방법을 보여줍니다. 브로커가 이미 배포되어 있다고 가정합니다.
Red Hat Fuse 7.6 Camel CDI with EAP (Cryo**stat-camel-cdi-template**)	camel-cdi 구성 요소를 사용하여 CDI 빈을 Camel 경로와 통합하는 방법을 보여줍니다.
Red Hat Fuse 7.6 CXF Cryostat-RS with EAP (#159**-camel-cxf-jaxrs-template**)	camel-cxf 구성 요소를 사용하여 Cryostat-RS REST 서비스를 생성하고 사용하는 방법을 보여줍니다.
Red Hat Fuse 7.6 CXF Cryostat-WS with EAP (#159**-camel-cxf-jaxws-template**)	camel-cxf 구성 요소를 사용하여 Cryostat-WS 웹 서비스를 생성하고 사용하는 방법을 보여줍니다.
Red Hat Fuse 7.6 Camel JPA + MySQL(Ephemeral)(EAP **-camel-jpa-template**포함)	Red Hat Fuse on EAP를 사용하여 Camel 애플리케이션을 MySQL 데이터베이스에 연결하고 REST API를 노출하는 방법을 보여줍니다. 이 예제에서는 두 개의 컨테이너, 즉 하나의 컨테이너를 MySQL 서버로 실행하는 컨테이너를 생성하고 다른 컨테이너는 데이터베이스의 클라이언트 역할을 하는 Camel 애플리케이션을 실행합니다.

12장. OPENSHIFT에서 FUSE에서 영구 스토리지 사용

OpenShift 애플리케이션의 Fuse는 영구 파일 시스템이 없는 OpenShift 컨테이너를 기반으로 합니다. 애플리케이션을 시작할 때마다 변경 불가능한 Docker 형식의 이미지가 있는 새 컨테이너에서 시작됩니다. 따라서 컨테이너가 중지되면 파일 시스템의 저장된 데이터가 손실됩니다. 그러나 애플리케이션은 일부 상태를 영구 저장소에 데이터로 저장해야 하며, 애플리케이션이 공통 데이터 저장소에 대한 액세스를 공유하는 경우도 있습니다. OpenShift 플랫폼은 외부 저장소 프로비저닝을 영구 스토리지로 지원합니다.

12.1. 볼륨 및 볼륨 유형 정보

OpenShift를 사용하면 포드 및 컨테이너가 여러 로컬 또는 네트워크 연결 스토리지 끝점에서 지원하는 파일 시스템으로 볼륨 을 마운트할 수 있습니다.

볼륨 유형은 다음과 같습니다.

- emptyDir(빈 디렉터리): 기본 볼륨 유형입니다. 로컬 호스트에서 Pod가 생성될 때 할당되는 디렉터리입니다. 서버에서 복사되지 않으며 Pod를 삭제하면 디렉터리가 제거됩니다.
- ConfigMap: 이름이 지정된 configmap의 키-값 쌍으로 채워진 콘텐츠가 있는 디렉터리입니다.
- hostPath(호스트 디렉터리): 모든 호스트에 특정 경로가 있는 디렉터리이며 승격된 권한이 필요합니다.
- 시크릿(마운트된 시크릿): 시크릿 볼륨은 이름이 지정된 시크릿을 제공된 디렉터리에 마운트합니다.
- PersistentVolumeClaim 또는 pvc(영구 볼륨 클레임): 컨테이너의 볼륨 디렉터리를 이름으로 할당한 영구 볼륨 클레임에 연결합니다. 영구 볼륨 클레임은 스토리지 할당 요청입니다. 클레임이 바인딩되지 않으면 Pod가 시작되지 않습니다.

볼륨은 Pod 수준에서 구성되며 hostPath 를 사용하여 외부 스토리지에 직접 액세스할 수 있습니다. 따라서 여러 Pod의 공유 리소스에 대한 액세스를 hostPath 볼륨으로 관리하기가 더 어렵습니다.

