



Installation Guide

/ ForgeRock Identity Management 7

Latest update: 7.0.4

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Copyright © 2011-2020 ForgeRock AS.

Abstract

Guide to installing, and uninstalling ForgeRock® Identity Management software. This software offers flexible services for automating management of the identity life cycle.



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.







Table of Contents

Overview	iv
1. Check Your Java Installation	1
2. Install and Run IDM	3
3. Interact with IDM	7
REST Interface Introduction	7
IDM User Interfaces	11
4. Install IDM as a Service	12
5. Start a New Project	18
6. Select a Repository	19
Embedded DS Repository	20
External DS Repository	22
MySQL Repository	27
Microsoft SQL Repository	30
Oracle DB Repository	35
PostgreSQL Repository	40
IBM DB2 Repository	43
7. Configure Your JDBC Repository	49
JDBC Database Access Rights	49
Configure Case Insensitivity for a JDBC Repo	49
Connect to a JDBC Repo Over SSL	50
8. Configuration and Monitoring	53
Specify the Startup Configuration	53
Monitor Server Health	55
Display Installed Modules and Features	58
9. Run IDM as a Cluster	61
Configure an IDM Instance as Part of a Cluster	63
How IDM Instances Read Configuration Changes	66
Scheduled Tasks Across a Cluster	68
Manage Nodes in a Cluster	69
A. Host and Port Information	73
B. Property Files	74
C. Embedded Jetty Configuration	76
IDM Configuration Properties in Jetty	76
Jetty Default Settings	78
Register Additional Servlet Filters	78
Enable and Disable Secure Protocols and Cipher Suites	80
Adjust Jetty Thread Settings	81
Gzip Compression for HTTP Responses	82
IDM Glossary	84

Overview

This guide shows you how to install ForgeRock Identity Management services for identity management, provisioning, and compliance. You do not need a complete understanding of ForgeRock Identity Management software to learn something from this guide, although a background in identity management and maintaining web application software can help. You do need some background in managing services on your operating systems and in your web application containers. Unless you are planning an evaluation or test installation, read the Release Notes before you get started.

Quick Start

 Install IDM Download the IDM software and get a minimal deployment up and running.	 Interact With IDM Introduction to the IDM REST API and browser-based user interface.	 Repository Configure IDM to use your selected production repository.
 Startup Configuration Learn about the startup configuration and how to verify system health.	 Install in a Cluster Install IDM in a cluster for availability.	 Jetty Configuration Configure the embedded Jetty server.

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see <https://www.forgerock.com>.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1

Check Your Java Installation

Before you start, follow these steps to ensure that your Java environment is suitable:

1. Verify that your computer has a supported Java version installed:

Supported Java Versions

Vendor	Versions
OpenJDK, including OpenJDK-based distributions: <ul style="list-style-type: none"> • AdoptOpenJDK/Eclipse Adoptium • Amazon Corretto • Azul Zulu • Red Hat OpenJDK ForgeRock tests most extensively with AdoptOpenJDK/Eclipse Adoptium.	11
Oracle Java	11

2. Read the pre-installation requirements.
3. Set the `JAVA_HOME` environment variable:

+ *Set `JAVA_HOME` on Windows*

- a. Locate the JRE installation directory (typically, `C:\Program Files\Java\`).
- b. Select Start > Control Panel > System and Security > System.
- c. Click Advanced System Settings.
- d. Click Environment Variables.
- e. Under System Variables, click New.
- f. Enter the Variable name (`JAVA_HOME`) and set the Variable value to the JRE installation directory; for example `C:\Program Files\Java\jre8`.

g. Click OK.

+ *Set JAVA_HOME on Linux*

- a. Open the user shell configuration file found in your home directory.
- b. Add the `JAVA_HOME` variable to the user shell configuration file, setting the value to `/usr`. In Bash, this would appear as `export JAVA_HOME="/usr"`.

Chapter 2

Install and Run IDM

Use the procedures in this section to install, start, run, and stop IDM.

+ *Install IDM*

Follow these steps to install IDM:

1. Make sure you have an appropriate version of Java installed:

```
java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.6+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.6+10, mixed mode)
```

For a description of the Java requirements, see "*Before You Install*" in the *Release Notes*.

2. Download IDM from the [ForgeRock BackStage download site](#). Releases on the ForgeRock BackStage download site are thoroughly validated for ForgeRock customers who run the software in production deployments, and for those who want to try or test a given release.
3. Unpack the contents of the .zip file into the install directory:

```
unzip ~/Downloads/IDM-7.0.4.zip
Archive:  IDM-7.0.4.zip
  inflating: openidm/.checksums.csv
   creating: openidm/bundle/
  extracting: openidm/bundle/openidm-audit-7.0.4.jar
...
```

4. By default, IDM listens for HTTP and HTTPS connections on ports 8080 and 8443, respectively. To change these port numbers, edit the following settings in your `resolver/boot.properties` file:

- `openidm.port.http`
- `openidm.port.https`

When you deploy IDM in production, you *must* set `openidm.host` to the URL of your deployment, in the same `resolver/boot.properties` file. Otherwise, calls to the `/admin` endpoint are not properly redirected.

Deployment URLs will vary, depending on whether you're using a load balancer. While IDM documentation does not specify how you'd configure a load balancer, you'll need to configure

IDM in a cluster as described in "Configure an IDM Instance as Part of a Cluster", and specifically in "Deploy Securely Behind a Load Balancer" in the *Security Guide*.

5. Before running IDM in production, replace the default embedded DS repository with a supported repository.

For more information, see "*Select a Repository*".

+ Start IDM

Follow these steps to run IDM interactively:

1. Start the Felix container, load all services, and start a command shell to allow you to manage the container:

Bash

```
/path/to/openidm/startup.sh
```

```
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

PowerShell

```
\path\to\openidm\startup.bat
```

```
"Using OPENIDM_HOME: \path\to\openidm"
"Using PROJECT_HOME: \path\to\openidm"
"Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dfile.encoding=UTF-8"
"Using LOGGING_CONFIG: -Djava.util.logging.config.file=\path\to\openidm\conf\logging.properties"
-> OpenIDM version "7.0.4"
OpenIDM ready
```

At the OSGi console `->` prompt, you can enter commands such as **help** for usage, or **ps** to view the bundles installed.

Startup errors and messages are logged to the console by default. You can also view these messages in the log files at </path/to/openidm/logs>.

2. Alternatively, you can manage the container and services from the Apache Felix Web Console.

Use these hints to connect to the Apache Felix Web Console:

- Default URL: <https://localhost:8443/system/console>
- Default user name: `admin`
- Default password: `admin`

Select Main > Components to see core services and their respective states.

+ Run IDM as a Background Process

You can also start IDM as a background process on UNIX and Linux systems. Follow these steps, preferably before you start IDM for the first time:

1. If you have already started the server, shut it down and remove the Felix cache files under `openidm/felix-cache/`:

```
shutdown
...
rm -rf felix-cache/*
```

2. Start the server in the background. The `nohup` survives a logout, and the `2>&1&` redirects standard output and standard error to the noted `console.out` file:

```
nohup ./startup.sh > logs/console.out 2>&1&
[1] 2343
```

To stop the server running as a background process, use the `shutdown.sh` script:

```
./shutdown.sh
Stopping OpenIDM (2343)
```

Note

Although installations on macOS systems are not supported in production, you might want to run IDM on macOS in a demo or test environment. To run IDM in the background on a macOS system, take the following additional steps:

- Remove the `org.apache.felix.shell.tui-*.jar` bundle from the `openidm/bundle` directory.
- Disable ConsoleHandler logging.

+ Stop IDM

You can stop IDM from the `->` prompt in the OSGi console, or through the Apache Felix Web Console. Both of these options stop the Felix container.

1. In the OSGi console, enter the `shutdown` command at the `->` prompt.
2. In the Apache Felix Web Console, select Web Console > System Information to stop the container.
3. On Unix systems, you can stop IDM by using the `shutdown.sh` script:

```
/path/to/openidm/shutdown.sh
Stopping OpenIDM (31391)
```

+ Uninstall IDM

1. Stop the server if it is running, as described in Stop IDM.
2. Remove the directory where you installed the software:

```
rm -rf /path/to/openidm
```

3. If you use a JDBC database for the repository, drop the `openidm` database.

+ Start in Debug Mode

To debug custom libraries, start the server with the Java Platform Debugger Architecture (JPDA):

1. Start IDM with the `jpda` option:

```
/path/to/openidm/startup.sh jpda
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Djava.compiler=NONE -Xnoagent -Xdebug
-Xrunjdwp:transport=dt_socket,address=5005,server=y,suspend=n
Using LOGGING_CONFIG:
-Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Listening for transport dt_socket at address: 5005
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.0.4" (revision: xxxx)
OpenIDM ready
```

The relevant JPDA options are listed in the startup script (`startup.sh`).

2. In your IDE, attach a Java debugger to the JVM via socket, on port 5005.

Caution

This interface is internal and subject to change. If you depend on this interface, contact ForgeRock support.

Chapter 3

Interact with IDM

There are two primary ways to interact with IDM; programmatically, using REST to access IDM's API endpoints, or using the browser-based user interfaces.

REST Interface Introduction

IDM provides RESTful access to users in its repository, and to its configuration. To access the repository over REST, you can use a browser-based REST client, such as the *Simple REST Client* for Chrome, or *RESTClient* for Firefox. You can also use applications such as *Postman* to create, run, and manage collections of REST calls. Alternatively you can use the **curl** command-line utility, included with most operating systems. For more information about **curl**, see <https://github.com/curl/curl>.

IDM is accessible over the regular and secure HTTP ports of the Jetty Servlet container, 8080, and 8443. Most of the command-line examples in this documentation set use the regular HTTP port, so that you don't have to use certificates just to test IDM. In a production deployment, install a CA-signed certificate and restrict REST access to a secure (HTTPS) port.

To run **curl** over the secure port, 8443, you must either include the **--insecure** option, or follow the instructions in "Restrict REST Access to the HTTPS Port" in the *Security Guide*. You can use those instructions with the self-signed certificate that is generated when IDM starts, or with a ***.crt** file provided by a certificate authority.

Note

Some of the examples in this documentation set use client-assigned IDs (such as **bjensen** and **scarter**) when creating objects because it makes the examples easier to read. If you create objects using the Admin UI, they are created with server-assigned IDs (such as **55ef0a75-f261-47e9-a72b-f5c61c32d339**). Generally, immutable server-assigned UUIDs are used in production environments.

+ Try Out IDM Using REST

1. Use the following REST query to list all users in the IDM repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/?_queryFilter=true&_fields=_id"
```

When you first install IDM with an empty repository, no users exist.

