



# **Reference Architectures 2017**

## **Business Process Management with Red Hat JBoss BPM Suite 6.3 on OpenShift Container Platform 3.3**

---

Jeremy Ary

Babak Mozaffari



# Reference Architectures 2017 Business Process Management with Red Hat JBoss BPM Suite 6.3 on OpenShift Container Platform 3.3

---

Jeremy Ary

Babak Mozaffari  
refarch-feedback@redhat.com

## Legal Notice

Copyright © 2017 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

This reference architecture reviews utilization of kie-server capabilities built into Red Hat JBoss BPM Suite (BPMS) 6.3 and walks through the design, implementation and deployment of a sample BPM application to Red Hat OpenShift Container Platform utilizing a PatternFly- and AngularJS-based UI alongside the exposed REST and Java application interfaces to provide a custom-built, business-specific user interface.

## Table of Contents

<b>COMMENTS AND FEEDBACK</b>	<b>3</b>
<b>CHAPTER 1. EXECUTIVE SUMMARY</b>	<b>4</b>
<b>CHAPTER 2. REFERENCE ARCHITECTURE ENVIRONMENT</b>	<b>5</b>
2.1. PROJECT MODULES	5
2.2. CREDIT SERVICE	5
2.3. MORTGAGE RULES AND INTELLIGENT PROCESS SERVER	5
2.4. MORTGAGE DASHBOARD	6
2.5. OPENSIFT CONTAINER PLATFORM 3.3	7
<b>CHAPTER 3. CREATING THE ENVIRONMENT</b>	<b>12</b>
3.1. PREREQUISITES	12
3.2. OPENSIFT CONTAINER PLATFORM	13
<b>CHAPTER 4. DESIGN AND DEVELOPMENT</b>	<b>22</b>
4.1. CREDIT SERVICE WEB SERVICE	22
4.2. MORTGAGE LOAN RULES	22
4.3. MORTGAGE DASHBOARD	24
<b>CHAPTER 5. BUSINESS PROCESS USAGE VIA USER INTERFACE</b>	<b>36</b>
5.1. APPLICATION	36
5.2. DATA CORRECTION	37
5.3. CREDIT SERVICE TROUBLESHOOTING	38
5.4. MORTGAGE CALCULATIONS AND BORROWER QUALIFICATION	38
5.5. DOWN PAYMENT ADJUSTMENT	39
5.6. FINANCIAL REVIEW	39
5.7. APPRAISAL	40
5.8. FINALIZED APPLICATIONS	40
<b>CHAPTER 6. CONCLUSION</b>	<b>42</b>
<b>APPENDIX A. REVISION HISTORY</b>	<b>43</b>
<b>APPENDIX B. CONTRIBUTORS</b>	<b>44</b>
<b>APPENDIX C. REVISION HISTORY</b>	<b>45</b>



## COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing [refarch-feedback@redhat.com](mailto:refarch-feedback@redhat.com). Please refer to the title within the email.

## CHAPTER 1. EXECUTIVE SUMMARY

With the increased prevalence of automation, service integration and electronic data collection, it is prudent for any business to take a step back and review the design and efficiency of its business processes. Paired with container technologies, business process management can be further enhanced by increasing deployment efficiency, environmental flexibility, and collaboration.

A business process is a defined set of business activities that represents the steps required to achieve a business objective. It includes the flow and use of information and resources. **Business Process Management (BPM)** is a systematic approach to defining, executing, managing and refining business processes. Processes typically involve both machine and human interactions, integrate with various internal and external systems, and include both static and dynamic flows that are subject to both business rules and technical constraints.

Container technologies, such as **Red Hat OpenShift Container Platform 3.3**, allow for simplified bundling of an entire runtime environment, including the application, its dependencies, libraries, configuration, and more. By packaging the project in such a way, differences in OS distributions and underlying infrastructures are abstracted away, thus allowing for simplified replication and deployment across multiple environments such as development, testing, and production.

This reference architecture reviews utilization of the *kie-server* capabilities built into **Red Hat JBoss BPM Suite (BPMS) 6.3** and walks through the design, implementation and deployment of a sample BPM application to Red Hat OpenShift Container Platform which utilizes a PatternFly and AngularJS-based UI alongside the exposed REST and Java application interfaces to provide a custom-built, business-specific interface rather than demonstrating the process monitoring and management capabilities of *business-central*. Within time and scope constraints, potential challenges are discussed, along with common solutions to each problem. A repository is provided that can be cloned directly to reproduce the application assets and mimic OpenShift deployments as seen herein. Artifacts, including supporting code, are included in the repository.



## CHAPTER 2. REFERENCE ARCHITECTURE ENVIRONMENT

### 2.1. PROJECT MODULES

This reference architecture takes advantage of the provided [Red Hat JBoss Intelligent Process Server](#) and [Enterprise Application Platform 7](#) container images and templates. The application discussed within this reference architecture consists of three modules:

- ✳ *ocp-credit-service*: A simple web service which takes a mortgage loan applicant's social security number and returns their credit score. The simple Credit Report Web Service source code has been included in the resources accompanying this document.
- ✳ *ocp-mortgage-rules*: The rule repository, or *Knowledge Store*, which houses the rules and BPMN 2.0 process definitions, which together define the Mortgage Loan business process. The rule repository can also be found within the resources accompanying this document.
- ✳ *ocp-mortgage-dashboard*: Each step within the Mortgage Loan business process (e.g., submitting an application, data correction, appraisal, final approval, and so forth) is represented as a tab in the user interface. Relevant information regarding the various steps is fetched, modified accordingly, and submitted via the UI in order to move the loan process along. While the final source code can be found within the resources accompanying this document, it's recommended that the user follow the exercises outlined herein to establish a similar project of their own as this portion encapsulates the most pertinent information in working with BPM Suite capabilities on OpenShift Container Platform (OCP).

### 2.2. CREDIT SERVICE

The Credit Service is a Java EE web service hosted on a JBoss Enterprise Application Platform 7 container image on the OpenShift Container Platform (discussed in further detail below). The Mortgage Loan business process which will reside on the Intelligent Process Server contains a step which will reach out to the credit service in order to determine the credit score of an applicant for inclusion in the application information prior to proceeding on to borrower qualification.

### 2.3. MORTGAGE RULES AND INTELLIGENT PROCESS SERVER

As a review, Red Hat JBoss BPM Suite (BPMS) 6.3 is an open source BPM suite that combines business process management and business rules management, enabling business and IT users to create, manage, validate, and deploy business processes and rules. **BPMS 6.3** provides advanced business process management capabilities compliant with the widely adopted **BPMN 2.0** standard. The primary goal of BPMN 2.0 is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, [BPMN 2.0](#) creates a standardized bridge for the gap between the business process design and process implementation.

BPMS 6.3 includes **Realtime Decision Server** and **Intelligent Process Server**, both which serve as standalone, out-of-the-box components that can be used to instantiate and execute rules and processes, respectively, in provisioned containers through interfaces available for REST, JMS or a Java client side application. These self-contained environments encapsulate a compiled rule package and deployed rule instance, thus allowing for a looser coupling of rules and the execution code of the application. Both decision servers ship with default extensions for JBoss BRMS and BPM Suite via Intelligent Process Server. These servers, based on the KIE Execution

Server, have low footprints, with minimal memory consumption, and therefore, can be deployed easily on a cloud instance. Each instance can open and instantiate multiple KIE Containers which allows execution of multiple rule services in parallel. Further reading on the KIE Servers' advantages of decoupling rules execution from authorship and maintenance, more information on the deployment, management, and interacting with the Realtime Decision and Intelligent Process Servers can be found in the [Red Hat JBoss BRMS 6.3 User Guide](#).

Business rules, process definition files and other assets and resources created in BPM Suite's Business Central during rule development phases are stored in the asset repository called the **Knowledge Store**. The Knowledge Store is a centralized repository for business knowledge which uses a Git repository to store its data. This document is not intended to cover the basics of authoring processes via BPM Suite functionality, already covered in length as part of the [Business Process Management with Red Hat JBoss BPM Suite 6.3 Reference Architecture](#), and Mortgage Loan business process created via Business Central therein. This paper will, however, build on the concepts and rules developed within that project, using the Knowledge Store as the executable rule base that's called by our new user interface and backend. With the process already authored, the new application will provide a means to progress a loan through the various steps included in the aforementioned business process. Given the relation, familiarity with the Business Central example from the linked paper will greatly increase the understanding of which components are emulated via REST and Java application interfaces herein, thus it's recommended that you work through that project or, at the least, briefly familiarize yourself with the functionality it contains prior to proceeding.

## 2.4. MORTGAGE DASHBOARD

The User Interface is written in Java and JavaScript using the PatternFly and AngularJS frameworks. The Java portion of the application utilizes the *kie-server's* REST and Java APIs to query, trigger, and manage steps throughout the Mortgage Loan business process.

### 2.4.1. PatternFly

PatternFly is an open source project that promotes design commonality and improved user experience. It's a place where open source developers and designers can collaborate, create, and share user interface (UI) components and widgets. PatternFly is based on Bootstrap, a mobile-first front-end framework for creating web sites and applications. More information on the PatternFly library, including available styles and guides for working with the framework, can be found at the [official PatternFly website](#).

### 2.4.2. AngularJS

According to the official [AngularJS Documentation](#), "AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology." The JavaScript framework allows for declarative decoupling of DOM logic from application logic, thereby giving a clean, straight-forward approach to utilizing controllers and services to interact with the Mortgage Dashboard's backend Java functionality.