12.2. PERSISTENTVOLUMES 정보

persistentVolumes 를 사용하면 클러스터 관리자가 NFS, Ceph RBD, AWS EBS(Elastic Block Store) 등과 같은 다양한 유형의 네트워크 스토리지에서 지원하는 클러스터 전체 스토리지를 프로비저닝할 수 있습니다. persistentVolumes는 용량, 액세스 모드 및 재활용 정책도 지정합니다. 이를 통해 여러 프로젝트의 Pod가 기본 리소스의 특성을 고려하지 않고 영구 스토리지에 액세스할 수 있습니다.

다양한 유형의 PersistentVolume을 생성하려면 영구 스토리지 구성 을 참조하십시오.

12.3.

절차

1.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 2Mi
  hostPath:
    path: /data/pv0001/
```

2.

```
oc create -f pv.yaml
```

3.

```
oc get pv
```

12.4.

절차

-

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc0001
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Mi
```

12.5.

절차

1.

```
spec:
  template:
    spec:
      containers:
        - volumeMounts:
          - name: vol0001
            mountPath: /usr/share/data
      volumes:
        - name: vol0001
          persistentVolumeClaim:
            claimName: pvc0001
```

2.

```
mvn fabric8:resource-apply
```

3.

```
oc describe deploymentconfig <application-dc-name>
```

13장.

13.1.

13.2.

- fuse7-java-openshift
- fuse7-karaf-openshift
- fuse7-eap-openshift
- fuse7-console

- ***apicurito-ui***
- ***fuse-apicurito-generator***

절차

1.

```
$ oc get is -n openshift
NAME                        DOCKER REPO                                    TAGS
UPDATED
fuse7-console               172.30.1.1:5000/openshift/fuse7-console
1.0,1.1,1.2,1.3,1.4,1.5,1.6     About an hour ago
fuse7-eap-openshift         172.30.1.1:5000/openshift/fuse7-eap-openshift
1.0,1.1,1.2,,1.3,1.4,1.5,1.6    About an hour ago
fuse7-java-openshift        172.30.1.1:5000/openshift/fuse7-java-openshift
1.0,1.1,1.2,1.3,1.4,1.5,1.6     About an hour ago
fuse7-karaf-openshift       172.30.1.1:5000/openshift/fuse7-karaf-openshift
1.0,1.1,1.2,1.3,1.4,1.5,1.6     About an hour ago...
fuse-apicurito-generator      172.30.1.1:5000/openshift/fuse-apicurito-generator
1.2,1.3,1.4,1.5,1.6          About an hour ago...
apicurito-ui                172.30.1.1:5000/openshift/apicurito-ui          1.2,1.3,1.4,1.5,1.6
About an hour ago...
```

2.

```
oc import-image -n openshift fuse7-java-openshift:1.6
oc import-image -n openshift fuse7-karaf-openshift:1.6
oc import-image -n openshift fuse7-eap-openshift:1.6
oc import-image -n openshift fuse7-console:1.6
oc import-image -n openshift apicurito-ui:1.6
oc import-image -n openshift fuse-apicurito-generator:1.6
```

참고

참고

13.3.

절차

1.

```
oc login URL -u ADMIN_USER -p ADMIN_PASS
```

2.

```
BASEURL=https://raw.githubusercontent.com/jboss-fuse/application-templates/application-templates-2.1.0.fuse-760043-redhat-00003
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cdi-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cxf-jaxrs-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-cxf-jaxws-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/eap-camel-jpa-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-log-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-camel-rest-sql-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/karaf-cxf-rest-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-amq-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-config-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-drools-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-infinispan-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-camel-xml-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-cxf-jaxrs-template.json
oc replace --force -n openshift -f ${BASEURL}/quickstarts/spring-boot-cxf-jaxws-template.json
```

참고

13.4.

13.4.1.