2. Create a user `joe` by sending a RESTful POST.

The following `curl` command creates a managed user in the repository, and set the user's ID to `jdoe`:

Bash

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "TestPassw0rd",
  "description": "My first user",
  "id": "joe"
}' \
http://localhost:8080/openidm/managed/user?_action=create
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

PowerShell

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "TestPassw0rd",
  "description": "My first user",
  "_id": "joe"
}'
http://localhost:8080/openidm/managed/user?_action=create
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

3. Fetch the newly created user from the repository with a RESTful GET:

Bash

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/managed/user/joe
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

PowerShell

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/managed/user/joe
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Format REST Output For Readability

By default, **curl**-based REST calls return the JSON object on one line, which can be difficult to read. For example:

```
{"mail":"joe@example.com","sn":"smith","passwordAttempts":"0",
"lastPasswordAttempt":"Mon Apr 14 2014 11:13:37 GMT-0800 (GMT-08:00)",
"givenName":"joe","effectiveRoles":["internal/role/openidm-authorized"],
"password":{"$crypto":{"type":"x-simple-encryption","value":{"data":
"0BFVL9cG8uaLoo1N+SMJ3g==","cipher":"AES/CBC/PKCS5Padding","iv":
"7r1V4EwkwdRHkt19F8g22A==","key":"openidm-sym-default"}}},"country":"","
"city":"","_rev":"00000000c03fd7aa","lastPasswordSet":"","postalCode":"","
"_id":"joe3","description":"My first user","accountStatus":"active","telephoneNumber":
"555-123-1234","roles":["internal/role/openidm-authorized"],"effectiveAssignments":{"},
"postalAddress":"","stateProvince":"","userName":"joe3"}
```

At least two options are available to clean up this output:

+ *Format Output Using a JSON Parser*

The standard way to format JSON output is with a JSON parser such as **jq**. **jq** is not installed by default on most operating systems, but you can install it and then "pipe" the output of a REST call to **jq**, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/joe" \
| jq .
```

+ Format Output Using the REST API

The ForgeRock REST API includes an optional `_prettyPrint` request parameter. The default value is `false`. To use the ForgeRock REST API to format output, add a parameter such as `?_prettyPrint=true` or `&_prettyPrint=true`, depending on whether it is added to the end of an existing request parameter. In this case, the following command would return formatted output:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/joe?_prettyPrint=true"
```

Note

Most command-line examples in this guide do not show this parameter, but the output is formatted for readability.

IDM User Interfaces

IDM provides UIs at two different endpoints; `/` and `/admin`. The administrative tools available at `/admin` are called the Admin UI. The End User UI enables end users to manage certain aspects of their own accounts.

For information about the Admin UI, see "*Admin UI*" in the *Setup Guide* .

For information about the End User UI, see "*Self-Service End User UI*" in the *Self-Service Reference* .

Chapter 4

Install IDM as a Service

These sections describe how to install and run IDM as a service, on Windows and Linux systems:

+ *Install as a Windows Service*

You can install IDM to run as a Windows service, so that it automatically starts and stops with Windows. You must be logged in as an administrator to install a Windows service.

Note

On a 64-bit Windows server, you must have a 64-bit Java version installed to start the service. If a 32-bit Java version is installed, you will be able to install IDM as a service, but starting the service will fail.

Before you launch the `service.bat` file, which registers the service within the Windows registry, make sure that your `JAVA_HOME` environment variable points to a valid 64-bit version of the JRE or JDK. If you have already installed the service with the `JAVA_HOME` environment variable pointing to a 32-bit JRE or JDK, delete the service first, then reinstall the service.

1. Unpack the IDM-7.0.4.zip file, as described previously, and navigate to the `install-directory\bin` directory:

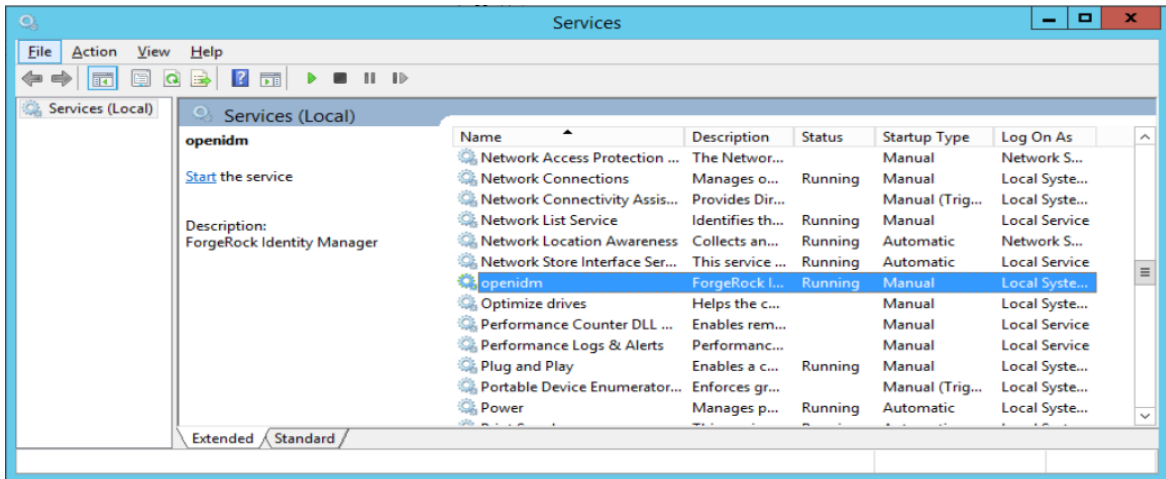
```
C:\>cd openidm\bin
C:\openidm\bin>
```

2. Run the `service.bat` command with the `/install` option, specifying the name that the service should run as:

```
C:\openidm\bin>service.bat /install openidm
ForgeRock Identity Management Server successfully installed as "openidm" service
```

3. Use the Windows Service manager to manage the IDM service.

Running as a Windows Service



- By default, the IDM service is run by **Local System**, which is a system-level service account built in to Windows. Before you deploy IDM in production, you should switch to an account with fewer permissions. The account running the IDM service must be able to read, write, and execute only the directories related to IDM.
- Use the Windows Service Manager to start, stop, or restart the service.
- If you want to uninstall the IDM service, first use the Windows Service Manager to stop IDM and then run the following command:

```
C:\install-directory\openidm\bin>service.bat /uninstall openidm
Service "openidm" removed successfully
```

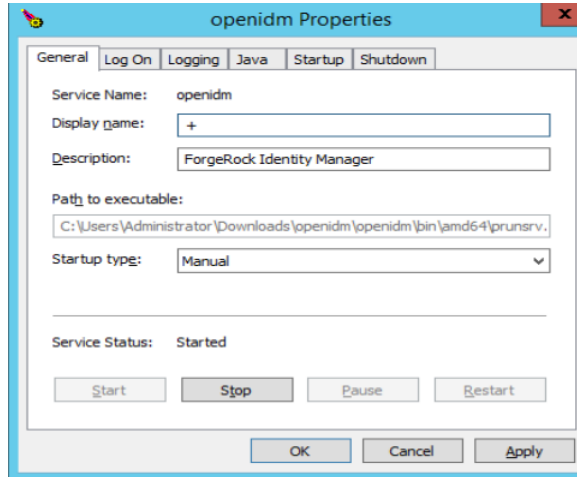
- If desired, you can then set up IDM with a specific project directory:

```
C:\install-directory\openidm\bin>service.bat /install openidm -p C:\project-directory
ForgeRock Identity Management Server successfully installed as "openidm" service
```

You can also manage configuration details with the Procrun monitor application. IDM includes the associated **prunmgr.exe** executable in the **C:\install-directory\openidm\bin** directory.

For example, you can open the Windows service configuration application for IDM with the following command, where **ES** stands for *Edit Service Configuration*

```
C:\install-directory\openidm\bin>prunmgr.exe //ES/openidm
```



The `prunmgr.exe` executable also includes the monitor application functionality described in the following Apache Commons page on the: *Procrun monitor Application*. However, IDM does not include the Procrun service application.

For example, if you've configured IDM as a Windows service, you can start and stop it with the following commands:

```
C:\install-directory\openidm\bin>prunmgr.exe //MR/openidm
C:\install-directory\openidm\bin>prunmgr.exe //MQ/openidm
```

In these commands, **MR** is the option to *Monitor and Run* IDM, and **MQ** stands for *Monitor Quit*, which stops the IDM service.

+ Install as a Linux Service

IDM provides a script that can generate **SysV** or **Systemd** service initialization scripts. You can start the script as the root user, or configure it to start during the boot process.

When IDM runs as a service, logs are written to the installation directory.

1. If you have not yet installed IDM, follow the steps in [Install IDM](#).
2. Review the options by running the following script:

```
/path/to/openidm/bin/create-openidm-rc.sh
Usage: ./create-openidm-rc.sh --[systemd|chkconfig|lsb]
Outputs OpenIDM init file to stdout for the given system

--systemd    Generate Systemd init script. This is preferred for all modern distros.
--chkconfig  Generate SysV init script with chkconfig headers (RedHat/CentOS)
--lsb        Generate SysV init script with LSB headers (Debian/Ubuntu)
...