### 2.4.3. Remote KIE Server API

JBoss BPM offers a number of *Client* classes in the *org.kie.server.client* package for accessing and manipulating business processes remotely via Java. Some of the most prominent classes available follow:

- ✧ *KieServicesConfiguration*: used to specify configuration details, such as the REST URL, user name, user password, marshalling format, and more.
- ✧ *KieServicesClient*: Base class for all clients which houses a few shared functionalities, such as container manipulation and server interaction.
- ✧ *KieServicesFactory*: Factory class responsible for building *KieServicesConfiguration* objects as well as new use-specific *KieServicesClient* instances.
- ✧ *ProcessServicesClient*: allows fetching of process details, process manipulation, and user task input/output definitions
- ✧ *QueryServicesClient*: used for fetching of processes using a multitude of different search criteria as well as querying against the ruleset.
- ✧ *RuleServicesClient*: useful when needing to perform command execution against the ruleset.
- ✧ *UserTaskServicesClient*: responsible for fetching, claiming, starting, stopping, releasing, etc. tasks. Also can be used to fetch task Input/Output by taskId and manipulate comments and attachments related to a task.

Example client configuration and instantiation:

```
KieServicesClient client = null;
    try {

        KieServicesConfiguration conf =
KieServicesFactory.newRestConfiguration(
            Configuration.REST_BASE_URI,
            user.username(),
            user.password());

        conf.setMarshallingFormat(MarshallingFormat.JSON);

        client = KieServicesFactory.newKieServicesClient(conf);

    } catch (Exception e) {
        logger.error("ERROR in initializing REST client...", e);
    }
    return client;
```

## 2.5. OPENSIFT CONTAINER PLATFORM 3.3

Red Hat OpenShift Container Platform 3.3, or OCP, provides compute services on-demand. It allows developers to create and deploy applications by delivering a consistent environment for both development and run-time life-cycle that requires no server management. It also enforces software management and aids in the development and operations management of services by providing visibility and stability for the users.

OCP is supported across multiple platforms, however, the infrastructural details and administration of the base OCP system is beyond the focus of this document, which is intended as a guide for developer-level usage of the OCP platform and its various features. Rather than delve into details on how to establish the OCP deployment with redundancy and interactivity on a host platform like *RHOSP*, this paper aims to assist with the usage of images and templates to generate container-capable server/application instances and describe other tooling available via the OCP command line interface which allows a developer to build, deploy, and test their application in the development phase. Once complete, the container can be promoted to other environments as-is so that others

may utilize and experience the instances consistently with that which was used in development. Support documentation for installation on other platforms may also be found in the [Red Hat Reference Architecture Catalog](#).

### 2.5.1. Command Line Interface

The OpenShift Container Platform provides a command line interface (CLI) for creating applications and managing OpenShift projects from a terminal. The CLI is ideal in situations where you are working directly with project source code, scripting OpenShift operations, have restricted bandwidth resources.

The CLI is available using the `oc` command:

```
$ oc <command>
```

For further information, refer to the official [Red Hat OpenShift Documentation](#) for installation and setup instructions.

### 2.5.2. Image Streams

Containers are based on images. An image is a binary that includes all of the requirements for running a single container, as well as metadata describing its needs and capabilities. You can think of it as a packaging technology. Containers only have access to resources defined in the image, unless you give the container additional access when creating it. As alluded to previously, by deploying the same image in multiple containers across multiple hosts and load balancing between them, OpenShift can provide redundancy and horizontal scaling for a service packaged into an image.

OpenShift also supplies builders that assist with creating an image by adding your code or configuration to existing images. An image stream can be used to automatically perform an action, such as updating a deployment, when a new image, such as a new version of the base image that is used in that deployment, is created. An image stream comprises one or more Docker images identified by tags. It presents a single virtual view of related images, similar to a Docker image repository, and may contain images from any of the following: its own image repository in OpenShift's integrated Docker Registry, other image streams, or Docker image repositories from external registries. OpenShift components such as builds and deployments can watch an image stream to receive notifications when new images are added and react by performing a build or a deployment.

Example Image Stream Object Definition:

```
{
  "kind": "ImageStream",
  "apiVersion": "v1",
  "metadata": {
    "name": "origin-ruby-sample",
    "namespace": "p1",
    "selfLink": "/osapi/v1/namespaces/p1/imageStreams/origin-ruby-sample",
    "uid": "480dfe73-f340-11e4-97b5-001c422dcd49",
    "resourceVersion": "293",
    "creationTimestamp": "2015-05-05T16:03:34Z",
    "labels": {
      "template": "application-template-stibuild"
    }
  }
}
```

```

    },
    "spec": {},
    "status": {
      "dockerImageRepository": "172.30.30.129:5000/p1/origin-ruby-
sample",
      "tags": [
        {
          "tag": "latest",
          "items": [
            {
              "created": "2015-05-05T16:05:47Z",
              "dockerImageReference": "172.30.30.129:5000/p1/origin-
ruby-
sample@sha256:4d3a646b58685449179a0c61ad4baa19a8df8ba668e0f0704b9ad16f5
e16e642",
              "image":
"sha256:4d3a646b58685449179a0c61ad4baa19a8df8ba668e0f0704b9ad16f5e16e64
2"
            }
          ]
        }
      ]
    }
  }
}

```

### 2.5.3. Source to Image (S2I)

Image Streams usage allows for applications to be injected into containers via source-to-image. S2I is a tool for building reproducible Docker images; the new image incorporates the base image (the builder) and built source and is ready to use with the docker run command. S2I supports incremental builds, which re-use previously downloaded dependencies, previously built artifacts, etc. In short, S2I allows us to take an EAP or Intelligent Process Server Image which establishes the base requirements for the container and inject our rules or application into the image.

### 2.5.4. Templates

A template describes a set of objects that can be parameterized and processed to produce a list of objects for creation by OpenShift. The objects to create can include anything that users have permission to create within a project, for example services, build configurations, and deployment configurations. A template may also define a set of labels to apply to every object defined in the template.

A template describes a set of related object definitions to be created together, as well as a set of parameters for those objects. For example, an application might consist of a front-end web application backed by a database; each consists of a service object and deployment configuration object, and they share a set of credentials (parameters) for the front-end to authenticate to the backend. The template can be processed, either specifying parameters or allowing them to be automatically generated (for example, a unique DB password), in order to instantiate the list of objects in the template as a cohesive application.

Example YAML Template Object Definition:

```

apiVersion: v1
kind: Template
metadata:

```

```

name: redis-template
annotations:
  description: "Description"
  iconClass: "icon-redis"
  tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
  parameters:
  - description: Password used for Redis authentication
    from: '[A-Z0-9]{8}'
    generate: expression
    name: REDIS_PASSWORD
  labels:
    redis: master

```

### 2.5.5. Intelligent Process Server xPaaS Image

This image provides a means for executing business processes on JBoss BPMS Intelligent Process Server 6.3, which is a modular, standalone server component that can be used to instantiate and execute rules and processes. It exposes this functionality through REST, JMS and Java interfaces to client applications. There are a variety of images available based on persistence and interconnectivity needs. For the purposes of this document, the *basic* image is utilized, which includes a simple H2 database, although images exist for inclusion of both MySQL and PostgreSQL with either HornetQ or ActiveMQ messaging functionality.

### 2.5.6. Enterprise Application Platform (JBoss EAP) xPaaS Image

JBoss EAP, Red Hat's Java EE-based application server runtime, is utilized as the container for deployment of the credit check web service, PatternFly, and AngularJS-based front-end application. Various capabilities provided by the container are leveraged, such as Weld Contexts, Dependency Injection and more. It should be noted that there are significant differences in supported configurations and functionality in the JBoss EAP xPaaS image compared to the regular release of JBoss EAP. Some of these differences are as follows:

- ✎ The JBoss EAP Management Console is not available to manage OpenShift JBoss EAP xPaaS images.
- ✎ The JBoss EAP Management CLI is only bound locally. This means that you can only access the Management CLI of a container from within the pod.
- ✎ Domain mode is not supported. OpenShift controls the creation and distribution of applications in the containers.

- ✱ The default root page is disabled. You may want to deploy your own application to the root context (as ROOT.war).
- ✱ A-MQ is supported for inter-pod and remote messaging. HornetQ is only supported for intra-Pod messaging and only enabled in the absence of A-MQ.

## CHAPTER 3. CREATING THE ENVIRONMENT

### 3.1. PREREQUISITES

This reference architecture assumes the developer's point of view in the application lifecycle, and thus has an expectation of an existing OpenShift Container Platform installation with development and administration users and roles properly defined. Further information on installation and administration of OSP on the Red Hat OpenStack Platform can be found in the [Deploying Red Hat OpenShift Container Platform 3 on Red Hat OpenStack Platform](#) Reference Architecture.