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

    <fuse.version>7.6.0.fuse-760027-redhat-00001</fuse.version>
    ...
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
  <build>
    ...
    <plugins>
      <!-- Core plugins -->
      ...
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        ...
        <version>${fuse.version}</version>
      </plugin>
    </plugins>
  </build>
```

```
  <profiles>
    <profile>
      <id>openshift</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.jboss.redhat-fuse</groupId>
            <artifactId>fabric8-maven-plugin</artifactId>
            ...
            <version>${fuse.version}</version>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
</project>
```

fuse.version

13.4.2.

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

    <fuse.version>7.6.0.fuse-760027-redhat-00001</fuse.version>
    ...
  </properties>

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

```
  ...
  <plugins>
    ...
    <plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>karaf-maven-plugin</artifactId>
      <version>${fuse.version}</version>
      ...
    </plugin>
    ...
    <plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>fabric8-maven-plugin</artifactId>
      <version>${fuse.version}</version>
      ...
    </plugin>
  </plugins>
 </build>

</project>
```

fuse.version

13.4.3.

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

  <fuse.version>7.6.0.fuse-760027-redhat-00001</fuse.version>
  ...
 </properties>

 <!-- Dependency Management -->
 <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>
```

fuse.version

13.5.

표 **13.1.**

Red Hat Fuse 7.0.0 GA	7.6.0.fuse-760027-redhat-00001
	7.0.1.fuse-000008-redhat-4

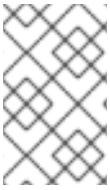
부록 A.

A.1.

-
-
-
-

A.2.

-
-



참고

A.2.1.

예제

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>

    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

      <!-- configure the Fuse version you want to use here -->
      <fuse.bom.version>7.6.0.fuse-sb2-760028-redhat-00001</fuse.bom.version>

      <!-- maven plugin versions -->
      <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
      <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
    </properties>


    <build>
      <defaultGoal>spring-boot:run</defaultGoal>

      <plugins>
        <plugin>
          <groupId>org.jboss.redhat-fuse</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
          <version>${fuse.bom.version}</version>
          <executions>
            <execution>
              <goals>
                <goal>repackage</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>

    <repositories>
      <repository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
      <repository>
```

```
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
    </repositories>

    <pluginRepositories>
      <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
</project>
```

A.2.2.

예제

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>
```

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

  <!-- configure the Fuse version you want to use here -->
  <fuse.bom.version>7.6.0.fuse-760027-redhat-00001</fuse.bom.version>

  <!-- maven plugin versions -->
  <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
  <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
</properties>


<build>
  <defaultGoal>spring-boot:run</defaultGoal>

  <plugins>
    <plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>${fuse.bom.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

<repositories>
  <repository>
    <id>redhat-ga-repository</id>
    <url>https://maven.repository.redhat.com/ga</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>redhat-ea-repository</id>
    <url>https://maven.repository.redhat.com/earlyaccess/all</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

```
    <pluginRepositories>
      <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
</project>
```

부록 B.

B.1.

-
-
-
-
-

B.2.

-
-

-

-

예제

```
<libraries>
  <library>mvn:org.postgresql/postgresql/9.3-1102-jdbc41;type:=endorsed</library>
</libraries>
```

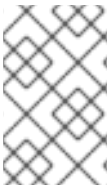
B.3.

-

-

-

B.3.1.



참고

예제

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>karaf-maven-plugin</artifactId>
  <version>${fuse.version}</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <id>karaf-assembly</id>
      <goals>
        <goal>assembly</goal>
      </goals>
      <phase>install</phase>
    </execution>
  </executions>
  <configuration>

    <karafVersion>{karafMavenPluginVersion}</karafVersion>
    <useReferenceUrls>true</useReferenceUrls>
    <archiveTarGz>false</archiveTarGz>
    <includeBuildOutputDirectory>false</includeBuildOutputDirectory>
    <startupFeatures>
      <feature>karaf-framework</feature>
          </startupFeatures>
          <bootFeatures>
      <feature>shell</feature>
      <feature>jaas</feature>
      <feature>aries-blueprint</feature>
      <feature>camel-blueprint</feature>
      <feature>fabric8-karaf-blueprint</feature>
      <feature>fabric8-karaf-checks</feature>
    </bootFeatures>
    <bootBundles>
      <bundle>mvn:${project.groupId}/${project.artifactId}/${project.version}</bundle>
    </bootBundles>
  </configuration>
</plugin>
```

부록 C.

-
-
-
-

C.1.

중요

C.2.

설정

-

-

-

C.3.

절차

-

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>fabric8-maven-plugin</artifactId>
  <version>${fuse.version}</version>

  <configuration>
    ....
    <images>
      <!-- A single's image configuration -->
      <image>
        ...
        <build>
          ....
        </build>
      </image>
      ....
    </images>
  </configuration>

  <!-- Connect fabric8:resource and fabric8:build to lifecycle phases -->
  <executions>
    <execution>
      <id>fabric8</id>
      <goals>
        <goal>resource</goal>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

C.4.

-
-

C.4.1.

표 C.1.

	설명
fabric8:build	
fabric8:resource	
fabric8:apply	
fabric8:resource-apply	

표 C.2.

	설명
fabric8:run	
fabric8:deploy	
fabric8:undeploy	
fabric8:start	
fabric8:stop	
fabric8:log	
fabric8:debug	
fabric8:watch	

C.4.2.

예를 들면 다음과 같습니다.

예제