```

These examples describe how to create each of these scripts:

+ Set up a Systemd Service

If you're running relatively standard versions of Red Hat Enterprise Linux (CentOS Linux) version 7.x, or Ubuntu 16.04 and later, you'll want to set up a systemd service script. To set up such a script, navigate to the `/path/to/openidm/bin` directory, and run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --systemd
```

As noted in the output, you can set up the IDM service on a standard systemd-based Linux distribution with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --systemd > openidm.service  
sudo cp openidm.service /etc/systemd/system/  
systemctl enable openidm  
systemctl start openidm
```

To stop the IDM service, run the following command:

```
systemctl stop openidm
```

You can modify the `openidm.service` script. The following excerpt would run IDM with a startup script in the `/home/idm/project` directory:

```
[Unit]  
Description=ForgeRock OpenIDM  
After=network.target auditd.target  
  
[Service]  
Type=simple  
SuccessExitStatus=143  
Environment=JAVA_HOME=/usr  
User=testuser  
ExecStart=/root/openidm/startup.sh -p /home/idm/project  
ExecStop=/root/openidm/shutdown.sh  
  
[Install]  
WantedBy=multi-user.target
```

Run the following command to reload the configuration and then start the IDM service script:

```
systemctl daemon-reload  
systemctl start openidm
```

+ Set up a SysV Service (Red Hat)

If you are running standard versions of Red Hat Enterprise Linux (CentOS Linux) version 6.x, set up a SysV service script with runlevels controlled through the `chkconfig` command. To set up such a script, run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --chkconfig
```

You can then set up and start the IDM service on a Linux distribution that uses SysV init scripts, with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --chkconfig > openidm
sudo cp openidm /etc/init.d/
sudo chmod u+x /etc/init.d/openidm
sudo chkconfig --add openidm
sudo chkconfig openidm on
sudo service openidm start
```

To stop the IDM service, run the following command:

```
sudo service openidm stop
```

You can modify the `/etc/init.d/openidm` script. The following excerpt would run IDM with the `startup.sh` script in the `/path/to/openidm` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh >$OPENIDM_HOME/logs/server.out
2>&1 &"
```

You can modify this line to point to some `/path/to/production` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh -p /path/to/production >
$OPENIDM_HOME/logs/server.out 2>&1 &"
```

Run the following command to reload the configuration and then start the IDM service script:

```
sudo service openidm start
```

If you run Linux with SELinux enabled, change the file context of the newly copied script with the following command:

```
sudo restorecon /etc/init.d/openidm
```

Verify the change to SELinux contexts with the `ls -Z /etc/init.d` command. For consistency, change the user context to match other scripts in the same directory with the `sudo chcon -u system_u /etc/init.d/openidm` command.

+ Set up a SysV Service (Ubuntu)

If you're running an older version of Ubuntu Linux that supports SysV services, set up a SysV service script, with runlevels controlled through the `update-rc.d` command. To set up such a script, run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --lsb
```

You can then set up and start the IDM service on a Linux distribution that uses SysV init scripts, with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --lsb > openidm
sudo cp openidm /etc/init.d/
sudo chmod u+x /etc/init.d/openidm
sudo update-rc.d openidm defaults
sudo service openidm start
```

To stop the IDM service, run the following command:

```
sudo service openidm stop
```

You can modify the `/etc/init.d/openidm` script. The following excerpt would run IDM with the `startup.sh` script in the `/path/to/openidm` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh >$OPENIDM_HOME/logs/server.out  
2>&1 &"
```

You can modify this line to point to some `/path/to/production` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh -p /path/to/production >  
$OPENIDM_HOME/logs/server.out 2>&1 &"
```

You can then run the following command to reload the configuration and then start the IDM service script:

```
sudo service openidm restart
```

Chapter 5

Start a New Project

When you extract the IDM .zip file, you have a default *project* under `/path/to/openidm`.

Important

You can use this project to test customizations, but you should not run the default project in production.

Set up a new project as follows:

1. Create a directory for your new project:

```
mkdir /path/to/my-project
```

2. Set up a minimal configuration:

- If your project will be similar to any of the sample configurations, copy the contents of the sample to your new project.

For example:

```
cp -r /path/to/openidm/samples/sync-with-ldap/* /path/to/my-project/
```

You can then customize the sample configuration according to your requirements.

- If you do not want to start with one of the sample configurations, copy the `conf/` and `script/` directories from the default project to your new project directory:

```
cp -pr /path/to/openidm/conf /path/to/my-project/  
cp -pr /path/to/openidm/script /path/to/my-project/
```

You can then customize the default configuration according to your requirements.

3. Start your new project as follows:

```
/path/to/openidm/startup.sh -p /path/to/my-project
```

Chapter 6

Select a Repository

By default, IDM uses an embedded ForgeRock Directory Services (DS) instance for its internal repository. This means that you don't need to install a database to evaluate the software. Before you use IDM in production, you must replace the embedded DS repository with a supported repository. For supported versions, see Supported Repositories in the *Release Notes*:

- External DS instance

Important

Both the default embedded and the external DS repositories do *not* support storage of audit or workflow data. Audit logging to the repository is disabled by default. Do not enable logging to the repository if you are using a DS repository.

- MySQL
- MariaDB

The "MySQL Repository" instructions are also applicable to MariaDB.

- Microsoft SQL
- PostgreSQL
- Oracle Database (Oracle DB)
- IBM DB2 Database

You must also decide how IDM should map objects to the tables in a JDBC database or to organizational units in DS:

- *Generic mapping*, which allows you to store arbitrary objects without special configuration or administration.
- *Explicit mapping*, which maps specific objects and properties to tables and columns in the JDBC database or to organizational units in DS.

By default, IDM uses a generic mapping for user-definable objects, for both a JDBC and a DS repository. A generic mapping speeds up initial deployment, and can make system maintenance more flexible by providing a stable database structure. In a test environment, generic tables enable you to modify the user and object model easily, without database access, and without the need to

constantly add and drop table columns. However, generic mapping does not take full advantage of the underlying database facilities, such as validation within the database and flexible indexing. Using an explicit mapping generally results in a *substantial* performance improvement. It is therefore strongly advised that you change to an explicit mapping before deploying in a production environment. If you are integrating IDM with AM, and using a shared DS repository, you *must* use an explicit schema mapping.

IDM provides a sample configuration, for each JDBC repository, that sets up an explicit mapping for the managed *user* object and a generic mapping for all other managed objects. This configuration is defined in the files named `/path/to/openidm/db/repository/conf/repo.jdbc-repository-explicit-managed-user.json`. To use this configuration, copy the file that corresponds to your repository to your project's `conf/` directory and rename it `repo.jdbc.json`. Run the `sample-explicit-managed-user.sql` data definition script (in the `path/to/openidm/db/repository/scripts` directory) to set up the corresponding tables when you configure your JDBC repository.

This chapter describes how to set up IDM to work with each of the supported repositories, and lists the minimum rights required for database installation and operation.

For information about the repository configuration, see "*Store Managed Objects in the Repository*" in the *Object Modeling Guide*. For more information about generic and explicit mappings, see "Generic and Explicit Object Mappings" in the *Object Modeling Guide*.

Embedded DS Repository

By default, IDM uses the `conf/repo.ds.json` file to start an embedded DS instance. The embedded DS repository is not supported in production environments.

The embedded DS server uses the embedded DS keystore, and has the following configuration by default:

- `hostname` - `localhost`
- `ldapPort` - `31389`
- `ldapsPort` - `31636`
- `bindDN` - `uid=admin`
- `bindPassword` - `5up3r53cr3t`
- `adminPort` - `34444`

You can query the embedded repository directly by using the LDAP command-line utilities provided with DS:

+ *Query the Embedded DS Repository*

This command returns all the objects in the repository of a default IDM project:


```
/path/to/openssl/bin/ldapsearch \  
--hostname localhost \  
--port 31636 \  
--bindDN uid=admin \  
--bindPassword 5up3r53cr3t \  
--baseDN "dc=openidm,dc=forgerock,dc=com" \  
--useSSL \  
--trustAll \  
"(objectclass=*)"\  
dn: dc=openidm,dc=forgerock,dc=com\  
objectClass: domain\  
objectClass: top\  
dc: openidm\  
  
dn: ou=links,dc=openidm,dc=forgerock,dc=com\  
objectClass: organizationalUnit\  
objectClass: top\  
ou: links\  
  
dn: ou=internal,dc=openidm,dc=forgerock,dc=com\  
objectClass: organizationalUnit\  
objectClass: top\  
ou: internal\  
  
dn: ou=users,ou=internal,dc=openidm,dc=forgerock,dc=com\  
objectClass: organizationalUnit\  
objectClass: top\  
ou: users\  
...
```

For more information about the DS command-line utilities, see the [DS Tools Reference](#).

To change the administrative port of the embedded DS server, add an `adminPort` property to your project's `conf/repo.ds.json` file before you start IDM. To change any of the other default values, add an `LdapConnectionFactory` property, as shown in the following example.

This excerpt of a `repo.ds.json` sets the administrative port to `4444`. The example changes the bind password to `MyPassw0rd` but shows the structure of the entire `LdapConnectionFactory` property for reference:

```
{  
  "embedded": true,  
  "maxConnectionAttempts": 5,  
  "adminPort": 4444,  
  "LdapConnectionFactory": {  
    "bind": {  
      "primaryLdapServers": [{ "hostname": "localhost", "port": 31389 }]  
    },  
    "root": {  
      "authentication": {  
        "simple": { "bindDn": "uid=admin", "bindPassword": "MyPassw0rd" }  
      }  
    }  
  },  
  ...  
}
```

It is not necessary to add the entire `ldapConnectionFactory` block to your configuration file but you must respect the JSON structure. For example, to change only the `hostname`, you would need to add at least the following:

```
{
  ...
  "ldapConnectionFactory": {
    "bind": {
      "primaryLdapServers": [{ "hostname": "my-hostname" }]
    }
  },
  ...
}
```

If you don't include an `ldapConnectionFactory` object, IDM installs an embedded DS server with the default configuration.

External DS Repository

IDM supports the following deployment scenarios with a DS repository:

- Single external DS instance
- Two DS instances in an active/passive configuration

IDM supports two replicated DS instances for backup/availability purposes only. Using multiple replicated DS instances as repositories (in a multimaster DS deployment) is not supported.

Configure a Single External DS Instance as a Repository

1. If you have not yet installed DS, download it from the ForgeRock BackStage download site and extract the .zip archive.
2. Install DS according to the instructions in the DS Installation Guide:
 - a. If you are planning to use a generic object mapping in the *Object Modeling Guide* for managed users, install DS with the `idm-repo` profile (see Install DS as an IDM Repository).
 - b. If you are planning to use an explicit object mapping in the *Object Modeling Guide* for managed users, install DS with both the `idm-repo` and `am-identity-store` profiles (see Install DS as an IDM Repository and Install DS for AM Identities).

This example configures DS on the localhost, listening on the following ports:

- LDAP port: `31389`
- Admin port: `34444`
- LDAPS port: `31636`

We've used these ports to avoid a port conflict with the default ports used in the LDAP samples. You can use any host and available ports in the setup. If you use a different host and ports, change the `primaryLdapServers` property in your `repo.ds-external.json` file accordingly.

Note

Every DS deployment requires a *deployment key* and a *deployment key password* to secure network connections. The deployment key is a random string generated by DS software. The deployment key password is a secret string that you choose. It must be at least 8 characters long. The deployment key and password automate key pair generation and signing without storing the CA private key. For more information, see Deployment Keys in the *DS Security Guide*.