#### 3.1.1. Default Container Image Streams and Templates

The necessary Red Hat middleware xPaaS image streams and templates, if not already installed, should be added for working with the Intelligent Process Server and EAP images:

```
$ oc get is -n openshift | grep jboss
```

Figure 3.1. List Existing OpenShift Image Streams

```
[jary@bxms-ose-master ~]$ oc get is -n openshift | grep jboss
fis-java-openshift          registry.access.redhat.com/jboss-fuse-6/fis-java-openshift      1.0,1.0-10,1.0-11 + 2 more... 3 weeks ago
fis-karaf-openshift        registry.access.redhat.com/jboss-fuse-6/fis-karaf-openshift     1.0,1.0-10,1.0-11 + 2 more... 3 weeks ago
jboss-amq-62               registry.access.redhat.com/jboss-amq-6/amq62-openshift         1.1-2,1.2,1.3 + 2 more...    3 weeks ago
jboss-datagrid65-openshift registry.access.redhat.com/jboss-datagrid-6/datagrid65-openshift 1.2,1.2-13,1.2-18 + 2 more... 3 weeks ago
jboss-decisionserver62-openshift registry.access.redhat.com/jboss-decisionserver-6/decisionserver62-openshift 1.2-18,latest,1.2 + 2 more... 3 weeks ago
jboss-eap64-openshift      registry.access.redhat.com/jboss-eap-6/eap64-openshift         1.3,1.4,latest + 2 more...    3 weeks ago
jboss-eap70-openshift      registry.access.redhat.com/jboss-eap-7/eap70-openshift         1.3-22,1.4,1.4-10 + 2 more... 3 weeks ago
jboss-processserver63-openshift registry.access.redhat.com/jboss-processserver-6/processserver63-openshift 1.3,1.3-17,latest             3 months ago
jboss-webserver30-tomcat7-openshift registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat7-openshift latest,1.1,1.1-2 + 2 more... 3 weeks ago
jboss-webserver30-tomcat8-openshift registry.access.redhat.com/jboss-webserver-3/webserver30-tomcat8-openshift 1.1,1.1-3,1.1-7 + 2 more... 3 weeks ago
[jary@bxms-ose-master ~]$
```

```
$ oc get is -n templates | grep eap
```

Figure 3.2. List Existing OpenShift Templates

```
[jary@bxms-ose-master ~]$ oc get templates -n openshift | grep eap
eap64-amq-persistent-s2i    Application template for EAP 6 A-MQ applications with persistent storage built using S2I. 28 (6 blank) 10
eap64-amq-s2i              Application template for EAP 6 A-MQ applications built using S2I. 26 (6 blank) 9
eap64-basic-s2i            Application template for EAP 6 applications built using S2I. 12 (3 blank) 5
eap64-https-s2i            Application template for EAP 6 applications built using S2I. 21 (8 blank) 7
eap64-mongodb-persistent-s2i Application template for EAP 6 MongoDB applications with persistent storage built using S2I. 33 (15 blank) 10
eap64-mongodb-s2i         Application template for EAP 6 MongoDB applications built using S2I. 32 (15 blank) 9
eap64-mysql-persistent-s2i Application template for EAP 6 MySQL applications with persistent storage built using S2I. 34 (16 blank) 10
eap64-mysql-s2i           Application template for EAP 6 MySQL applications built using S2I. 33 (16 blank) 9
eap64-postgresql-persistent-s2i Application template for EAP 6 PostgreSQL applications with persistent storage built using S2I. 31 (13 blank) 10
eap64-postgresql-s2i      Application template for EAP 6 PostgreSQL applications built using S2I. 30 (13 blank) 9
eap64-sso-s2i             Application template for EAP 6 applications built using S2I, enabled for SSO. 36 (11 blank) 7
eap70-amq-persistent-s2i   Application template for EAP 7 A-MQ applications with persistent storage built using S2I. 28 (6 blank) 10
eap70-amq-s2i             Application template for EAP 7 A-MQ applications built using S2I. 26 (6 blank) 9
eap70-basic-s2i           Application template for EAP 7 applications built using S2I. 12 (3 blank) 5
eap70-https-s2i           Application template for EAP 7 applications built using S2I. 21 (8 blank) 7
eap70-mongodb-persistent-s2i Application template for EAP 7 MongoDB applications with persistent storage built using S2I. 33 (15 blank) 10
eap70-mongodb-s2i        Application template for EAP 7 MongoDB applications built using S2I. 32 (15 blank) 9
eap70-mysql-persistent-s2i Application template for EAP 7 MySQL applications with persistent storage built using S2I. 34 (16 blank) 10
eap70-mysql-s2i           Application template for EAP 7 MySQL applications built using S2I. 33 (16 blank) 9
eap70-postgresql-persistent-s2i Application template for EAP 7 PostgreSQL applications with persistent storage built using S2I. 31 (13 blank) 10
eap70-postgresql-s2i     Application template for EAP 7 PostgreSQL applications built using S2I. 30 (13 blank) 9
[jary@bxms-ose-master ~]$
```

Should you find that the required images and/or templates aren't presently available, more information on loading them can be found in the relevant section of the [OpenShift Container Platform 3.3 Installation and Configuration Guide](#).

#### 3.1.2. Downloads



This reference architecture makes mention and use of several files associated to this document. These files will be used in configuring the reference architecture environment and providing a copy of the application developed herein for reference. They can be found at the following public repository:

<https://github.com/RHsyseng/BPMS-OCF>

## 3.2. OPENSIFT CONTAINER PLATFORM

Each module of our project will be housed within OCP under its own project for separation of concerns purposes and ease of diagnostics and build processes, although it is possible to group multiple images and sources, as is discussed later in this chapter.

### 3.2.1. Credit Service

#### 3.2.1.1. Create New Project

If, as the developer, access to the *oadm* command is available, either via *sudo* or requesting assistance from the OpenShift Container Platform Administrator, create a new project for housing the Credit Service application and assign administrative rights of the new project to the user to be utilized by the developer from that point forward:

```
$ sudo oadm new-project creditservice --display-name="Credit Service" -
-description="This is a web service responsible for calculating loan
applicant credit scores" --admin=ocuser
```

#### 3.2.1.2. Authenticating and Navigating Projects

Once the new project is established, the developer can use *oc login* to assume the role assigned as the admin for the project before continuing. The shorthand form of logging in while also specifying the server and security certificate, is as follows:

```
$ oc login -u ocuser --certificate-authority=/etc/origin/master/ca.crt
--server=https://bxms-ose-master.cloud.lab.eng.bos.redhat.com:8443
```

If you log in after creating or being granted access to a project, a project you have access to is automatically set as the current default, until switching to another one. The following also demonstrates the interactive login prompt with password authentication in lieu of security certificate:

```
$ oc login
Username: alice
Authentication required for https://openshift.example.com (openshift)
Password:
Login successful.

Using project "exampleProject".
```

Switching between projects as necessary can be completed using the *oc project* command:

```
$ oc project differentProjectName
```

#### 3.2.1.3. Create New Application

Once the role of the project admin has been accessed, from this point on assumed to be *ocuser*, a new application can be created using the EAP 7.0 image stream:

```
$ oc new-app jboss-eap70-openshift~https://github.com/RHsyseng/BPMS-
OCP/ --context-dir=ocp-credit-service --name=cs-service
```

Note that with some images and applications, some environmental variables are required to be passed in, as during the S2I process, the image will detect and utilize these variables in scripts which are responsible for initializing or configuring the image and/or environment. To pass environment variables into a *new-app*, use the *-e* argument format as follows:

```
$ oc new-app jboss-processserver63-
openshift~https://github.com/RHsyseng/BPMS-OCP/ --context-dir=ocp-
mortgage-dashboard --name=mort-dashboard -e
SECDOMAIN_USERS_PROPERTIES=bpm_users.properties,SECDOMAIN_ROLES_PROPERT
IES=bpm_roles.properties,SECDOMAIN_NAME=bpm_domain,SECDOMAIN_PASSWORD_S
TACKING=true
```

### 3.2.1.4. Monitoring Application Build

Once a *new-app* command has been performed, OCP will create several artifacts related to the new application. The set of artifacts created depends on the artifacts passed as input: source repositories, images, or templates.

**Table 3.1. *new-app* Artifact Output**

Artifact	Description
BuildConfig	A BuildConfig is created for each source repository specified in the command line. The BuildConfig specifies the strategy to use, the source location, and the build output location.
ImageStreams	For BuildConfig, two ImageStreams are usually created: one to represent the input image (the builder image in the case of Source builds or FROM image in case of Docker builds), and another one to represent the output image. If a Docker image was specified as input to <i>new-app</i> , then an image stream is created for that image as well.
DeploymentConfig	A DeploymentConfig is created either to deploy the output of a build, or a specified image. The <i>new-app</i> command creates EmptyDir volumes for all Docker volumes that are specified in containers included in the resulting DeploymentConfig.

Artifact	Description
Service	The new-app command attempts to detect exposed ports in input images. It uses the lowest numeric exposed port to generate a service that exposes that port. In order to expose a different port, after new-app has completed, simply use the <code>oc expose</code> command to generate additional services.
Other	Other objects can be generated when instantiating templates.

#### Artifact Creation Example:

```
--> Creating resources with label app=cs-service ...
    imagestream "cs-service" created
    buildconfig "cs-service" created
    deploymentconfig "cs-service" created
    service "cs-service" created
--> Success
    Build scheduled, use 'oc logs -f bc/cs-service' to track its
    progress.
    Run 'oc status' to view your app.
```

The *buildconfig* artifact is of particular interest at this point as it's automatically triggered to begin the S2I process, which we can monitor, as the prompt indicates, by issuing the following command:

```
$ oc logs -f bc/cs-service
```

#### Example BuildConfig Log Excerpt:

```
$ oc logs -f bc/mort-dashboard
I1201 11:02:06.215391      1 builder.go:57] Master version "v3.2.1.13-1-gc2a90e1", Builder version "v3.2.1.13-1-gc2a90e1"
I1201 11:02:06.304554      1 builder.go:145] Running build with
cgroup limits: api.CGroupLimits{MemoryLimitBytes:92233720368547,
CPUShares:2, CPUPeriod:100000, CPUQuota:-1, MemorySwap:92233720368547}
I1201 11:02:06.330643      1 sti.go:206] The value of ALLOWED_UIDS is
[1-]
I1201 11:02:06.330677      1 sti.go:214] The value of DROP_CAPS is
[KILL,MKNOD,SETGID,SETUID,SYS_CHROOT]
I1201 11:02:06.352427      1 docker.go:355] Using locally available
image "registry.access.redhat.com/jboss-processserver-6/processserver63-
openshift@sha256:faf8fe2aa3747a51ff5aa9745278a0e9d7669f05f718773e195624
551428fc79"
I1201 11:02:06.356752      1 sti.go:233] Creating a new S2I builder
with build config: "Builder Name:\t\t\tJBoss BPMS Intelligent Process
Server 6.3\nBuilder Image:\t\t\tregistry.access.redhat.com/jboss-
processserver-6/processserver63-
```

[illegible]

At this point in the process, S2I is fetching the specified source code for injection into the supplied image. Once complete, the logs will continue feedback (much too long for full inclusion here) as the S2I process runs to completion. The process results in a docker container instance with the finalized image within ready to be exposed. During this process, S2I fetches and builds the code from the specified repository and pulls down all dependencies required for both build time and runtime offline capabilities. It's possible to specify a maven mirror/proxy to increase the speed of the downloads. Following the build, KIE Container verification is performed to validate all rules and process resources. With the verification complete, there's less potential for issues to arise around erroneous rules when the new container starts up. Given all the work that is performed at this time, this process may take some time to complete, so monitoring of logs or project status is recommended in order to be aware of the point at which the process is finished. If using the above command, this point is demarcated by an ending of the streaming log tail and a return to the prompt.