```
<configuration>
  <resources>
    <env>
      <JAVA_OPTIONS>-Dmy.custom=option</JAVA_OPTIONS>
      <MY_VAR>value</MY_VAR>
    </env>
  </resources>
</configuration>
```

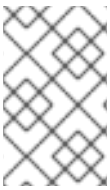
C.4.3.

표 C.3.

설정	설명	Default
fabric8.skipResourceValidation		false
fabric8.failOnValidationError		false
fabric8.build.switchToDeployment		false
fabric8.openshift.trimImageInContainerSpec		false

C.5.

- *C.5.3절. “”*
- *C.5.4절. “Karaf”*



참고

C.5.1.

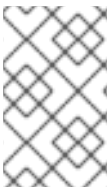
C.5.2.

istag

```
<namespace>/<image-stream-name>:<tag>
```

docker

```
[<registry-location-url>/]<image-namespace>/<image-name>:<tag>
```



참고

C.5.2.1.

```
fuse7/fuse-eap-openshift:1.6
fuse7/fuse-java-openshift:1.6
fuse7/fuse-karaf-openshift:1.6
```

참고

C.5.2.2.

```
fuse7/fuse-eap-openshift:1.6
fuse7/fuse-java-openshift:1.6
fuse7/fuse-karaf-openshift:1.6
```

C.5.2.3.

예제

```
<configuration>
  <generator>
    <config>
      <spring-boot>
        <fromMode>{istag|docker}</fromMode>
```

```
        <from>{image locations}</from>
      </spring-boot>
    </config>
  </generator>
</configuration>
```

C.5.2.4.

예제

```
<configuration>
  <generator>
    <config>
      <karaf>
        <fromMode>{istag|docker}</fromMode>
        <from>{image locations}</from>
      </karaf>
    </config>
  </generator>
</configuration>
```

C.5.2.5.

예제

```
//build from Docker-formatted image directly, registry location, image name or tag are subject to
change if desirable
-Dfabric8.generator.fromMode=docker
-Dfabric8.generator.from=<custom-registry-location-url>/<image-namespace>/<image-name>:<tag>

//to use ImageStream from different namespace
-Dfabric8.generator.fromMode=istag //istag is default
-Dfabric8.generator.from=<namespace>/<image-stream-name>:<tag>
```

C.5.3.

표 C.4.

	설명	Default
assemblyRef		
targetDir		**/deployments**
jolokiaPort		8778
mainClass		
webPort		8080

C.5.4. Karaf

표 C.5.

	설명	Default
baseDir		**/deployments**

	설명	Default
jolokiaPort		8778
mainClass		
user		**jboss:jboss:jboss**
webPort		8080

부록 D.

D.1.

-

D.2.

절차

1.

```
<plugin>
  <groupId>io.fabric8.forge</groupId>
  <artifactId>fabric8-camel-maven-plugin</artifactId>
  <version>2.3.90</version>
</plugin>
```

2.

```
mvn fabric8-camel:validate
```

예제

```
<plugin>
  <groupId>io.fabric8.forge</groupId>
  <artifactId>fabric8-camel-maven-plugin</artifactId>
  <version>2.3.80</version>
  <executions>
    <execution>
      <phase>process-classes</phase>
    <goals>
      <goal>validate</goal>
    </goals>
    </execution>
  </executions>
</plugin>
```

예제

```
<plugin>
  <groupId>io.fabric8.forge</groupId>
  <artifactId>fabric8-camel-maven-plugin</artifactId>
  <version>2.3.80</version>
  <executions>
    <execution>
      <configuration>
        <includeTest>true</includeTest>
      </configuration>
      <phase>process-test-classes</phase>
      <goals>
        <goal>validate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

D.3.