3. In your IDM installation, remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/  
rm repo.ds.json
```

4. Copy the external DS repository configuration file (`repo.ds-external.json`) to your project's `conf` directory and rename it `repo.ds.json`:

```
cp /path/to/openidm/db/ds/conf/repo.ds-external.json my-project/conf/repo.ds.json
```

5. Enable IDM to trust the DS server certificate for your deployment.

For example, in the default case, where DS servers use TLS key pairs generated using a deployment key and password, import the deployment key-based CA certificate into the IDM truststore:

```
/path/to/opensj/bin/dskeymgr \  
export-ca-cert \  
--deploymentKey your-deployment-key \  
--deploymentKeyPassword password \  
--outputFile ds-ca-cert.pem  
  
keytool \  
-importcert \  
-alias ds-ca-cert \  
-keystore /path/to/openidm/security/truststore \  
-storepass:file /path/to/openidm/security/storepass \  
-file ds-ca-cert.pem  
Owner: CN=Deployment key, O=ForgeRock.com  
Issuer: CN=Deployment key, O=ForgeRock.com  
...  
Trust this certificate? [no]: yes  
Certificate was added to keystore
```

6. Adjust the connection settings from IDM to DS in the IDM repository configuration file, `repo.ds.json`:
 - If your DS instance is *not* running on the localhost and listening for LDAP connections on port `31389`, adjust the `primaryLdapServers` property in that file to match your DS setup.

- Make sure the password for the DS directory superuser (`uid=admin`) matches the DS root user password in the IDM configuration.

For details about the connection settings, see the information in *Gateway LDAP Connections* in the *DS HTTP User Guide*. (IDM shares these configuration settings with the DS REST to LDAP Gateway.)

7. Start IDM with the configuration for your project. For example:

```
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/my-project
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/my-project/conf/logging.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

8. (Optional) Verify that IDM successfully connects to DS:

```
grep 31389 /path/to/opensj/logs/ldap-access.audit.json | tail -n 1 | jq .
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "127.0.0.1",
    "port": 35874
  },
  "server": {
    "ip": "127.0.0.1",
    "port": 31389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "SEARCH",
    "connId": 1,
    "msgId": 232,
    "dn": "ou=triggers,ou=scheduler,dc=openidm,dc=forgerock,dc=com",
    "scope": "one",
    "filter": "(&(&(fr-idm-json:caseIgnoreJsonQueryMatch:=/state eq \"NORMAL\")(!(fr-idm-
json:caseIgnoreJsonQueryMatch:=/nodeId pr)))(objectClass=uidObject)(objectClass=fr-idm-generic-obj
(objectClass=top))",
    "attrs": [
      "objectClass",
      "uid",
      "etag",
      "createTimestamp",
      "modifyTimestamp",
      "fr-idm-json"
    ]
  },
  "transactionId": "transaction-id",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 1,
    "elapsedTimeUnits": "MILLISECONDS",
    "nentries": 0
  }
}
```

```
},  
"timestamp": "timestamp",  
"_id": "id"  
}
```

Configure Two DS Repositories in an Active/Passive Deployment

With this configuration, IDM fails over to the secondary DS instance if the primary instance becomes unavailable. When the primary DS instance is restarted, that instance again becomes the target of all requests.

1. Download DS from the ForgeRock BackStage download site and extract the .zip archive.
2. Install two DS servers, according to the instructions in the DS Installation Guide.

Important

When you set up each server, specify a `replicationPort` and `bootstrapReplicationServer` so that both servers are installed as replicas. For information on these setup options, see setup in the *DS Tools Reference*.

- a. If you are planning to use a generic object mapping in the *Object Modeling Guide* for managed users, install DS with the `idm-repo` profile (see Install DS as an IDM Repository).
- b. If you are planning to use an explicit object mapping in the *Object Modeling Guide* for managed users, install DS with both the `idm-repo` and `am-identity-store` profiles (see Install DS as an IDM Repository and Install DS for AM Identities).

Note

Every DS deployment requires a *deployment key* and a *deployment key password* to secure network connections. The deployment key is a random string generated by DS software. The deployment key password is a secret string that you choose. It must be at least 8 characters long. The deployment key and password automate key pair generation and signing without storing the CA private key. For more information, see Deployment Keys in the *DS Security Guide*.

3. In your IDM installation, remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/  
rm repo.ds.json
```

4. Copy the external DS repository configuration file (`repo.ds-external.json`) to your project's `conf` directory and rename it `repo.ds.json`:

```
cp /path/to/openidm/db/ds/conf/repo.ds-external.json my-project/conf/repo.ds.json
```

5. Enable IDM to trust each DS server certificate for your deployment.

For example, in the default case, where DS servers use TLS key pairs generated using a deployment key and password, import the deployment key-based CA certificate *for each server* into the IDM truststore.

You will need to give the CA certificate of the second server a different alias.

```
/path/to/openssl/bin/dskeymgr \
export-ca-cert \
--deploymentKey your-deployment-key \
--deploymentKeyPassword password \
--outputFile ds-ca-cert.pem
keytool \
-importcert \
-alias ds-ca-cert \
-keystore /path/to/openidm/security/truststore \
-storepass:file /path/to/openidm/security/storepass \
-file ds-ca-cert.pem
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

- Specify the connection settings from IDM to the two DS servers in the `ldapConnectionFactories` property of the repository configuration file (`repo.ds.json`).

This example assumes that the first DS server runs on the host `ds1.example.com`, and the second DS server runs on the host `ds2.example.com`:

```
"ldapConnectionFactories": {
  "bind": {
    "connectionSecurity": "startTLS",
    "heartBeatIntervalSeconds": 60,
    "heartBeatTimeoutMilliSeconds": 10000,
    "primaryLdapServers": [{ "hostname": "ds1.example.com", "port": 31389 }],
    "secondaryLdapServers": [{ "hostname": "ds2.example.com", "port": 31389 }]}
}
```

Adjust the settings to match your DS server setup.

For details about the connection settings, see the information in *Gateway LDAP Connections* in the *DS HTTP User Guide*. (IDM shares these configuration settings with the DS REST to LDAP Gateway.)

- Also in the `repo.ds.json` file, check the `authentication` settings:

```
"root": {
  "inheritFrom": "bind",
  "authentication": {
    "simple": { "bindDn": "uid=admin", "bindPassword": "5up3r53cr3t" }
  }
}
```

Make sure that the `bindDn` and `bindPassword` match the bind details of the DS superuser.

8. Start IDM and verify that the connection to the primary DS server is successful.
9. (Optional) Shut down the primary DS server and verify that the failover to the secondary server occurs, as expected.

MySQL Repository

This procedure assumes that you have installed MySQL on the local host. Follow the [MySQL documentation](#) that corresponds to your MySQL version. For supported MySQL versions, see [Supported Repositories](#) in the *Release Notes*.

Configure IDM to use the new repository *before you start IDM for the first time*. This procedure assumes that a password has already been set for the MySQL root user:

1. Download [MySQL Connector/J](#) version 5.1 or later.
2. Unpack the downloaded file, and copy the .jar file to `openidm/bundle/`:

```
cp mysql-connector-java-version.jar /path/to/openidm/bundle/
```

3. Make sure that IDM is stopped:

```
/path/to/openidm/shutdown.sh  
OpenIDM is not running, not stopping.
```

4. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory:

```
rm my-project/conf/repo.ds.json
```

5. Copy the MySQL database connection configuration file (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's `conf` directory:

```
cp /path/to/openidm/db/mysql/conf/datasource.jdbc-default.json my-project/conf/  
cp /path/to/openidm/db/mysql/conf/repo.jdbc.json my-project/conf/
```

6. If you have previously set up a MySQL repository for IDM, you *must* drop the `openidm` database and users before you continue:

```
mysql> drop database openidm;  
Query OK, 21 rows affected (0.63 sec)  
mysql> drop user openidm;  
Query OK, 0 rows affected (0.02 sec)  
mysql> drop user openidm@localhost;  
Query OK, 0 rows affected (0.00 sec)
```

7. Import the IDM database and tables:

```
cd /path/to/mysql  
mysql -u root -p < /path/to/openidm/db/mysql/scripts/openidm.sql  
Enter password:
```

Note

If you see errors like `Access denied for user 'root'@'localhost'`, and are deploying on a new installation of Ubuntu 16.04 or later, the `UNIX_SOCKET` plugin may be installed, which applies Linux `root` credentials to MySQL. In that case, substitute `sudo mysql -u root` for `mysql -u root -p` in the commands in this section.

8. Create the IDM database user.

+ *For MySQL 5.7 or higher*

Run the following script:

```
cd /path/to/mysql
mysql -u root -p < /path/to/openidm/db/mysql/scripts/createuser.sql
Enter password:
```

+ *For MySQL prior to 5.7*

Run the following script:

```
cd /path/to/mysql
mysql -u root -p < /path/to/openidm/db/mysql/scripts/createuser.mysql56.sql
Enter password:
```

9. Run the script that creates the tables required by the workflow engine:

```
cd /path/to/mysql
mysql -D openidm -u root -p < /path/to/openidm/db/mysql/scripts/flowable.mysql.all.create.sql
Enter password:
```

Warning

You must use MySQL version 5.6.4 or later. If you are using an older version, perform the MySQL upgrade before upgrading to IDM 7 or later. For additional information, see the *Flowable Note for MySQL users*.

10. If you are planning to direct audit logs to this repository, run the script that sets up the audit tables:

```
mysql -D openidm -u root -p < /path/to/openidm/db/mysql/scripts/audit.sql
Enter password:
```

11. Update the connection configuration to reflect your MySQL deployment. The default connection configuration (in the `conf/datasource.jdbc-default.json` file) is:


```
{
  "driverClass" : "com.mysql.jdbc.Driver",
  "jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}&{openidm.repo.port}/openidm?
allowMultiQueries=true&characterEncoding=utf8&serverTimezone=UTC",
  "databaseName" : "openidm",
  "username" : "openidm",
  "password" : "openidm",
  "connectionTimeout" : 30000,
  "connectionPool" : {
    "type" : "hikari",
    "minimumIdle" : 20,
    "maximumPoolSize" : 50
  }
}
```

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

+ Set in an IDM Properties File

Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file, for example:

```
openidm.repo.host=localhost
openidm.repo.port=3306
```

These lines are commented out by default in `resolver/boot.properties`.

The default MySQL port is `3306`. You can use the `netstat -tlnp` command to check which port your MySQL instance is running on.