If using the `oc status` command, completion will be exemplified by a successful deployment similar to the following:

```
$ oc status
In project Credit Service Web Service (creditservice) on server
https://bxms-ose-master.cloud.lab.eng.bos.redhat.com:8443

http://cs.bxms.ose to pod port 8080-tcp (svc/cs-service)
  dc/cs-service deploys istag/cs-service:latest <-
    bc/cs-service builds https://github.com/jeremyary/jboss-os-
service.git with openshift/jboss-eap70-openshift:latest
```

```
deployment #2 deployed 2 days ago - 1 pod
deployment #1 deployed 4 days ago
```

```
1 warning identified, use 'oc status -v' to see details.
```

### 3.2.1.5. Exposing Service via Route

An OpenShift route exposes a service at a host name, like `www.example.com`, so that external clients can reach it by name.

DNS resolution for a host name is handled separately from routing; your administrator may have configured a cloud domain that will always correctly resolve to the OpenShift router, or if using an unrelated host name you may need to modify its DNS records independently to resolve to the router.

Once a build has been completed, it's necessary to expose the new service via routing, as follows:

```
$ oc expose service cs-service --hostname=www.example.com
```

Unsecured routes are the default configuration, and are therefore the simplest to set up. However, secured routes offer security for connections to remain private. To create a secured HTTPS route encrypted with a key and certificate (PEM-format files which you must generate and sign separately), you can use the `create route` command and optionally provide certificates and a key. More information on this can be found in the official [Red Hat OpenShift Developer Guide](#).

### 3.2.1.6. Listing Resources

At times, it may be beneficial to list all the entities associated with a project or application. To view all associated entities, use the `oc get all` command:

```
$ oc get all
NAME                                TYPE
FROM                                LATEST
cs-service                          Source
Git                                  2
NAME                                TYPE
FROM                                STATUS    STARTED    DURATION
cs-service-1                        Source
Git@16553c3                         Complete  2 days ago  5m15s
cs-service-2                        Source
Git@16553c3                         Complete  4 days ago  15m21s
NAME                                DOCKER REPO
TAGS                                UPDATED
cs-service                          172.30.14.135:5000/creditservice/cs-service
latest                              4 days ago
NAME                                REVISION
REPLICAS                            TRIGGERED BY
cs-service                          2                                                    1
config,image(cs-service:latest)
NAME                                DESIRED
CURRENT                            AGE
cs-service-1                        0                                                    0
12d
cs-service-2                        1                                                    1
10d
NAME                                HOST/PORT
```

PATH	SERVICE	TERMINATION	LABELS
cs-service	cs.bxms.ose		
cs-service:8080-tcp		app=cs-service	
NAME	CLUSTER-IP		
EXTERNAL-IP	PORT(S)	AGE	
cs-service	172.30.226.97		
<none>	8080/TCP,8443/TCP,8778/TCP	12d	
NAME	READY		
STATUS	RESTARTS	AGE	
cs-service-1-build	0/1		
Completed	0	12d	
cs-service-2-build	0/1		
Completed	0	10d	
cs-service-2-uep2k	1/1		
Running	0	10d	

In cases where further diagnostic troubleshooting is required, it's also possible to get the YAML-based configuration listing for all entities, by specifying the format of output to *get*:

```
$ oc get all -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: BuildConfig
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    creationTimestamp: 2016-09-30T18:20:03Z
    labels:
      app: cs-service
      name: cs-service
      namespace: creditservice
      resourceVersion: "2230152"
      selfLink: /oapi/v1/namespaces/creditservice/buildconfigs/cs-service
      uid: 81886a1f-873a-11e6-9013-0001a4ac33b4
  spec:
    output:
      to:
        kind: ImageStreamTag
        name: cs-service:latest
... (output abbreviated)
```

Rather than list all entities, it's also possible to generate a list of specific entity types. For instance, in order to list the pods associated with the project, specify *pods* with the *get* command:

```
$ oc get pods
NAME                READY    STATUS    RESTARTS    AGE
cs-service-1-build  0/1      Completed  0            12d
cs-service-2-build  0/1      Completed  0            10d
cs-service-2-uep2k  1/1      Running    0            10d
```

### 3.2.1.7. Monitoring Pods

As evident in the example above, a single pod is now running the container instance post-build completion. In order to view and diagnose issues on the server, such as viewing EAP logs, it's possible to access the output without directly connecting to the pod:

```
$ oc logs cs-service-2-uep2k
```

With the images in use as part of this project, the output for the above command resembles a *cat* of the server logs. Tailing of live logs is also possible with the *-f* parameter.

### 3.2.1.8. Accessing Pods

Should you further need to diagnose issues on the pod container instance, remote access to pod is performed as follows:

```
$ oc rsh cs-service-2-uep2k
sh-4.2$
```

While it's possible to test server configuration changes or tweak properties via the last command, it's highly recommended to avoid doing so as the changes made manually via *rsh* will **not** persist on a new build and further, will not replicate to other pods hosting your application. Should you need to make changes to the runtime environment, research the image stream and S2I process in use for possibilities of passing through properties or configuration that will instead persist on restarts and replication. Often times, S2I processes expose variables as needed for changing configuration. When working with JBoss EAP, it's possible to include configuration files in the code repository used to instantiate the BuildConfig which will be added to the *standalone* configuration directory for further reference.

### 3.2.1.9. Deleting Components and Projects

Should you wish to delete the entities associated to a created app, you may do so by referring to the entities by label:

```
$ oc delete all -l app=cs-service
```

It's also possible to delete a project in its entirety rather than resetting at the app level:

```
$ oc delete project cs-service
```

## 3.2.2. Mortgage Loan Business Process

### 3.2.2.1. Create New Project

As before, either via *sudo* or with Administrator assistance, create a new project for housing the Business Process rules application and assign administrative rights of the new project to the user to be utilized by the developer from that point forward:

```
$ sudo oadm new-project mortgage-rules --display-name="Mortgage Loan
Rules" --description="This is an Intelligent Process Server app housing
the Mortgage Loan Knowledge Store" --admin=ocuser
```

### 3.2.2.2. Create New Application

Initialize a new Intelligent Process Server application for hosting the Mortgage Loan process rules:

```
$ oc new-app jboss-processserver63-  
openshift~https://github.com/RHsyseng/BPMS-OCF --context-dir=ocp-  
mortgage-rules --name=mortgage-rule
```

Expose the new service upon completion.

### 3.2.3. Mortgage Dashboard

#### 3.2.3.1. Running Locally during Development

During development phases, it may be more practical to host the Dashboard User Interface portion of the project on a localhost server or local EAP instance rather than deploying and undergoing timely S2I processes with each iteration to OpenShift.

#### 3.2.3.2. Deploying to OpenShift Container Platform

Once again, create a new project for housing the Business Process rules application and assign administrative rights of the new project to the user to be utilized by the developer from that point forward:

```
$ sudo oadm new-project mortgage-dashboard --display-name="Mortgage  
Dashboard" --description="This is a User Interface Application intended  
to interact with the Mortgage Loan Business Process" --admin=ocuser
```

Once development of the application portion of the project has been completed, another EAP 7.0 container can be used for hosting it in a container:

```
$ oc new-app jboss-eap70-openshift~https://github.com/RHsyseng/BPMS-OCF  
--context-dir=ocp-mortgage-dashboard --name=mortgage-dashboard
```

#### 3.2.4. Logging Out of OpenShift

Once work has been completed or halted for a period of time, it's possible to log out of the role that the developer has assumed for administration of the various projects established:

```
oc logout
```

#### 3.2.5. Creating Multiple OpenShift Applications At Once

The new-app command allows creating multiple applications from source, images, or templates at once. To do this, simply specify multiple parameters to the new-app call. Labels specified in the command line apply to all objects created by the single call. Environment variables apply to all components created from source or images.