절차

-

```
$cd camel-example-cdi
$mvn io.fabric8.forge:fabric8-camel-maven-plugin:2.3.80:validate
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Camel :: Example :: CDI 2.16.2
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- fabric8-camel-maven-plugin:2.3.80:validate (default-cli) @ camel-example-cdi ---
[INFO] Endpoint validation success: (4 = passed, 0 = invalid, 0 = incapable, 0 = unknown
components)
[INFO] Simple validation success: (0 = passed, 0 = invalid)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

예제

1.

```
@Uri("timer:foo?period=5000")
```

2.

```
@Uri("timer:foo?perid=5000")
```

3.

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Camel :: Example :: CDI 2.16.2
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- fabric8-camel-maven-plugin:2.3.80:validate (default-cli) @ camel-example-cdi ---
[WARNING] Endpoint validation error at:
```

```
org.apache.camel.example.cdi.MyRoutes(MyRoutes.java:32)

 timer:foo?perid=5000

                perid    Unknown option. Did you mean: [period]


[WARNING] Endpoint validation error: (3 = passed, 1 = invalid, 0 = incapable, 0 = unknown components)
[INFO] Simple validation success: (0 = passed, 0 = invalid)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

D.4. 옵션

표 D.1.

매개변수	기본값	설명
downloadVersion	true	
failOnError	false	
logUnparseable	false	
includeJava	true	
includeXML	true	
includeTest	false	
includes	-	
excludes	-	
ignoreUnknownComponent	true	
ignoreIncapable	true	
ignoreLenientProperties	true	

매개변수	기본값	설명
showAll	false	

D.5.

절차

-

```
$cd myproject
$mvn io.fabric8.forge:fabric8-camel-maven-plugin:2.3.80:validate -DincludeTest=true
```

부록 E.

E.1.

E.2.

절차