+ Set as an Environment Variable

Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=3306"
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=3306
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

Tip

In a production environment, set up SSL as described in the MySQL Connector Developer Guide. If you are using an older version of MySQL, add `&useSSL=true` to the end of the `jdbcURL`.

If you are running a relatively recent version of MySQL (5.5.45+, 5.6.26+, 5.7.6+), the default configuration expects you to set up SSL, unless you add `&useSSL=false` to the end of the `jdbcUrl`.

When you have set up MySQL for use as the internal repository, make sure that the server starts without errors.

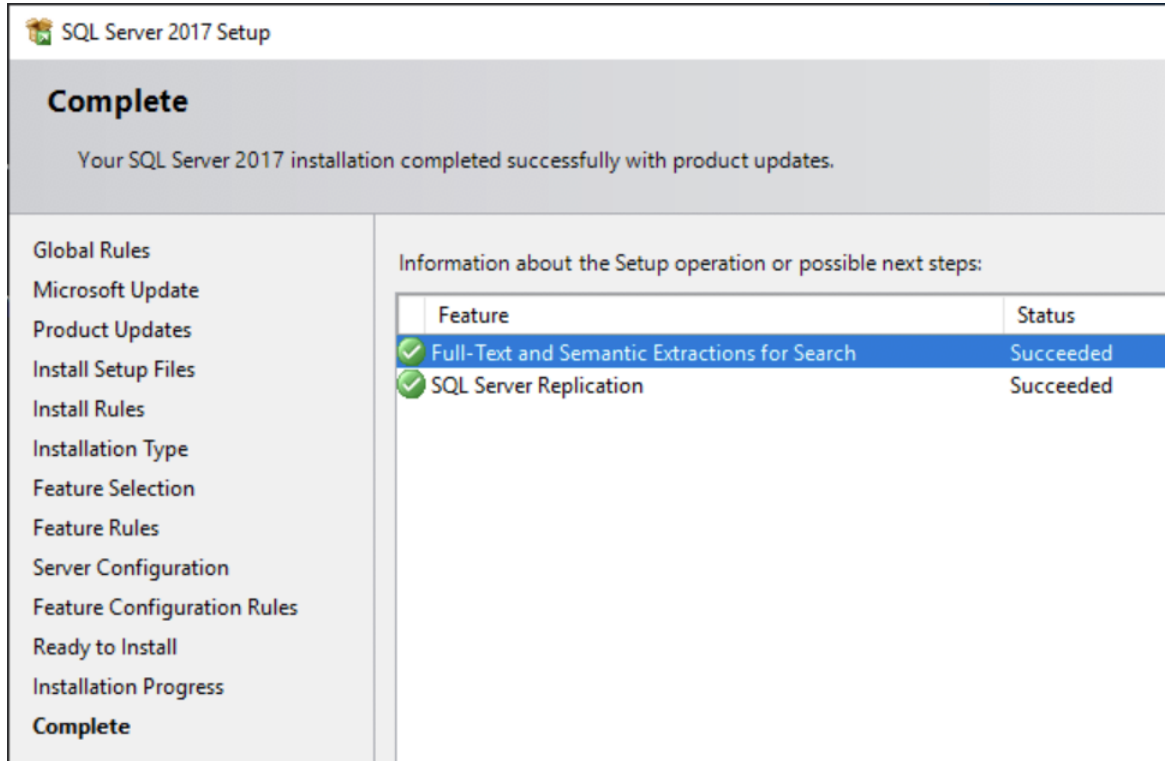
Microsoft SQL Repository

Note

These instructions are specific to Microsoft SQL Server 2017 Evaluation Edition, running on Windows Server 2019, and may require adjustments for other environments.

Install Microsoft SQL Server and Associated Tools

1. Install Microsoft SQL Server:
 - a. On the Installation has completed successfully! page of the installer, click Customize to launch the SQL Server Setup application.
 - b. Select the server instance you just created, and continue through setup. On the Feature Selection step, select *at least* the following options:
 - SQL Server Replication
 - Full-Text and Semantic Extractions for Search
 - c. Continue through the setup and verify that the required options were successfully installed, as displayed on the Complete page.



2. Download and install SQL Server Management Studio (SSMS).
3. Restart the server.
4. Launch SSMS, and connect to the SQL server instance.
5. From the Object Explorer, right-click the SQL server instance, and then click Properties.
6. On the Security page, in the Server authentication area, select SQL Server and Windows Authentication Mode, and then click OK.
7. From the Object Explorer, right-click the SQL server instance, and then click Restart.
8. Configure TCP/IP:
 - a. Launch SQL Server Configuration Manager.
 - b. From the left pane, expand the SQL Server Network Configuration node, and click Protocols for *serverName*.
 - c. Double-click TCP/IP.

- d. In the TCP/IP Properties window, from the Protocol tab, click the Enable drop-down menu, and select Yes.
- e. Click the IP Addresses tab, and make the following changes under IPAll, and then click OK:
 - In the TCP Dynamic Ports field, enter 0.
 - In the TCP Port field, enter 1433.
- f. From the left pane, click SQL Server Services, right-click SQL Server (*serverName*), and then click Restart.
- g. Configure the firewall to allow IDM to access the SQL Server.

Install and Configure IDM to Use the SQL Repository

1. Install IDM.

Warning

Do not start IDM.

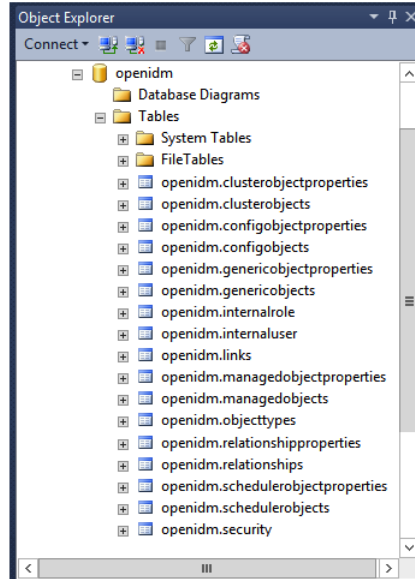
2. Import the IDM data definition language script into Microsoft SQL:
 - a. Launch SSMS.
 - b. In the Connect to Server window, select Windows Authentication and click Connect.
 - c. From the main menu, select File > Open > File, navigate to the data definition language script (`C:\path\to\openidm\db\mssql\scripts\openidm.sql`), and click Open.
 - d. Click Execute.

SSMS displays a message in the Messages tab:

```
Commands completed successfully.  
Completion time: 2020-11-02T09:26:39.1548666-08:00
```

Executing the `openidm.sql` script creates an `openidm` database for use as the internal repository, and an `openidm` user with password `openidm` who has all the required privileges to update the database. You may need to refresh the view in SSMS to see the `openidm` database in the Object Explorer.

If you expand Databases > `openidm` > Tables, you should see the IDM tables in the `openidm` database:



- Execute the script that creates the tables required by the workflow engine; for example:

```
sqlcmd -S localhost -d openidm ^  
-i C:\path\to\openidm\db\mssql\scripts\flowable.mssql.all.create.sql  
(1 rows affected)  
(1 rows affected)  
(0 rows affected)  
...
```

Note

When you run the `flowable.mssql.all.create.sql` script, you may see the following warning in the log:

```
Warning! The maximum key length is 900 bytes. The index 'ACT_UNIQ_PROCDEF' has maximum  
length of 1024 bytes. For some combination of large values, the insert/update operation will fail.
```

It is very unlikely that the key length will be an issue in your deployment, and you can safely ignore this warning.

- If you are going to direct audit logs to this repository, run the script that sets up the audit tables:

```
sqlcmd -S localhost -d openidm ^  
-i C:\path\to\openidm\db\mssql\scripts\audit.sql
```

- Download the Microsoft JDBC Drivers for SQL Server:
 - Download the JDBC Drivers from Microsoft's download site. IDM requires at least version 7.2 of the driver, which supports OSGi by default.

- b. Extract the driver JAR files.
- c. Copy the JAR file that corresponds to your Java environment to the `C:\path\to\openidm\bundle` directory. For example:

```
copy mssql-jdbc-7.4.1.jre11.jar C:\path\to\openidm\bundle
```

6. Download the JDBC OSGi Service Package JAR and place it in the `C:\path\to\openidm\bundle` directory.

Note

IDM was tested with version 1.0.0 of the service package.

7. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd C:\path\to\openidm\my-project\conf\  
del repo.ds.json
```

8. Copy the database connection configuration file for Microsoft SQL (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cd C:\path\to\openidm  
copy db\mssql\conf\datasource.jdbc-default.json my-project\conf\  
copy db\mssql\conf\repo.jdbc.json my-project\conf\  

```

9. Update the connection configuration to reflect your Microsoft SQL deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{  
  "driverClass" : "com.microsoft.sqlserver.jdbc.SQLServerDriver",  
  "jdbcUrl" : "jdbc:sqlserver://  
&{openidm.repo.host};instanceName=default;databaseName=openidm;applicationName=OpenIDM",  
  "databaseName" : "openidm",  
  "username" : "openidm",  
  "password" : "openidm",  
  "connectionTimeout" : 30000,  
  "connectionPool" : {  
    "type" : "hikari",  
    "minimumIdle" : 20,  
    "maximumPoolSize" : 50  
  }  
}
```

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

+ *Set in an IDM Properties File*

Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file, for example:

```
openidm.repo.host=localhost
openidm.repo.port=1433
```

+ *Set as an Environment Variable*

Set the properties in the `OPENIDM_OPTS` environment variable before startup. You must include the JVM memory options when you set this variable. For example:

```
set:OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=1433"
```

10. Start IDM.

Oracle DB Repository

Before you set up Oracle DB as the IDM repository, confer with your Oracle DBA to create the database schema, tables, and users. This section assumes that you have configured an Oracle DB with *Local Naming Parameters* (`tnsnames.ora`) and a service user for IDM.

Important

IDM supports two connection pools for an Oracle DB:

- Hikari Connection Pool (HikariCP), described in the [HikariCP GitHub Repository](#)
- Oracle Universal Connection Pool (Oracle UCP), described in the [Universal Connection Pool for JDBC Developer's Guide](#)

Many steps in this procedure will depend on your connection pool type.

To Set Up Oracle as an IDM Repository

1. As the appropriate schema owner, import the IDM schema using the data definition language script (`/path/to/openidm/db/oracle/scripts/openidm.sql`).
2. Use the Oracle SQL Developer Data Modeler to run the script that creates the tables required by the workflow engine:

```
/path/to/openidm/db/oracle/scripts/flowable.oracle.all.create.sql
```

3. If you are planning to direct audit logs to this repository, run the script that sets up audit tables. Use the Oracle SQL Developer Data Modeler to run the following script:

```
/path/to/openidm/db/oracle/scripts/audit.sql
```

- Set the host and port of the Oracle DB instance, either in the `resolver/boot.properties` file or through the `OPENIDM_OPTS` environment variable.

+ *Set in an IDM Properties File*

If you use the `resolver/boot.properties` file, set values for the following variables:

- `openidm.repo.host = localhost`
- `openidm.repo.port = 1521`

+ *Set as an Environment Variable*

If you use the `OPENIDM_OPTS` environment variable, include the JVM memory options when you set the repo host and port. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=1521"
```

- Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
rm /path/to/openidm/my-project/conf/repo.ds.json
```

- Copy the Oracle DB repository configuration file (`repo.jdbc.json`) to your project's configuration directory:

```
cp /path/to/openidm/db/oracle/conf/repo.jdbc.json my-project/conf/
```

+ *For Oracle UCP Only*

Edit the `repo.jdbc.json` file as follows:

```
{
  "dbType" : "ORACLE",
  "useDataSource" : "ucp-oracle",
  ...
}
```

- Copy the connection configuration file to your project's configuration directory and edit the file for your Oracle DB deployment. The connection configuration file depends on the connection pool that you use:

+ *For Hikari CP*

1. Copy the following file:

```
cp /path/to/openidm/db/oracle/conf/datasource.jdbc-default.json my-project/conf/
```

Edit the file to reflect your deployment. The default configuration for a HikariCP connection pool is as follows:

```
{
  "driverClass" : "oracle.jdbc.OracleDriver",
  "jdbcUrl" : "jdbc:oracle:thin:@//&{openidm.repo.host}&{openidm.repo.port}/
  DEFAULTCATALOG",
  "databaseName" : "openidm",
  "username" : "openidm",
  "password" : "openidm",
  "connectionTimeout" : 30000,
  "connectionPool" : {
    "type" : "hikari",
    "minimumIdle" : 20,
    "maximumPoolSize" : 50
  }
}
```

The `jdbcUrl` corresponds to the URL of the Oracle DB listener, including the service name, based on your configured Local Naming Parameters `tnsnames.ora`. Set this parameter according to your database environment.

The `DEFAULTCATALOG` refers to the SID (system identifier), for example, `orcl`.

The `username` and `password` correspond to the credentials of the service user that connects from IDM.

+ For Oracle UCP

1. Copy the following file:

```
cp /path/to/openidm/db/oracle/conf/datasource.jdbc-ucp-oracle.json my-project/conf/
```

Edit the file to reflect your deployment. The default connection configuration for an Oracle UCP connection pool is as follows:

```
{
  "databaseName" : "openidm",
  "jsonDataSource" : {
    "class" : "oracle.ucp.jdbc.PoolDataSourceImpl",
    "settings" : {
      "connectionFactoryClassName" : "oracle.jdbc.pool.OracleDataSource",
      "url" : "jdbc:oracle:thin:@&{openidm.repo.host}&{openidm.repo.port}:SID",
      "user" : "openidm",
      "password" : "openidm",
      "connectionTimeout" : "30000",
      "minPoolSize" : 20,
      "maxPoolSize" : 50
    }
  }
}
```

The `url` corresponds to the URL of the Oracle DB listener, including the service ID (`SID`), based on your configured Local Naming Parameters `tnsnames.ora`. Set this property to the appropriate value for your environment, for example: `jdbc:oracle:thin:@localhost:1521:orcl`.

The `user` and `password` correspond to the credentials of the service user that connects from IDM.

8. Create an OSGi bundle for the Oracle DB driver, as follows:

a. Download the JDBC drivers for your Oracle DB version.

The files that you download depend on your Oracle DB version, and on whether you are using HikariCP or Oracle UCP. Because the version numbers change with minor updates, you must search for the precise corresponding files on [oracle.com](https://www.oracle.com):

- Download the `ojdbc*.jar` file that corresponds to your Oracle DB version.
- Download the most recent `bnd` JAR file from <https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/>. The `bnd` utility lets you create OSGi bundles for JDBC libraries that do not yet support OSGi.

+ *For Oracle UCP Only*

1. Download the following additional files:

- `ucp.jar`
- `ons.jar`

Copy the downloaded files to the `/path/to/openidm/db/oracle/scripts` directory.

b. The `/path/to/openidm/db/oracle/scripts` directory includes an `ojdbc8.bnd` file that specifies the version information for your JDBC driver.

Edit the driver version in that file if necessary. The default file is as follows:

```
version=12.2.0.1
Export-Package: *;version=${version}
Bundle-Name: Oracle Database 12.2.0.1 JDBC Driver
Bundle-SymbolicName: oracle.jdbc.OracleDriver
Bundle-Version: ${version}
Import-Package: *;resolution:=optional
```

Note

- Do not include trailing zeros in the version number. For example, for Oracle 12.2.0.1.0, set the version string to `version=12.2.0.1`.
- Oracle DB 12cR2 (12.2.0.1) uses the drivers in `ojdbc8.jar`.

- c. From the `/path/to/openidm/db/oracle/scripts` directory, run the following command to create the OSGi bundle, replacing the `*` with your Oracle DB driver version:

```
java -jar biz.aQute.bnd-version.jar wrap --properties ojdbc*.bnd --output ojdbc*-osgi.jar ojdbc*.jar
```

+ For Oracle UCP Only

1. Create `bnd` files for the `ucp.jar` and `ons.jar` files. The following examples assume version 12.2.0 Oracle JDBC drivers:

- `ucp.bnd`

```
version=12.2.0
Export-Package: oracle.ucp.*;version=${version}
Bundle-Name: Oracle Universal Connection Pool
Bundle-SymbolicName: oracle.ucp
Bundle-Version: ${version}
Import-Package: *;resolution:=optional
DynamicImport-Package: *
```

- `ons.bnd`

```
version=12.2.0
Export-Package: *;version=${version}
Bundle-Name: Oracle ONS
Bundle-SymbolicName: oracle.ons
Bundle-Version: ${version}
Import-Package: *;resolution:=optional
```

Save the `bnd` files in the `/path/to/openidm/db/oracle/scripts` directory, then run the following commands to create the corresponding OSGi bundles:

```
cd /path/to/openidm/db/oracle/scripts
java -jar biz.aQute.bnd-version.jar wrap --properties ucp.bnd --output ucp-osgi.jar
ucp.jar
java -jar biz.aQute.bnd-version.jar wrap --properties ons.bnd --output ons-osgi.jar
ons.jar
```

You can ignore any `private references` warnings that are logged when you build these bundles.

- d. Move all the OSGi bundle files to the `openidm/bundle` directory.
9. When you have set up Oracle DB for use as the internal repository, make sure that the server starts without errors.

PostgreSQL Repository

This procedure assumes that PostgreSQL is installed and running on the local host. For supported versions, see Supported Repositories in the *Release Notes*.

Before starting IDM for the first time, configure the server to use a PostgreSQL repository, as described in the following procedure:

Note

The `path/to/openidm/db/postgresql/scripts/createuser.pgsql` script creates an `openidm` database and role, with a default password of `openidm`. The script also grants the appropriate permissions.

Edit this script if you want to change the password of the `openidm` role, for example:

```
create database openidm encoding 'utf8';
create role openidm with LOGIN NOSUPERUSER NOCREATEDB NOCREATEROLE inherit password 'newPassword';
grant all privileges on database openidm to openidm;
```

1. Edit the Postgres client authentication configuration file, `pg_hba.conf`. Add the following entries for the following users: `postgres` and `openidm`:

local	all	openidm	trust
local	all	postgres	trust

2. As the `postgres` user, execute the `createuser.pgsql` script as follows:

```
psql -U postgres < /path/to/openidm/db/postgresql/scripts/createuser.pgsql
CREATE DATABASE
CREATE ROLE
GRANT
```

3. Run the `openidm.pgsql` script as the new `openidm` user that you created in the first step:

```
psql -U openidm < /path/to/openidm/db/postgresql/scripts/openidm.pgsql

CREATE SCHEMA
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE INDEX
CREATE INDEX
...
START TRANSACTION
INSERT 0 1
INSERT 0 1
COMMIT
CREATE INDEX
CREATE INDEX
```

Your database has now been initialized.

4. Run the script that creates the tables required by the workflow engine:

```
psql -d openidm -U openidm < /path/to/openidm/db/postgresql/scripts/flowable.postgres.all.create.sql
```

5. If you plan to direct audit logs to this repository, run the script that sets up the audit tables:

```
psql -d openidm -U openidm < /path/to/openidm/db/postgresql/scripts/audit.pgsql
```

6. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

7. Copy the database connection configuration file for PostgreSQL (`datasource.jdbc-default.json`) and the database table file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cp /path/to/openidm/db/postgresql/conf/datasource.jdbc-default.json my-project/conf/
cp /path/to/openidm/db/postgresql/conf/repo.jdbc.json my-project/conf/
```

8. Update the connection configuration to reflect your PostgreSQL deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{
  "driverClass" : "org.postgresql.Driver",
  "jdbcUrl" : "jdbc:postgresql://&{openidm.repo.host}&{openidm.repo.port}/openidm",
  "databaseName" : "openidm",
  "username" : "openidm",
  "password" : "openidm",
  "connectionTimeout" : 30000,
  "connectionPool" : {
    "type" : "hikari",
    "minimumIdle" : 20,
    "maximumPoolSize" : 50
  }
}
```

If you changed the password in step 1 of this procedure, edit the `datasource.jdbc-default.json` file to set the value for the `password` field to whatever password you set for the `openidm` user.

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

+ *Set in an IDM Properties File*

Set the values in your `resolver/boot.properties` file:

```
openidm.repo.host = localhost
openidm.repo.port = 5432
```

+ *Set as an Environment Variable*

Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=5432"
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=5432
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

9. PostgreSQL is now set up for use as the internal repository. Make sure that the server starts without errors.
10. Set up indexes to tune the PostgreSQL repository according to your specific deployment.

Important

No indexes are set by default. If you do not tune the repository correctly by creating the required indexes, the performance of your service can be severely impacted. For example, setting too many indexes can have an adverse effect on performance during managed object creation. Conversely, not indexing fields that are searched will severely impact search performance.

IDM includes a `/path/to/openidm/db/postgresql/scripts/default_schema_optimization.pgsq` script that sets up a number of indexes. This script includes extensive comments on the indexes that are being created. Review the script before you run it to ensure that all the indexes are suitable for your deployment.

When you have refined the script for your deployment, execute the script as a user with superuser privileges, so that the required extensions can be created. By default, this is the `postgres` user:

```
psql -U postgres openidm < /path/to/openidm/db/postgresql/scripts/default_schema_optimization.pgsql
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
```

IBM DB2 Repository

This section makes the following assumptions about the DB2 environment. If these assumptions do not match your DB2 environment, adapt the subsequent instructions accordingly.

- DB2 is running on the localhost, and is listening on the default port (50000).
- The user `db2inst1` is configured as the DB2 instance owner, and has the password `Passw0rd1`.

This section assumes that you will use basic username/password authentication. You can also configure Kerberos authentication with a DB2 repository.

Before you start, make sure that the server is stopped.

