# IBM ODM source of truth best practices What is DevOps?

As companies have evolved their Software Delivery Life Cycle to increase both speed and quality at which software gets delivered to final users, they’ve followed different methods and styles of production and delivery.

The first meaningful evolution came in the 90s, with Agile. It did a very good job at tearing down the wall between the traditional development and test teams. Yet, if the software produced is of higher quality, one can still commonly see the model by which it gets “thrown over the fence” to the operations team.

DevOps is a cultural shift, which aims at tearing down that other wall. Developers need understand how software is operated. IT teams need understand how software is built.

# Source Storage

One of the first questions that comes to mind when setting up a software factory is: where is the source code located? ODM offers two choices here.  
The first option is to handle rule artifacts the same way as code. That is, they are stored in a source code control system such as SVN, GIT, or whatever has been chosen in your team for the rest of the java code. They can thus be checked-in and checked-out through the Eclipse Team Collaboration feature in the Rule Designer environment.

If this approach works fine and is very natural to developers, it can find limitations when comes time to let several users collaborate. Moreover, it is achieved through the Eclipse tooling, which is an environment suitable for developers, but not something that you would want to expose your business users to.

Consequently, the second option is recommended. In this approach, you can leverage the features of the ODM Business Console to help business users organize and version rule artifacts. Indeed, the Business Console allows for versioning of artifacts, maintains integrity between them, guarantees traceability of the authoring actions and handles collaboration between rule writers. Moreover, it offers more sophisticated governance features through DGF (Decision Governance Framework) when your project grows to several parallel releases and a wider author team, and you want to dispatch and trace work.

In this approach, decision artifacts are then persisted to the Business Console database, and consequently it is important to back it up, like you would any other

database, as part of your policy of source code backup, since, should an outage occur, you would want to restore it for business continuity.

## Building ODM decision artifacts when they are stored in source code control

# Build automation

Rules and other ODM artifacts can be extracted out of source code control, either by using the Team Collaboration feature of the Eclipse environment in which Rule Designer runs, or by using dedicated tooling such as a Command Line Interface, depending on your preference and of the underlying Source Code Control tool chosen

## **Using the Build Command**

Since ODM 8.9, an alternative is to use the new Build Command feature (documented here <https://www.ibm.com/support/knowledgecenter/SSQP76_8.10.x/com.ibm.odm.dserver.rules.designer.run/build_topics/con_buildcmd_intro.html> ), which provides the same capability to generate a Rule Application out of a Decision Service, but with a lighter package. This feature is provided as a Maven plugin (a MOJO), so the setup of the build agents is greatly simplified, since each build agent will download the necessary plugin binary from the artifact repository. You can then direct decision artifact compilation by writing your directives in a standard maven pom file, using the following syntax:

<artifactId>loan-validation-service</artifactId>

<packaging>decisionservice</packaging>

<build>

  <plugins>

    <plugin>

      <groupId>com.ibm.rules.buildcommand</groupId>

        <artifactId>rules-compiler-maven-plugin</artifactId>

          <configuration>

            <deployments>

              <deployment>

                <name>test deployment</name>

                  </deployment>

              <deployment>

              <!-- If you have several deployment configurations,

                   make sure that they have different names. -->

                <name>production deployment</name>

              <!-- You must specify a classifier for each

                   additional deployment configuration. -->

                <classifier>prod</classifier>

              </deployment>

          </deployments>

        </configuration>

    </plugin>

  </plugins>

</build>

One of the strength of using Maven is that you can leverage the built-in dependency mechanism for retrieving the eXecution Object Model binary that the rule compiler requires to generate the Rule App, with the same standard declarative approach that Maven uses to manage dependencies.

**Note:** the Build Command is only available for building Decision Services, set to use the Decision Engine, and cannot handle Classic Rule Engine.

## Building ODM Decision Artifacts when they are stored in Decision Center:

In this case, the generation of a RuleApp is performed by the compiler built into Decision Center, and you don’t need to extract out source rules from Decision Center to compile them manually.

Instead you can use the IlrDeploymentFacility java API to specify the RuleApp to generate.

**Note:** you can also achieve the same extraction by using a series of **REST API** calls instead.

GET /v1/decisionservices

gives the list of the Decision Services

GET /v1/decisionservices/{decisionServiceId}/deployments

gives the list of Deployment Configurations for one of those Decision Service

GET /v1/deployments/{deploymentId}/download

gives the RuleApp matching the selected Deployment Configuration.

### Choosing Between the Source of Truth

Decision center as source of truth is recommended when:

1. There are no frequent XOM changes.
2. There are frequent backups to the DC database.
3. When business users have knowledge of Decision center testing and can perform deployment tasks independently.
4. When business users make changes to the decision artifacts.