-

```
s2i build <git repo url> registry.redhat.io/fuse7/fuse-karaf-openshift:1.6 <target image name>
docker run <target image name>
```

E.2.1.

-
-
-

E.2.2.

> `Karaf4: install karaf:assembly karaf:archive -DskipTests -e`

E.3.

-
-
-
-
-

E.4.

-
-
-
  ***JAVA_OPTIONS: java**를 호출할 때 추가할 옵션입니다.*
-
  ***JAVA_MAX_MEM_RATIO: JAVA_OPTIONS**에서 **-Xmx** 옵션이 제공되지 않을 때 사용됩니다. 이는 컨테이너 제한을 기반으로 기본 최대 힙 메모리를 계산하는 데 사용됩니다. 컨테이너에 대한 메모리 제약 조건 없이 **Docker** 컨테이너에서 사용하는 경우 이 옵션은 적용되지 않습니다.*

- *JAVA_MAX_CORE: 사용 가능한 코어 수를 수동으로 제한하며 가비지 수집기 스레드 수와 같은 특정 기본값을 계산하는 데 사용됩니다. 0으로 설정하면 코어 수에 따라 기본 JVM 튜닝을 수행할 수 없습니다.*

- *JAVA_DIAGNOSTICS: 상황이 발생할 때 표준으로 일부 진단 정보를 가져오도록 설정합니다.*

- *JAVA_MAIN_CLASS: java 인수로 사용할 기본 클래스입니다. 이 환경 변수를 사용하면 $JAVA_APP_DIR 디렉터리의 모든 Cryostat 파일이 classpath 및 $JAVA_LIB_DIR 디렉터리에 추가됩니다.*

- *JAVA_APP_JAR: java -jar 로 시작할 수 있도록 적절한 매니페스트가 있는 파일 그러나 제공되지 않는 경우 $JAVA_MAIN_CLASS 가 설정됩니다. 모든 경우에서 이 Cryostat 파일은 classpath에 추가됩니다.*

- *JAVA_APP_NAME: 프로세스에 사용할 이름입니다.*

- *JAVA_CLASSPATH: 사용할 classpath 지정하지 않으면 시작 스크립트에서 ${JAVA_APP_DIR}/classpath 파일을 확인하고 해당 콘텐츠를 classpath로 사용합니다. 이 파일이 존재하지 않는 경우 애플리케이션 디렉터리의 모든 Cryostat가 (classes:${JAVA_APP_DIR}/*) 아래에 추가됩니다.*

- *JAVA_DEBUG: 설정된 경우 원격 디버깅이 켜집니다.*

- *JAVA_DEBUG_PORT: 원격 디버깅에 사용되는 포트입니다. 기본값은 5005입니다.*

E.5. JOLOKIA 구성

Jolokia에서 다음 환경 변수를 사용할 수 있습니다.

- *AB_JOLOKIA_OFF: 설정된 경우 Jolokia의 활성화를 비활성화하십시오 (빈 값 선택). 기본적으로 Jolokia가 활성화됩니다.*

- *AB_JOLOKIA_CONFIG: 설정된 경우 파일을 (경로 포함)를 Jolokia JVM 에이전트 속성으로 사용합니다. 그러나 설정되지 않은 경우 설정을 사용하여 /opt/jolokia/etc/jolokia.properties 가*

생성됩니다.

- **AB_JOLOKIA_HOST:** 바인딩할 호스트 주소 (기본값 **0.0.0.0**)
- **AB_JOLOKIA_PORT:** 사용할 포트 (기본값은 **8778**)
- **AB_JOLOKIA_USER:** 기본 인증을 위한 사용자 기본적으로 **jolokia** 입니다.
- **AB_JOLOKIA_PASSWORD:** 기본 인증을 위한 암호 기본적으로 인증은 꺼집니다.
- **AB_JOLOKIA_PASSWORD_RANDOM:** 값을 생성하고 **/opt/jolokia/etc/jolokia.pw** 파일에 작성됩니다.
- **AB_JOLOKIA_HTTPS: HTTPS** 와의 보안 통신을 통해 전환 기본적으로 **AB_JOLOKIA_OPTS** 에는 **serverCert** 구성이 제공되지 않는 경우 자체 서명된 서버 인증서가 생성됩니다.
- **AB_JOLOKIA_ID:** 사용할 에이전트 **ID**
- **AB_JOLOKIA_DISCOVERY_ENABLED: Jolokia** 검색을 활성화합니다. 기본값은 **false**입니다.
- **AB_JOLOKIA_OPTS:** 에이전트 구성에 추가 옵션 추가 옵션은 **key=value** 형식으로 제공됩니다.

다음은 다양한 환경과의 통합을 위한 옵션입니다.

- **AB_JOLOKIA_AUTH_OPENSHIFT: OpenShift TSL** 통신을 위한 클라이언트 인증으로 전환 이 매개변수의 값이 클라이언트 인증서에 있어야 합니다. 이 매개변수를 활성화하면 **Jolokia**를 **HTTPS** 통신 모드로 자동 전환합니다. 기본 **CA** 인증서는 **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt** 로 설정됩니다.