Creating an Application from a Source Repository and a Docker Hub Image:

```
$ oc new-app https://github.com/openshift/ruby-hello-world mysql
```

#### 3.2.6. Grouping OpenShift Images and Source in a Single Pod



The `new-app` command allows deploying multiple images together in a single pod using the `+` separator. The `--group` command line argument can also be used to specify which images should be grouped together:

To deploy two images in a single pod:

```
$ oc new-app nginx+mysql
```

To deploy an image built from source and an external image together:

```
$ oc new-app \  
  ruby~https://github.com/openshift/ruby-hello-world \  
  mysql \  
  --group=ruby+mysql
```

## CHAPTER 4. DESIGN AND DEVELOPMENT

### 4.1. CREDIT SERVICE WEB SERVICE

For the purpose of this reference architecture where the focus is on the BPM Suite, assume that an external web service provides the required information on the credit worthiness of mortgage applicants. For the sake of simplicity, create a basic Web Service that takes an applicant's social security number as its only input and returns their Credit Score as the result. Rather than recreating the service, if you would prefer, you can build a deployable .war file from the accompanying source code (ocp-credit-service module) for this document. Creating a simple Web Service using JSR-181 and JSR-224 requires a simple Web Application with an empty web.xml file and an annotated Java class:

```
package com.redhat.bpms.examples.mortgage;
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public class CreditService
{
    @WebMethod
    public Integer getCreditScore(Integer ssn)
    {
        int lastDigit = ssn - 10 * ( ssn / 10 );
        int score = 600 + ( lastDigit * 20 );
        System.out.println( "For ssn " + ssn + ", will return credit score of "
            + score );
        return score;
    }
}
```

This class simply uses the last digit of the social security number to mock up a credit score. The web deployment descriptor remains empty:

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemalocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
</web-app>
```

Once deployed to OpenShift Container Platform, the service can then be reached via URL similar to the following:

<http://cs.bxms.ose/jboss-project-name/creditservice?WSDL>

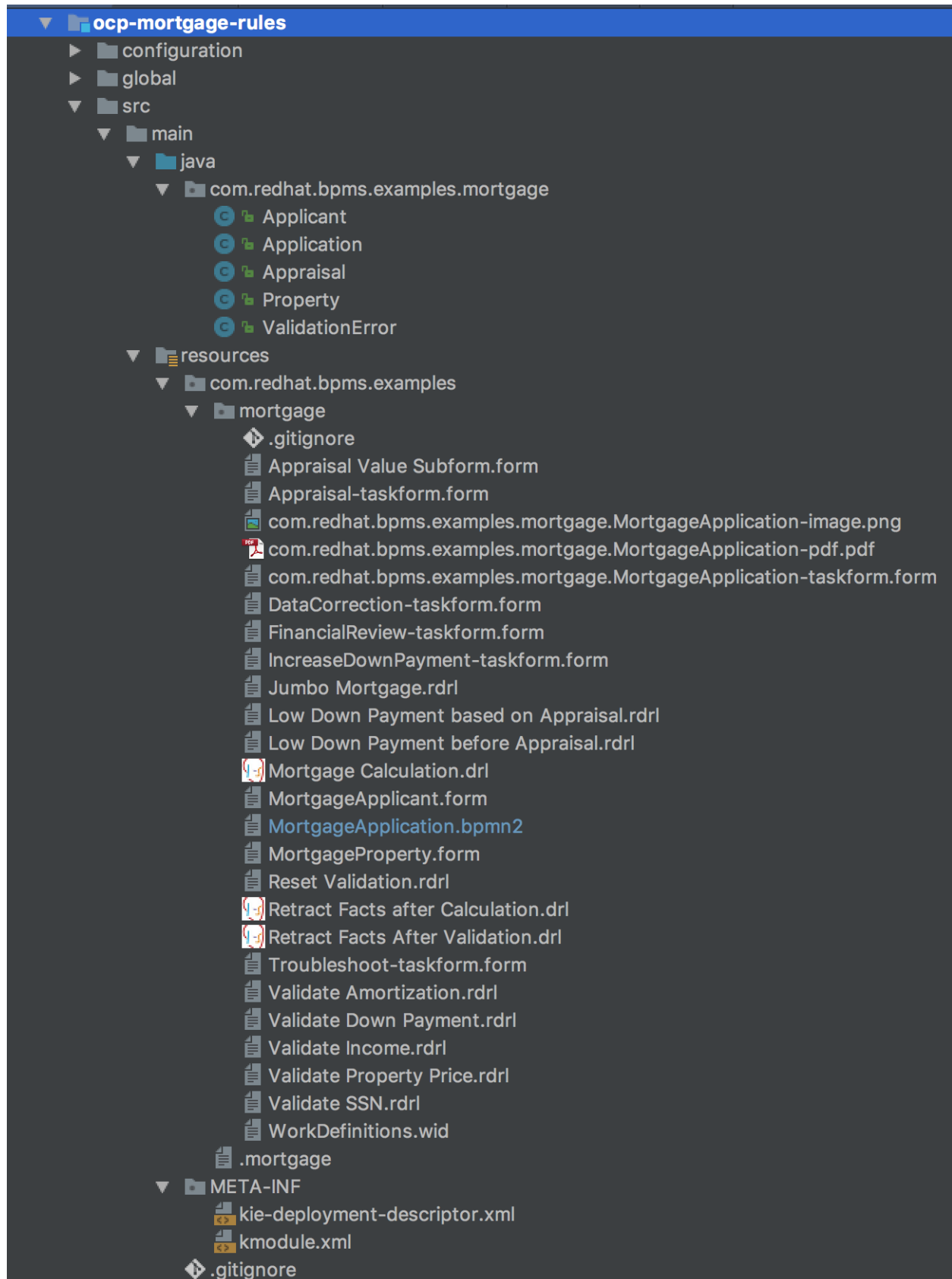
### 4.2. MORTGAGE LOAN RULES

As alluded to previously, the Knowledge Store portion of our project will be borrowed from a previously constructed project, discussed in greater detail as a part of the [Business Process Management with Red Hat JBoss BPM Suite 6.3 Reference Architecture](#).

#### 4.2.1. Knowledge Repository

The knowledge repository, once exported from an instance of Business Central (or similar authoring tool environment) will have the following structure:

**Figure 4.1. ocp-mortgage-rules Module Directory Structure**



A few changes are necessary prior to utilizing this repository for S2I image creation. First, the web service call URL and namespace can be modified, if necessary, with a search and replace within the .bpmn2 file associated to the business process.

Secondly, the Intelligent Process Server defaults to a very limited user and roles configuration at the server level. In order to allow multiple user types (such as broker, manager, appraiser) to perform functions via our User Interface, we need to see the S2I process with a few configuration files to bolster the list of users with access to the system.

Add a top-level folder called *configuration* with two files:

*application-roles.properties*:

```
kieserver=kie-server,guest
brokerUser=kie-server,user,broker
managerUser=kie-server,user,manager
appraiserUser=kie-server,user,appraiser
```

*application-users.properties*:

```
kieserver=16c6511893651c9b4b57e0c027a96075
brokerUser=f5aac98dcdfb823f6cc6c31f9ebbce7
managerUser=b4d2a2fd7c71bda4583f17d9d78e160a
appraiserUser=a02ebd14456015693ba2038c47a6a5d8
```

Note that the user *kieserver* will be overridden by the server on image creation. The passwords used for the *application-users* file above can be calculated at the terminal to fit the format expected within the server container. To determine a user password hash, perform the following command:

```
$ echo -n brokerUser:ApplicationRealm:password1! | openssl dgst -md5 -
binary | od -A n -t x1 | tr -d ' '
```

The format of the *-n* attribute value should follow the *username:app\_realm:password* format, where, unless specified differently via environmental variables, app realm will be *ApplicationRealm*. The examples above use *password1!* for every user. Should you wish to use different users or passwords, be sure to calculate and substitute in necessary values in both configuration files and the Java enum within the source code used as a lookup reference for available users.

Once defined and added as part of the S2I process, the content of these files (with the exception of the *kieserver* user), will get appended to the single default user setup created within the image.

## 4.3. MORTGAGE DASHBOARD

To this point, the code utilized for various deployments to OCP have either come from preexisting applications or exports. The final portion of the project, the user interface responsible for allowing interactivity with loan applications throughout the Mortgage Loan business process, differs in that it is original code demonstrating the capability of building *kie-server* API interfacing into a standalone project. This module consists of a PatternFly and AngularJS-based front-end and Java backend.

The source code for all modules, including this one, can be found in the repository referenced in this document, and previously linked. You may choose to start by instantiating a new module from scratch, using the accompanying code in its entirety, or using the code as a boilerplate by removing the majority of working files and keeping the directory structures and configuration portions intact.

### 4.3.1. User Interface

Further guidance and discussion regarding the main frameworks, PatternFly and AngularJS, can be found in the Red Hat Developers Blog article titled [Building JBoss Projects with PatternFly and AngularJS](#) and the [official Angular-PatternFly documentation](#).

#### 4.3.1.1. JavaScript and CSS Inclusions

The main entry point (in this use case, index.html) for the module application should include the following framework-required references and calls to script files created as a part of this module:

CSS Links:

```
<link rel="stylesheet" href="vendor/patternfly/dist/css/patternfly.min.css">
<link rel="stylesheet" href="vendor/patternfly/dist/css/patternfly-additions.min.css">
<link rel="stylesheet" href="vendor/angular-patternfly/dist/styles/angular-patternfly.min.css" />
<link rel="stylesheet" href="css/specific.css">
```

Script Links:

```
<script src="vendor/angular/angular.js"></script>
<script src="vendor/angular-route/angular-route.js"></script>
<script src="vendor/angular-resource/angular-resource.js"></script>
<script src="vendor/jquery/dist/jquery.js"></script>
<script src="vendor/bootstrap-select/dist/js/bootstrap-select.js"></script>
<script src="vendor/moment/moment.js"></script>
<script src="vendor/angular-route/angular-route.js"></script>
<script src="vendor/angular-sanitize/angular-sanitize.js"></script>
<script src="vendor/angular-animate/angular-animate.js"></script>
<script src="vendor/patternfly/dist/js/patternfly.js"></script>
<script src="vendor/angular-bootstrap/ui-bootstrap.js"></script>
<script src="vendor/angular-bootstrap/ui-bootstrap-tpls.js"></script>
<script src="vendor/bootstrap/dist/js/bootstrap.js"></script>
<script src="vendor/d3/d3.js"></script>
<script src="vendor/c3/c3.js"></script>
<script src="vendor/lodash/lodash.js"></script>
<script src="vendor/matchHeight/dist/jquery.matchHeight.js"></script>

<script src="js/app.js"></script>
<script src="js/controllers/home.js"></script>
<script src="js/controllers/appraiser.js"></script>
<script src="js/controllers/applicant.js"></script>
<script src="js/controllers/dataCorrection.js"></script>
<script src="js/controllers/manager.js"></script>
<script src="js/controllers/troubleshoot.js"></script>
<script src="js/controllers/downPayment.js"></script>
<script src="js/controllers/financialReview.js"></script>
```

#### 4.3.1.2. Bower Configuration

Bower, a dependency management (package manager) tool for front end utilities such as frameworks, libraries, assets, etc, can be configured with two files placed in the module's top-level directory for configuration of Bower:

*.bowerrc* - used to specify where third-party libraries should be included and referenced from within the module:

```
{
  "directory": "src/main/webapp/vendor"
}
```

*bower.json* - used to specify module dependencies and properties relevant to the project:

```
{
  "name": "ocp-mortgage-dashboard",
  "description": "",
  "main": "index.js",
  "authors": [
    "jary@redhat.com"
  ],
  "license": "Apache-2.0",
  "homepage": "",
  "private": true,
  "dependencies": {
    "angular": "^1.4.0",
    "angular-loader": "^1.4.0",
    "angular-mocks": "^1.4.0",
    "angular-route": "^1.4.0",
    "angular-resource": "^1.4.0",
    "bootstrap": "3.2.0",
    "angular-patternfly": "^3.13.0"
  },
  "ignore": [
    "**/*.\"",
    "node_modules",
    "bower_components",
    "vendor",
    "test",
    "tests"
  ],
  "resolutions": {
    "bootstrap": "~3.3.7",
    "angular": "1.5.9"
  }
}
```

#### 4.3.1.3. AngularJS Application Configuration

The main AngularJS configuration file, *app.js*, contains a few pertinent sections worth further inspection. The first of these is module inclusions:

```
var app = angular.module('app', [
  'ngRoute',
  'ngResource',
  'ngSanitize'
]);
```

##### 4.3.1.3.1. Constants

Next, we can define a set of constants to globally represent the various states that work items can be found in through the business process:

```
app.constant('Constants', {
  PROCESS_STATE: {
    0: 'PENDING',
```

```

1: 'ACTIVE',
2: 'COMPLETED',
3: 'ABORTED',
4: 'SUSPENDED'
}
});

```

#### 4.3.1.3.2. Routing

Lastly, we can configure the routing of the application to map addresses to relevant controllers and views:

```

app.config(['$routeProvider',
function ($routeProvider) {
    $routeProvider
        .when('/', {
            templateUrl: 'views/home.html',
            controller: 'HomeCtrl'
        })
        .when('/dataCorrection', {
            templateUrl: 'views/dataCorrection.html',
            controller: 'DataCorrectionCtrl'
        })
        .when('/manager', {
            templateUrl: 'views/manager.html',
            controller: 'ManagerCtrl'
        })
        .when('/applicant', {
            templateUrl: 'views/applicant.html',
            controller: 'ApplicantCtrl'
        })
        .when('/appraiser', {
            templateUrl: 'views/appraiser.html',
            controller: 'AppraiserCtrl'
        })
        .when('/troubleshoot', {
            templateUrl: 'views/troubleshoot.html',
            controller: 'TroubleshootCtrl'
        })
        .when('/downPayment', {
            templateUrl: 'views/downPayment.html',
            controller: 'DownPaymentCtrl'
        })
        .when('/financialReview', {
            templateUrl: 'views/financialReview.html',
            controller: 'FinancialReviewCtrl'
        })
        .otherwise({
            redirectTo: '/'
        });
}]);

```

#### 4.3.1.3.3. Controllers

The AngularJS controllers mapped above, in this use case, largely consist of calls to the Java back-end for API manipulation and code regarding UI formatting and notifications. The Data Correction controller lists various prime examples of how various workflow actions related to work items, such as claiming, starting, stopping, releasing and completing a task, can be invoked from within the UI:

*dataCorrection.js:*

```
'use strict';

angular.module('app')
  .controller("DataCorrectionCtrl", ['$scope', '$http', 'Constants',
    '$location', '$timeout',
    function ($scope, $http, Constants, $location, $timeout) {

      $scope.consts = Constants;
      $scope.taskInProgress = false;

      fetchTasks();

      function fetchTasks() {

        $http.get("service/broker/dataCorrectionTasks").success(function (data)
        {
          $scope.tasks = data;
        });
      }

      $scope.claimTask = function (taskId) {
        $http.post("service/broker/claimTask", taskId).then(
          function (data) {
            $scope.successMessage = "Task claimed
Successfully";

            $scope.successVisible = true;
            fetchTasks();
            $timeout(function () {
              $scope.successVisible = false;
              $scope.successMessage = "";
            }, 2000);
          },
          function (data) {
            $scope.errorMessage = "Error occurred while
attempting to claim task";
            $scope.errorVisible = true;
            fetchTasks();
            console.log("ERROR: " + JSON.stringify({data:
data}));

            $timeout(function () {
              $scope.errorVisible = false;
              $scope.errorMessage = "";
            }, 2000);
          }
        );
      };

      ....

      $scope.submitApp = function() {

        var $application = {
```



```

        applicant : {
            name: $scope.applicantName,
            ssn: $scope.applicantSsn,
            income: $scope.applicantIncome,
            creditScore: null
        },
        property : {
            address: $scope.propertyAddress,
            price: $scope.propertyPrice
        },
        downPayment: $scope.downPayment,
        amortization: $scope.amortization,
        appraisal : null,
        mortgageAmount: ($scope.propertyPrice -
$scope.downPayment),
        apr: $scope.apr
    };

    $http.post("service/broker/completeTask/" +
$scope.taskInProgressId, $application).then(
    function(data) {
        $scope.successMessage = "Data Correction
submitted successfully";
        $scope.successVisible = true;
        $timeout(function() {
            $scope.successVisible = false;
            $scope.successMessage = "";
            $location.path("/home");
        }, 2000);
    },
    function (data) {
        $scope.errorMessage = "Error occurred while
attempting to submit Data Correction";
        $scope.errorVisible = true;
        console.log("ERROR: " + JSON.stringify({data:
data}));

        $timeout(function() {
            $scope.errorVisible = false;
            $scope.successMessage = "";
        }, 2000);
    });
});
});

```

Similar controllers are utilized for each of the various points in the business process where human interaction is required.

#### 4.3.1.3.4. Views

The *dataCorrection.html* file gives a great example of how data can be both displayed from scope, as responses to server queries are received, as well as how information can be collected and passed along for submission to other back-end controller points, thereby giving us the means to relay and receive information from the human task portions of the business process:

...

```

<tbody ng-repeat="task in tasks">
<tr>
<td>{{task["task-id"]}}</td>
<td>{{task["task-name"]}}</td>
<td>{{task["task-actual-owner"]}}</td>
<td>{{task["task-proc-inst-id"]}}</td>
<td>{{task["task-status"]}}</td>
<td>{{task["task-container-id"]}}</td>
<td>{{task["task-created-on"] | date:'MM-dd-yyyy HH:mm:ss Z'}}</td>
...
<form class="form-horizontal">
<div class="form-group">
<label class="col-sm-3 control-label" for="applicantName">Applicant Name</label>
<div class="col-sm-9">
<input ng-model="applicantName" type="text" id="applicantName" class="form-control">
</div>
</div>
<div class="form-group">
<label class="col-sm-3 control-label" for="applicantSsn">Social Security Number</label>
<div class="col-sm-9">
<input ng-model="applicantSsn" type="text" id="applicantSsn" class="form-control">
</div>
</div>
...
</form>
</div>
</div>

```

### 4.3.2. Java Application Components

#### 4.3.2.1. Maven Configuration and Dependencies

The high-level project 'parent' *pom.xml* file consists of basic project structural information, i.e. submodule listings:

```

<groupId>com.redhat.bpms.examples</groupId>
<artifactId>ocp-mortgage-example</artifactId>
<packaging>pom</packaging>
<version>1.0-SNAPSHOT</version>

<modules>
<module>ocp-mortgage-rules</module>
<module>ocp-mortgage-dashboard</module>
<module>ocp-credit-service</module>
</modules>

```

Each child module (*ocp-credit-service*, *ocp-mortgage-rules*, and *ocp-mortgage-dashboard*) has its own *pom.xml* file that is responsible for describing any dependencies and build plugin configuration relevant to the respective module. All three have been built according to each module's needs and worthy of inspection, but exploring the content of each is left as an exercise for the reader as brevity necessitates exclusion of these lengthy files.

#### 4.3.2.2. Logging Configuration

Logging, in this project example, is configured via *logback* and utilized via *slf4j*. The *logback.xml* file should be added to the */src/main/resources* directory, where it will be automatically detected by the framework.

*logback.xml*:

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

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
<layout class="ch.qos.logback.classic.PatternLayout">
<Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</Pattern>
</layout>
</appender>

<logger name="com.redhat.bpms" level="DEBUG"/>

<root level="INFO">
<appender-ref ref="STDOUT" />
</root>
</configuration>
```

Once configuration is in place, the logger can be instantiated and used inside any Java class as follows:

```
private static final Logger logger =
    LoggerFactory.getLogger(ThisClassName.class);
...
logger.error("ERROR in Rest endpoint releaseTask...", error);
```

#### 4.3.2.3. Data Model

Like the Mortgage Loan business process Knowledge Store, our module must be able to communicate via a common language in order to transfer information between the two. While it's possible to abstract the data model to its own submodule for easier sharing amongst dependent modules, in this use case the data model has been manually added to both the rules and user interface modules as they're unlikely to change during the lifetime of this example application and reference another repository or submodule complicates the ability to deploy the two modules independently in OCP. For this use case, the Java POJOs created automatically for the rules module by Business Central have simply been wholesale copied into a matching package structure within the dashboard module.