JAVA_ARGS 변수를 해당 값으로 설정하여 애플리케이션 인수를 제공할 수 있습니다.

부록 F. LINUX 컨테이너에서 실행되도록 JVM 튜닝

JVM ergonomics 가 가비지 수집기, 힙 크기 및 런타임 컴파일러에 대한 기본값을 설정할 수 있는 경우 Linux 컨테이너 내에서 실행되는 Java 프로세스는 예상대로 작동하지 않습니다. 튜닝 매개변수를 조정하지 않고 Java 애플리케이션을 실행하는 경우, java -jar mypplication-fat.jar Cryostat- Cryostat-everything JVM은 컨테이너 제한이 아닌 호스트 제한에 따라 여러 매개변수를 자동으로 설정합니다.

이 섹션에서는 Linux 컨테이너 내에서 Java 애플리케이션 패키징에 대한 정보를 제공하여 기본값을 계산하기 위해 컨테이너의 제한을 고려합니다.

F.1. JVM 튜닝

현재 생성된 Java JVM은 컨테이너를 인식하지 않으므로 컨테이너 크기가 아니라 물리적 호스트의 크기에 따라 리소스를 할당합니다. 예를 들어 JVM은 일반적으로 최대 힙 크기를 호스트의 실제 메모리의 1/4로 설정합니다. 대규모 호스트 시스템에서 이 값은 컨테이너에 정의된 메모리 제한을 쉽게 초과할 수 있으며 런타임에 컨테이너 제한을 초과하면 OpenShift에서 애플리케이션을 종료합니다.

이 문제를 해결하려면 Java JVM이 제한된 컨테이너 내에서 실행되고 수동으로 수행하지 않는 경우 최대 힙 크기를 자동으로 조정할 수 있는 OpenShift 기본 이미지에서 Fuse를 사용할 수 있습니다. 애플리케이션을 실행하는 JVM에서 최대 메모리 제한과 코어 제한을 설정하는 솔루션을 제공합니다. OpenShift 이미지의 Fuse의 경우 다음을 수행할 수 있습니다.

- 컨테이너 코어를 기반으로 CICompilerCount 설정
- 컨테이너 메모리 제한이 300MB 미만인 경우 C2#159 컴파일러 비활성화
- 300MB 미만의 경우 기본 힙 크기에 대해 컨테이너 메모리 제한의 1/4을 사용합니다.

F.2. OPENSHIFT 이미지에서 FUSE ON의 기본 동작

OpenShift의 Fuse에서 애플리케이션 빌드의 기본 이미지는 Java 이미지(프링 부팅 애플리케이션의 경우) 또는 Karaf 이미지(카운프 애플리케이션용)일 수 있습니다. OpenShift 이미지의 Fuse는 컨테이너 제한을 읽고 리소스 할당을 기반으로 이러한 제한을 사용하는 스크립트를 실행합니다. 기본적으로 스크립트는 다음 리소스를 JVM에 할당합니다.

- 컨테이너 메모리 제한의 50%

- *컨테이너 코어 제한의 **50%**입니다.*

*여기에는 몇 가지 예외가 있습니다. **Karaf** 및 **Java** 이미지의 경우 물리적 메모리가 **300MB** 임계값보다 작으면 힙 크기가 **1-half** 대신 기본 힙 크기의 **1/4**로 복원됩니다.*

F.3. OPENSHIFT 이미지에서 FUSE의 사용자 정의 튜닝

*이 스크립트는 내부 리소스를 조정하기 위해 사용자 지정 애플리케이션에서 읽은 **CONTAINER_MAX_MEMORY** 및 **CONTAINER_CORE_LIMIT** 환경 변수를 설정합니다. 또한 애플리케이션을 실행하는 **JVM**에서 설정을 사용자 지정할 수 있는 다음 런타임 환경 변수를 지정할 수 있습니다.*

- ***JAVA_OPTIONS***
- ***JAVA_MAX_MEM_RATIO***

*제한을 명시적으로 사용자 지정하려면 **Maven** 프로젝트에서 **deployment.yml** 파일을 편집하여 **JAVA_MAX_MEM_RATIO** 환경 변수를 설정할 수 있습니다.*

예제

```
spec:
  template:
    spec:
      containers:
        -
          resources:
            requests:
              cpu: "0.2"
               memory: 256Mi
            limits:
              cpu: "1.0"
               memory: 256Mi
          env:
          - name: JAVA_MAX_MEM_RATIO
            value: 60
```

F.4. 타사 라이브러리 튜닝

***Red Hat**은 모든 타사 **Java** 라이브러리의 제한을 사용자 지정하는 것이 좋습니다. 이러한 라이브러리는 수동으로 제한을 사용자 지정하지 않는 경우 지정된 기본 제한을 사용합니다. 시작 스크립트는 애플리케이션에서 사용할 수 있는 컨테이너 제한을 설명하는 일부 환경 변수를 노출합니다.*

CONTAINER_CORE_LIMIT

계산된 코어 제한

CONTAINER_MAX_MEMORY

컨테이너에 부여된 메모리 제한