##### 4.3.2.3.1. Mapping for Work Item Consumption

Intelligent Process Server human task items are particular about the format that must be met for inbound data. This means that in order to supply information needed for entry points of these steps, we must adhere to the custom JSON format required, rather than simply trusting a mapping framework to generate a default JSON format and anticipating successful consumption. The following is an example mapping for the *Application* POJO from the data model which properly generated the necessary wrapper JSON elements and adherence to generics properties lists:

```
public final class ApplicationMapper {
...
    try {
        Map<String, Object> wrapper = new HashMap<>();
```

```

        Map<String, Object> appContent = new HashMap<>();
        Map<String, Object> applicant = new HashMap<>();
        Map<String, Object> property = new HashMap<>();

        applicant.put("name",
application.getApplicant().getName());
        applicant.put("ssn", application.getApplicant().getSsn());
        applicant.put("income",
application.getApplicant().getIncome());

        property.put("address",
application.getProperty().getAddress());
        property.put("price",
application.getProperty().getPrice());

        appContent.put("applicant", applicant);
        appContent.put("property", property);
        appContent.put("downPayment",
application.getDownPayment());
        appContent.put("amortization",
application.getAmortization());

        if (application.getAppraisal() != null) {
            Map<String, Object> appraisal = new HashMap<>();
            appraisal.put("property", property);
            appraisal.put("value",
application.getAppraisal().getValue());
            appContent.put("appraisal", appraisal);
        }

        wrapper.put("com.redhat.bpms.examples.mortgage.Application",
appContent);
        payload.put("application", wrapper);

    } catch (Exception e) {
        ...
    }

```

#### 4.3.2.4. Controllers

Java controllers are used in typical RESTful web service style to simply relay requests from front-end interactions to services that are responsible for the business logic layer relevant to each entity or human task type. In more advanced examples, this allows possible combining of multiple KIE calls and/or decouple application code from KIE configuration and implementation details. Weld dependency injection and *javax.javaee\_api* annotations allow for simple, straight-forward controller classes that are easy to both read and consume. The following class, *BrokerController*, is a good example of code that allows various actions correlating to the user interface to be performed:

```

@GET
@Path("/dataCorrectionTasks")
@Produces(MediaType.APPLICATION_JSON)
public List<TaskSummary> listTasks() {

    return brokerRestService.listTasks();
}

```

```

    }

    @POST
    @Path("/claimTask")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response.Status claimTask(Long taskId) {

        return brokerRestService.claimTask(taskId);
    }
    ...
    @POST
    @Path("/completeTask/{taskId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response.Status completeTask(@PathParam("taskId") Long
taskId, Application application) {

        return brokerRestService.completeTask(taskId, application);
    }
}

```

#### 4.3.2.5. Services

The Java service classes all extend from a base class *RestClientService* which houses a few calls across the multiple services, and also build on that functionality to define business logic and *kie-server* API calls relevant to each entity or human task type. *ApplicantRestService* is a small yet efficient example of how such calls to the Knowledge Store are performed:

```

@Singleton
public class ApplicantRestService extends RestClientService {

    private static final Logger logger =
LoggerFactory.getLogger(ApplicantRestService.class);

    public Response.Status startApp(Application application) {

        try {

            ProcessServicesClient processServicesClient =
initClient(Configuration.Users.KIESERVER)
                .getServicesClient(ProcessServicesClient.class);

            processServicesClient.startProcess(containerId,
PROCESS_ID, ApplicationMapper.convert(application));

        } catch (Exception e) {
            logger.error("ERROR in Rest endpoint startApp...", e);
        }

        return Response.Status.OK;
    }
}

```

Like with the *pom.xml* configurations, the code required for each of the business process steps is lengthy and spans multiple services, so exploring the content of each is left as an exercise for the reader.

#### 4.3.2.6. Configuration and Utility

Lastly, there are a few files which serve to aid in configuring Weld and abstracting some of the RESTful information needed to establish and utilize clients.

*Configuration.java*

```
public final class Configuration {

    public static final String REST_BASE_URI =
"http://rules.bxms.ose/kie-server/services/rest/server";

    public static enum Users {

        KIESERVER("kieserver", "kieserver1!"),
        BROKER("brokerUser", "password1!"),
        MANAGER("managerUser", "password1!"),
        APPRAISER("appraiserUser", "password1!");

        private final String username;
        private final String password;

        Users (String username, String password) {
            this.username = username;
            this.password = password;
        }

        public String username() {
            return this.username;
        }

        public String password() {
            return this.password;
        }
    }
}
```

*EndpointApplication.java*

```
@ApplicationPath("service")
public class EndpointApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        final Set<Class<?>> resources = new java.util.HashSet<>();
        addRestResourceClasses(resources);
        return resources;
    }

    private void addRestResourceClasses(final Set<Class<?>> resources)
    {
        resources.add(HomeController.class);
        resources.add(BrokerController.class);
        resources.add(ApplicantController.class);
        resources.add(TroubleshootController.class);
    }
}
```

```
resources.add(AppraiserController.class);  
resources.add(FinanceController.class);  
}  
}
```

## CHAPTER 5. BUSINESS PROCESS USAGE VIA USER INTERFACE

The *Mortgage Dashboard* will be used for both tracking a loan throughout the various human tasks as well as performing said tasks as each is required. The *Overview*, the default landing page, lists the rule process containers, process instances, and all outstanding human tasks. With this information, it's easy to deduce which tasks are needed to progress an application forward.

Figure 5.1. Dashboard Overview

Mortgage Dashboard

Overview

New Application

Data Correction

Troubleshoot

Down Payment Adjustment

Appraisal

Financial Review

Rules Processes Available

Container Id	Process Name	Process Id	Process Version
a0d69104d915487b21eade670d5b0810	MortgageApplication	com.redhat.bpm.s.examples.mortgage.MortgageApplication	1.0

Running Processes Instances

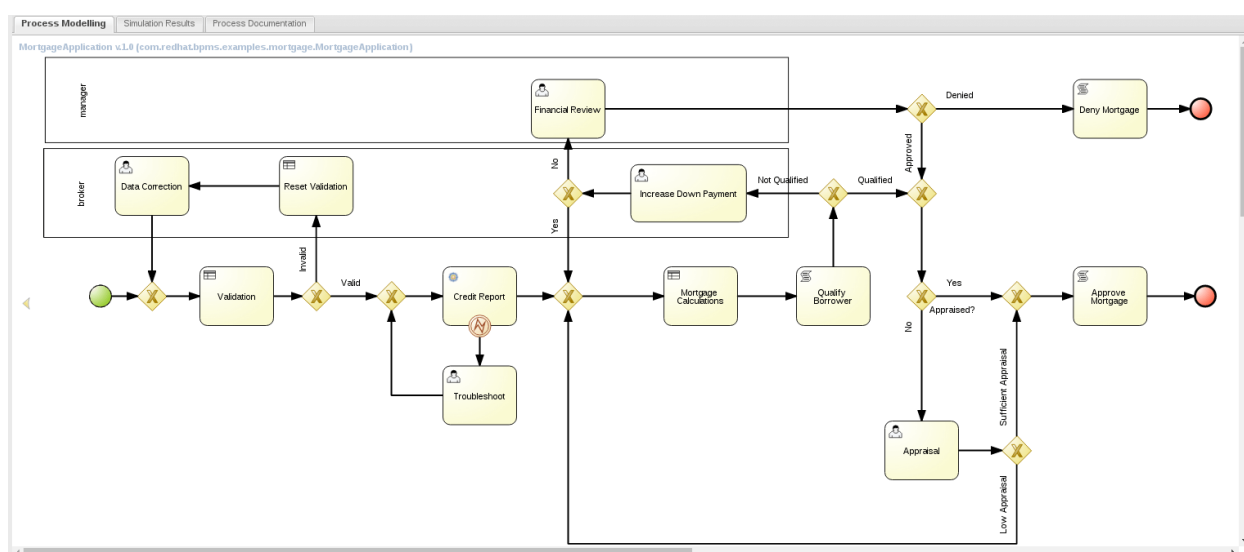
Process Instance Id	Process Name	Container Id	Process State	Process Start
5	MortgageApplication	a0d69104d915487b21eade670d5b0810	ACTIVE	12-09-2016 08:30:27 -0600
6	MortgageApplication	a0d69104d915487b21eade670d5b0810	ACTIVE	12-09-2016 08:30:42 -0600

Outstanding Work Items

Work Item Id	Work Item Name	Related Process Instance	Group Id	Item Category
14	Human Task	5	broker	Data Correction
16	Human Task	6	broker	Increase Down Payment

As a helpful reference to understanding the process flow and various steps involved, the business process can be visually represented as follows:

Figure 5.2. Process Overview



### 5.1. APPLICATION

To instantiate a new process instance, first we must submit a new application to the business process. Similarly to working with Business Central, this initial step is performed by submitting all



relevant data via a form for processing:

**Figure 5.3. Dashboard Application**

**Mortgage Dashboard**

Overview | New Application | Data Correction | Troubleshoot | Down Payment Adjustment | Appraisal | Financial Review

✓ Application submitted successfully

Applicant Name	Bob Smith
Social Security Number	313781234
Annual Income	150000
Property Address	100 Central Ave
Sales Price	350000
Down Payment	50000
Amortization	10
Annual Percentage Rate	4.5

**Submit Application**

## 5.2. DATA CORRECTION

Once an application has been submitted, the data is verified against a set of rules responsible for things like checking for a valid property price, down payment, social security number format, income, amortization, etc. If the validation fails, the application is moved into the *Data Correction* task, where the information can be reviewed, amended, and resubmitted to the process for continuation.

Initially, as is the case with each task category, the task itself is unassigned. While a group has already been specified by the business process, in this case the broker group, a specific user belonging to the group must claim and start the task in order to complete it. In more complex use cases, task delegation and oversight functionalities by supervisory or managerial roles are possible, but for the purposes of this application, only basic self-assignment and completion has been built in. The image below shows three Data Correction tasks, two which have already been claimed with one of those being in progress, thus disabling the manipulation of other tasks until the currently worked one is either stopped or completed:

**Figure 5.4. Data Correction Tasks**

Mortgage Dashboard

Overview
New Application
Data Correction
Troubleshoot
Down Payment Adjustment
Appraisal
Financial Review

Outstanding Data Correction Tasks

Task Id	Task Name	Task Owner	Related Process Instance	Task Status	Container Id	Task Start	Actions
10	Data Correction	brokerUser	5	Reserved	a0d69104d915487b21eade670d5b0810	12-09-2016 08:30:27 -0600	Release Start
14	Data Correction	brokerUser	9	InProgress	a0d69104d915487b21eade670d5b0810	12-09-2016 08:45:35 -0600	Stop
15	Data Correction		10	Ready	a0d69104d915487b21eade670d5b0810	12-09-2016 08:46:05 -0600	Claim

Applicant Name
Bob Smith

Social Security Number
313781234

Annual Income
150000

Property Address
100 Central Ave

Sales Price
350000

Down Payment
50000

Amortization
88

Annual Percentage Rate
4.5

Submit Correction

### 5.3. CREDIT SERVICE TROUBLESHOOTING

Once data correction has been performed, or if correction is not required for an application, the next step is to reach out to the Credit Service web service and get the information for the current applicant. This step is performed via a WebService task and, as far as the Dashboard is concerned, will largely go unobserved unless failures occur during the service call or response. Should errors occur, the relevant error information is relayed to the *Troubleshoot* tab of the dashboard to aid with diagnosing and solving any issues that arise:

Figure 5.5. Troubleshoot (Credit Service Errors)

Mortgage Dashboard				
Overview   New Application   Data Correction   Troubleshoot   Down Payment Adjustment   Appraisal   Financial Review				
Outstanding Troubleshoot Items				
Id	Name	Related Process Instance	Container Id	Error
		Instance		
		11	a0d69104d915487b21eade670d5b0810	org.apache.cxf.interceptor.Fault: Could not send Message. at org.apache.cxf.interceptor.MessageSenderInterceptor\$MessageSenderEndingInterceptor.handleMessage(MessageSenderInterceptor.java:6) at org.apache.cxf.phase.PhaseInterceptorChain.doIntercept(PhaseInterceptorChain.java:272) at org.apache.cxf.endpoint.ClientImpl.doInvoke(ClientImpl.java:572) at org.apache.cxf.endpoint.ClientImpl.invoke(ClientImpl.java:481) at org.apache.cxf.endpoint.ClientImpl.invoke(ClientImpl.java:382) at org.apache.cxf.endpoint.ClientImpl.invoke(ClientImpl.java:335) at org.apache.cxf.endpoint.ClientImpl.invoke(ClientImpl.java:355) at org.apache.cxf.endpoint.ClientImpl.invoke(ClientImpl.java:341) at org.jboss.bpm.process.workitem.webservice.WebServiceWorkItemHandler.executeWorkitem(WebServiceWorkItemHandler.jav

### 5.4. MORTGAGE CALCULATIONS AND BORROWER QUALIFICATION

Like the credit service task, the next few parts of the application process are seamless as far as the dashboard is concerned. First, a few calculations are performed to determine what offerings should be available for the applicant. The Mortgage Calculation step prices a mortgage by first calculating a minimum interest rate, based on only the length of the fixed-term mortgage (i.e., APR) and the applicant's credit score. This is followed by a look at the down payment ratio and the APR is adjusted upward if less than 20% is provided. Finally, jumbo mortgages are identified and result in yet another potential increase in the mortgage APR.

While the lending risk is already reflected in the calculated interest rate, there are also cases where the lender refuses to provide the applicant with a mortgage. One such case is the front-end ratio for the mortgage application exceeds the 28% threshold. Calculating the front-end ratio is a simple matter of determining the monthly mortgage payment (the process ignores other housing costs) and dividing it by the applicant's income. For simplicity, this calculation is performed in a script task. Following this, the process allows an application to either move on to the appraisal and/or approval phase, or go back to the broker for Down Payment Adjustment.

## 5.5. DOWN PAYMENT ADJUSTMENT

Once an applicant has been deemed unqualified given a specific scenario, it's still possible to avoid losing a business opportunity by exploring alternative ways to qualify the mortgage applicant. One simple solution is to request a larger down payment. This step in the process can be thought of as iterative. The broker may continually increase the down payment and re-submit the application for qualification in hopes that it may continue, or they may submit the same down payment twice in a row (i.e. no change in the most recent iteration) to signify that an upper limit has been reached and the application should process to financial review.

Figure 5.6. Down Payment Adjustment

**Mortgage Dashboard**

Overview | New Application | Data Correction | Troubleshoot | **Down Payment Adjustment** | Appraisal | Financial Review

**Outstanding Down Payment Adjustments**

Task Id	Task Name	Task Owner	Related Process Instance	Task Status	Container Id	Task Start	Actions
11	Increase Down Payment		6	Ready	a0d69104d915487b21eade670d5b0810	12-09-2016 08:30:42 -0600	<a href="#">Claim</a>
12	Increase Down Payment		7	Ready	a0d69104d915487b21eade670d5b0810	12-09-2016 08:36:44 -0600	<a href="#">Claim</a>
13	Increase Down Payment	brokerUser	8	InProgress	a0d69104d915487b21eade670d5b0810	12-09-2016 08:37:13 -0600	<a href="#">Stop</a>

Applicant Name:

Social Security Number:

Annual Income:

Property Address:

Sales Price:

Down Payment:

Amortization:

Annual Percentage Rate:

[Submit Adjustment](#)

## 5.6. FINANCIAL REVIEW

While the business process frequently solicits input from users, most logic decisions so far have been automated. It is common for most business processes to have some sort of manual override. In the loan application use case, a manager may have the authority to approve a mortgage application that does not meet the standard criteria. As a last resort before declining the application, a user task grants the manager group the ability to approve a mortgage previously flagged during the Down Payment Adjustment task and allow it to proceed to appraisal and approval. They may also choose to deny the mortgage, at which point signifies an end to the process, meaning that the process instance will no longer be visible on the dashboard or associated with any remaining tasks.

Figure 5.7. Financial Review

Mortgage Dashboard
Overview
New Application
Data Correction
Troubleshoot
Down Payment Adjustment
Appraisal
Financial Review

Outstanding Review Tasks

Task Id	Task Name	Task Owner	Related Process Instance	Task Status	Container Id	Task Start	Actions
18	Financial Review	managerUser	7	InProgress	a0d69104d915487b21eade670d5b0810	12-09-2016 09:27:43 -0600	<a href="#">Stop</a>

Sales Price
350000

Appraised Value

Down Payment
50000

Amortization
10

Annual Percentage Rate
4.5

Annual Income
150000

[Approve Loan](#)
[Deny Loan](#)

## 5.7. APPRAISAL

Once a borrower has been qualified for a loan, if an appraisal is not already present, the value must be collected prior to a final approval. This last step has a possibility of two outcomes.

The first possible outcome is that the appraisal could be sufficiently valued at over the sale price of the property, at which point the application gets final approval and completes the process. Should this occur, the process instance will no longer be visible on the dashboard or associated with any remaining tasks. In an extended use case, this application approval would likely trigger a separate business process that continues to another team or department for follow-up, but in this limited use case, the approval signifies the end.

The second possible outcome, wherein the appraisal is not sufficiently valued above the asking price of the property, the application will traverse back in the business process to the point of Mortgage Calculations and Borrower Qualification for recalculation and outcome pending the newly garnered information.

Figure 5.8. Appraisal

Mortgage Dashboard
Overview
New Application
Data Correction
Troubleshoot
Down Payment Adjustment
Appraisal
Financial Review

Outstanding Appraisal Tasks

Task Id	Task Name	Task Owner	Related Process Instance	Task Status	Container Id	Task Start	Actions
17	Appraisal	appraiserUser	8	InProgress	a0d69104d915487b21eade670d5b0810	12-09-2016 09:27:15 -0600	<a href="#">Stop</a>

Property Address
100 Central Ave

Sales Price
350000

Appraisal value

[Submit Appraisal](#)

## 5.8. FINALIZED APPLICATIONS

Once a final state of approval or denial is reached, the application process is complete. As alluded to previously, the process instance related to the finalized application will no longer be visible for tracking within the dashboard as all related work has been performed. As a final exercise for the reader, consider adding a summarization table to the Dashboard Overview which uses the available

API classes to track and list those process instances which are no longer in an active state and displays the outcome of the application process.

## CHAPTER 6. CONCLUSION

This reference environment discusses using BPM Suite Intelligent Process Server capabilities on OpenShift Container Platform. Through usage of various officially supported image streams and templates, the application built within demonstrates how to build upon and interface with a pre-existing rules projects, such as the one created in the [Business Process Management with Red Hat JBoss BPM Suite 6.3 Reference Architecture](#), and mimic the management of one or more running business processes via a custom User Interface in conjunction with provided REST and Java application APIs.

Various other technical considerations, as relevant to the developer role in creating and maintaining applications for the OpenShift Container Platform, are discussed, touching on disparate topics including image configuration, build and deployment, governance and monitoring, and interfacing with OCP projects, apps, deployments and pods.

## APPENDIX A. REVISION HISTORY

Revision	Release Date	Author(s)
1.0	January 2017	Jeremy Ary and Babak Mozaffari

## APPENDIX B. CONTRIBUTORS

We would like to thank the following individuals for their time and patience as we collaborated on this process. This document would not have been possible without their many contributions.

Contributor	Title	Contribution
Babak Mozaffari	Manager, Software Engineering and Consulting Engineer	Technical Content Review
David Ward	Senior Software Engineer, Cloud Enablement	Technical Content Review
Kris Verlaenen	Senior Principal Software Engineer, jBPM Project Lead	Technical Content Review
Prakasha Aradya	Senior Principal Product Manager	Content Review



# APPENDIX C. REVISION HISTORY

---

Revision 1.0-0	2017-01-12	JA
----------------	------------	----