```
/path/to/openidm/shutdown.sh
OpenIDM is not running, not stopping.
```

Configure IDM to use the DB2 repository, as described in the following steps:

1. Create an OSGi bundle for the DB2 JDBC driver, as follows:
 - a. Download the DB2 JDBC driver for your database version from the IBM download site and place it in the `openidm/db/db2/scripts` directory.

Use either the `db2jcc.jar` or `db2jcc4.jar`, depending on your DB2 version. For more information, see the [DB2 JDBC Driver Versions](#).

```
ls /path/to/db/db2/scripts/
db2jcc.jar  openidm.sql
```

- b. Create a `bnd` file and edit it to match the version information for your JDBC driver.

You can use the sample `bnd` file located in `openidm/db/mssql/scripts`. Copy that file to the directory with the JDBC driver:

```
cd /path/to/openidm/db
cp mssql/scripts/sqljdbc4.bnd db2/scripts/
ls db2/scripts
db2jcc.jar  openidm.sql  sqljdbc4.bnd
```

The JDBC driver version information for your driver is located in the `Specification-Version` property in the MANIFEST file of the driver.

```
cd /path/to/openidm/db/db2/scripts
unzip -q -c db2jcc.jar META-INF/MANIFEST.MF
Manifest-Version: 1.0
Created-By: 1.4.2 (IBM Corporation)
```

Edit the `bnd` file to match the JDBC driver version:

```
more sqljdbc4.bnd
...
version=1.0
Export-Package: *;version=${version}
Bundle-Name: IBM JDBC DB2 Driver
Bundle-SymbolicName: com.ibm.db2.jcc.db2driver
Bundle-Version: ${version}
```

- c. Download the most recent `bnd` JAR file from <https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/>. The `bnd` utility lets you create OSGi bundles for JDBC libraries that do not yet support OSGi.

Place the `bnd` JAR file in the same directory as the JDBC driver:

```
ls /path/to/openidm/db/db2/scripts
biz.aQute.bnd-version.jar db2jcc.jar
```

- d. Change to the directory in which the script files are located and run the following command to create the OSGi bundle:

```
cd /path/to/openidm/db/db2/scripts
java -jar biz.aQute.bnd-version.jar wrap --properties sqljdbc4.bnd --output db2jcc-osgi.jar
db2jcc.jar
```

This command creates an OSGi bundle, as defined by the `--output` option: `db2jcc-osgi.jar`:

```
ls /path/to/openidm/db/db2/scripts
biz.aQute.bnd-version.jar db2jcc-osgi.jar db2jcc.jar
```

- e. Move the OSGi bundle file to the `openidm/bundle` directory:

```
mv db2jcc-osgi.jar /path/to/openidm/bundle/
```

2. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

3. Copy the database connection configuration file for DB2 (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cp /path/to/openidm/db/db2/conf/datasource.jdbc-default.json my-project/conf/
cp /path/to/openidm/db/db2/conf/repo.jdbc.json my-project/conf/
```


4. Update the connection configuration to reflect your DB2 deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{
  "driverClass" : "com.ibm.db2.jcc.DB2Driver",
  "jdbcUrl" : "jdbc:db2://&{openidm.repo.host}&{openidm.repo.port}/
dopenidm:retrieveMessagesFromServerOnGetMessage=true;",
  "databaseName" : "sopenidm",
  "username" : "openidm",
  "password" : "openidm",
  "connectionTimeout" : 30000,
  "connectionPool" : {
    "type" : "hikari",
    "minimumIdle" : 20,
    "maximumPoolSize" : 50
  }
}
```

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

+ *Set in an IDM Properties File*

Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file, for example:

```
openidm.repo.host = localhost
openidm.repo.port = 50000
```

+ *Set as an Environment Variable*

Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=50000"
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=50000
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

5. Create a user database for IDM (`dopenidm`).

```
db2 create database dopenidm
```

6. Import the IDM data definition language script into your DB2 instance.

```
cd /path/to/openidm
db2 -i -tf db/db2/scripts/openidm.sql
```

The database schema is defined in the `SOPENIDM` database.

7. You can show the list of tables in the repository, using the `db2 list` command, as follows:

```
db2 LIST TABLES for all
```

Table/View	Schema	Type	Creation time
CLUSTEROBJECTPROPERTIES	SOPENIDM	T	2015-10-01-11.58.05.968933
CLUSTEROBJECTS	SOPENIDM	T	2015-10-01-11.58.05.607075
CONFIGOBJECTPROPERTIES	SOPENIDM	T	2015-10-01-11.58.01.039999
CONFIGOBJECTS	SOPENIDM	T	2015-10-01-11.58.00.570231
GENERICOBJECTPROPERTIES	SOPENIDM	T	2015-10-01-11.57.59.583530
GENERICOBJECTS	SOPENIDM	T	2015-10-01-11.57.59.152221
INTERNALUSER	SOPENIDM	T	2015-10-01-11.58.04.060990
LINKS	SOPENIDM	T	2015-10-01-11.58.01.349194
MANAGEDOBJECTPROPERTIES	SOPENIDM	T	2015-10-01-11.58.00.261556
MANAGEDOBJECTS	SOPENIDM	T	2015-10-01-11.57.59.890152
...			

8. Connect to the `openidm` database, and run the script that creates the tables required by the workflow engine:

```
db2 connect to dopenidm
db2 -i -tf /path/to/openidm/db/db2/scripts/flowable.db2.all.create.sql
```

9. If you plan to direct audit logs to this repository, run the script that sets up the audit tables:

```
db2 -i -tf /path/to/openidm/db/db2/scripts/audit.sql
```

When you have set up DB2 for use as the internal repository, make sure that the server starts without errors.

Kerberos Authentication With a DB2 Repository

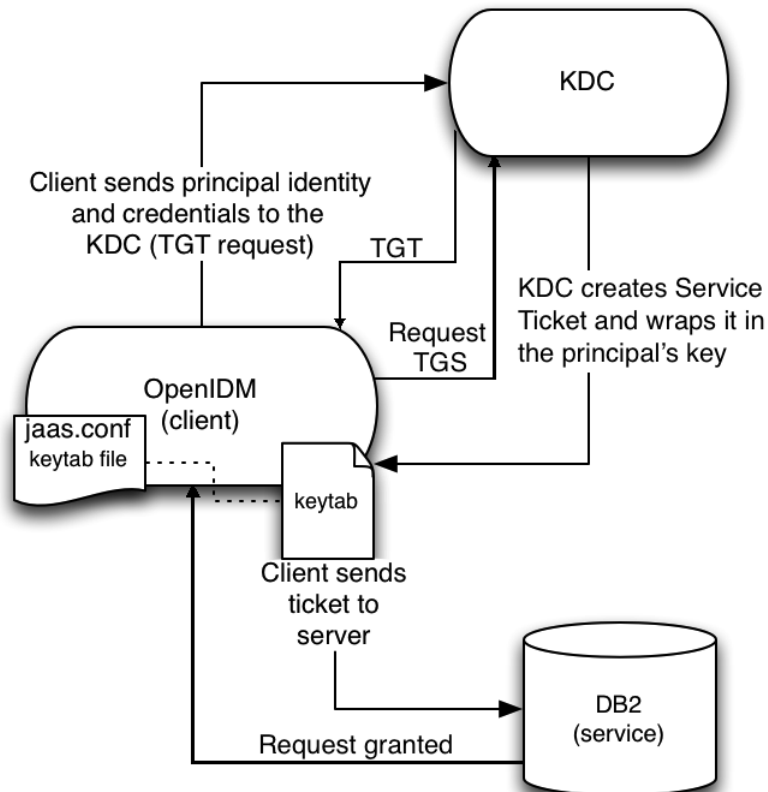
By default, IDM uses the username and password configured in the repository connection configuration file (`conf/datasource.jdbc-default.json`) to connect to the DB2 repository. You can configure IDM to use Kerberos authentication instead.

In this scenario, IDM acts as a *client* and requests a Kerberos ticket for a *service*, which is DB2, through the JDBC driver.

This section assumes that you have configured DB2 for Kerberos authentication. If that is not the case, follow the instructions in the corresponding DB2 documentation before you read this section.

The following diagram shows how the ticket is obtained and how the keytab is referenced from IDM's `jaas.conf` file.

Using Kerberos to Connect to a DB2 Repository



+ Configure IDM for Kerberos Authentication

1. Create a keytab file, specifically for use by IDM.

A Kerberos keytab file (`krb5.keytab`) is an encrypted copy of the host's key. The keytab enables DB2 to validate the Kerberos ticket that it receives from IDM. You must create a keytab file on the host that IDM runs on. The keytab file must be secured in the same way that you would secure any password file. Specifically, only the user running IDM should have read and write access to this file.

Create a keytab for DB2 authentication, in the file `openidm/security/idm.keytab/:`

```
kadmin -p kadmin/admin -w password
kadmin: ktadd -k /path/to/openidm/security/idm.keytab db2/idm.example.com
```

2. Make sure that the DB2 user has read access to the keytab.
3. Copy the DB2 Java Authentication and Authorization Service (JAAS) configuration file to the IDM `security` directory:

```
cp /path/to/openidm/db/db2/conf/jaas.conf /path/to/openidm/security/
```

By default, IDM assumes that the keytab is in the file `openidm/security/idm.keytab` and that the principal identity is `db2/idm.example.com@EXAMPLE.COM`. Change the following lines in the `jaas.conf` file if you are using a different keytab:

```
keyTab="security/idm.keytab"
principal="db2/idm.example.com@EXAMPLE.COM"
```

4. Adjust the authentication details in your DB2 connection configuration file (`conf/datasource.jdbc-default.json`). Edit that file to remove `password` field and change the username to the instance owner (`db2`). The following excerpt shows the modified file:

```
{
  ...
  "databaseName" : "sopenidm",
  "username" : "db2",
  "connectionTimeout" : 30000,
  ...
}
```

5. Edit your project's `conf/system.properties` file, to add the required Java options for Kerberos authentication.

In particular, add the following two lines to that file:

```
db2.jcc.securityMechanism=11
java.security.auth.login.config=security/jaas.conf
```

6. Restart IDM.

Chapter 7

Configure Your JDBC Repository

The following topics apply only if you have set up a JDBC repository, as described in "*Select a Repository*":

- "JDBC Database Access Rights"
- "Configure Case Insensitivity for a JDBC Repo"
- "Connect to a JDBC Repo Over SSL"

JDBC Database Access Rights

In general, IDM requires minimal access rights to the JDBC repository for daily operation. This section lists the minimum permissions required, and suggests a strategy for restricting database access in your deployment.

The JDBC repository used by IDM requires only one *relevant* user—the service account that is used to create the tables. Generally, the details of this account are configured in the repository connection file (`datasource.jdbc-default.json`). By default, the username and password for this account are `openidm` and `openidm`, regardless of the database type.

All other users are created by the `db/database-type/scripts/openidm.sql` script. The `openidm` user account must have SELECT, UPDATE, INSERT, and DELETE permissions on all the `openidm` tables that are created by this script, by the scripts that create the tables specific to the Flowable workflow engine, and by the script that sets up the audit tables if you are using the repository audit event handler.

Configure Case Insensitivity for a JDBC Repo

A DS repository is case-insensitive by default. The supported JDBC repositories are generally case-sensitive by default. Case-sensitivity can cause issues if queries expect results to be returned, regardless of case.

For example, with the default configuration of a MySQL database, a search for an email address of `scarter@example.com` might return a result, while a search for `scarter@EXAMPLE.COM` might return an `Unable to find account` error.

If you need to support case-insensitive queries, you must configure a case-insensitive collation in your JDBC repository, on the specific columns that require it. For example:

- For a generic managed object mapping in MySQL or MariaDB, change the default collation of the `managedobjectproperties.propvalue` column to `utf8_general_ci`. Note that this changes case-sensitivity for *all* managed object properties. To change case-sensitivity for all the properties of a specific object, specify a different table for the `propertiesTable` entry in your `repo.jdbc.json` for that object, and adjust the collation on that table. To change case-sensitivity only for certain properties of an object, use an explicit mapping.
- For a PostgreSQL repository, use an explicit table structure if you require case-insensitivity. Managing case-insensitivity at scale with generic tables in PostgreSQL is not supported. For more information about generic and explicit object mappings, see "Generic and Explicit Mappings With a JDBC Repository" in the *Object Modeling Guide*.
- For an Oracle DB repository, see the corresponding Oracle documentation.
- For a SQL Server repository, see the corresponding Windows documentation.
- For a DB2 repository, see the corresponding DB2 documentation.

Connect to a JDBC Repo Over SSL

This procedure assumes that you have already set up your JDBC repository, as described in the previous sections. The exact steps to connect to a JDBC repository over SSL depend on your repository. This procedure describes the steps for a MySQL 8 repository. If you are using a different JDBC repository, use the corresponding documentation for that repository, and adjust the steps accordingly.

1. Change the `jdbcUrl` property in your repository connection configuration file (`conf/datasource.jdbc-default.json`).

The exact value of the `jdbcUrl` property will depend on your JDBC database, and on the version of your JDBC driver:

+ *Configuration for MySQL with JDBC Driver Version 8.0.12 or Earlier*

```
"jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/openidm?  
allowMultiQueries=true&characterEncoding=utf8&useSSL=true&verifyServerCertificate=true&requireSSL=true"
```

+ *Configuration for MySQL with JDBC Driver Version 8.0.13 or Later*

```
"jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}&{openidm.repo.port}/openidm?
allowMultiQueries=true&characterEncoding=utf8&sslMode=VERIFY_CA&requireSSL=true"
```

Note

For Azure MySQL, JDBC Driver Version 8.0.17+ is **required**.

2. Create and verify the SSL certificate and key files required to support encrypted connections to the JDBC repository.

For MySQL 8, use one of the procedures in the [MySQL docs](#).

3. Configure the JDBC repository to use encrypted connections.

For MySQL 8, follow the [MySQL docs](#).

4. Check that the connection to the database is over SSL by running a command similar to the following:

```
mysql -u root -P 3306 -p
mysql>show variables like "%have_ssl%";

+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_ssl      | YES   |
+-----+-----+
1 row in set (0.00 sec)
```

5. Convert your MySQL client key and certificate files to a PKCS #12 archive. For example:

```
openssl pkcs12 -export \
-in client-cert.pem \
-inkey client-key.pem \
-name "mysqlclient" \
-passout pass:changeit \
-out client-keystore.p12
```

6. Import the `client-keystore.p12` into the IDM keystore:

```
keytool \
-importkeystore \
-srckeystore client-keystore.p12 \
-srcstoretype pkcs12 \
-srcstorepass changeit \
-destkeystore /path/to/openidm/security/keystore.jceks \
-deststoretype jceks \
-deststorepass changeit
```

Important

For AWS RDS MySQL **and** Azure MySQL, no client certificates are provided. In this case, you must create an empty keystore for client certificates, and add the following to the `jdbcUrl` property in your repository connection configuration file (`conf/datasource.jdbc-default.json`):

```
&clientCertificateKeyStoreUrl=file:/opt/idm/security/empty.jks&clientCertificateKeyStorePassword=changeit
```

7. Import your MySQL CA certificate into the IDM truststore.

```
keytool \  
-importcert \  
-trustcacerts \  
-file ca-cert.pem \  
-alias "DB cert" \  
-keystore /path/to/openidm/security/truststore
```

You are prompted for a keystore password. You must use the same password as is shown in your `resolver/boot.properties` file. The default truststore password is:

```
openidm.truststore.password=changeit
```

After entering a keystore password, you are prompted with the following question. Assuming you have included an appropriate `ca-cert.pem` file, enter **yes**.

```
Trust this certificate? [no]:
```

8. Open your project's `conf/system.properties` file. Add the following line to that file. If appropriate, substitute the path to your own truststore:

```
# Set the truststore  
javax.net.ssl.trustStore=&{idm.install.dir}/security/truststore
```

Even if you are setting up this instance of IDM as part of a cluster, you must configure this initial truststore. After this instance joins a cluster, the SSL keys in this particular truststore are replaced.

Chapter 8

Configuration and Monitoring

These topics describe the startup configuration variables for IDM, and how to check system health:

- "Specify the Startup Configuration"
- "Monitor Server Health"
- "Display Installed Modules and Features"

Specify the Startup Configuration

By default, IDM starts with the configuration, script, and binary files in the `openidm/conf`, `openidm/script`, and `openidm/bin` directories. You can launch IDM with a different set of configuration, script, and binary files for test purposes, to manage different projects, or to run one of the included samples.

The `startup.sh` script enables you to specify the following elements of a running instance:

`-p | --project-location {/path/to/project/directory}`

The project location specifies the directory that contains the configuration and script files that IDM will use.

All configuration objects and any artifacts that are not in the bundled defaults (such as custom scripts) *must* be included in the project location. These objects include all files otherwise included in the `openidm/conf` and `openidm/script` directories.

For example, the following command starts the server with the configuration of the `sync-with-csv` sample (located in `/path/to/openidm/samples/sync-with-csv`):

```
./startup.sh -p /path/to/openidm/samples/sync-with-csv
```

If you do not provide an absolute path, the project location path is relative to the system property, `user.dir`. IDM sets `idm.instance.dir` to that relative directory path. Alternatively, if you start the server without the `-p` option, IDM sets `idm.instance.dir` to `/path/to/openidm`.

Note

In this documentation, "your project" refers to the value of `idm.instance.dir`.

-w | --working-location {/path/to/working/directory}

The working location specifies the directory in which the embedded DS instance is installed, and the directory to which IDM writes its database cache, audit logs, and felix cache. The working location includes everything that is in the default `db/`, `audit/`, and `felix-cache/` subdirectories.

The following command specifies that IDM writes its database cache and audit data to `/Users/admin/openidm/storage`:

```
./startup.sh -w /Users/admin/openidm/storage
```

If you do not provide an absolute path, the path is relative to the system property, `user.dir`. IDM sets `idm.data.dir` to that relative directory path. If you do not specify a working location, IDM sets `idm.data.dir` to `/path/to/openidm`. This means the default working location data is located in the `openidm/db`, `openidm/felix-cache` and `openidm/audit` directories.

Note that this property does not affect the location of the IDM system logs. To change the location of these logs, edit the `conf/logging.properties` file.

You can also change the location of the Felix cache, by editing the `conf/config.properties` file, or by starting the server with the `-s` option, described later in this section.

-c | --config {/path/to/config/file}

A customizable startup configuration file (named `launcher.json`) enables you to specify how the OSGi Framework is started.

Unless you are working with a highly customized deployment, you should not modify the default framework configuration.

-P {property=value}

Any properties passed to the startup script with the `-P` option are used when the server loads the `launcher.json` startup configuration file.

Options specified here have the lowest order of precedence when the configuration is loaded. If the same property is defined in any other configuration source, the value specified here is ignored.

-s | --storage {/path/to/storage/directory}

Specifies the OSGi storage location of the cached configuration files.

You can use this option to redirect output if you are installing on a read-only filesystem volume. For more information, see "Securing IDM Server Files With a Read-Only Installation" in

the *Security Guide*. This option is also useful when you are testing different configurations. Sometimes when you start the server with two different sample configurations, one after the other, the cached configurations are merged and cause problems. Specifying a storage location creates a separate `felix-cache` directory in that location, and the cached configuration files remain completely separate.

Additionally, IDM sets the system property `idm.install.dir` to the location IDM is installed in. For example, if IDM was installed in `/Users/admin/openidm/`, that is what `idm.install.dir` will be set to.

For information about changing the startup configuration by substituting property values, see "Property Value Substitution" in the *Setup Guide*.

Monitor Server Health

Because IDM is highly modular and configurable, it is often difficult to assess whether a system has started up successfully, or whether the system is ready and stable after dynamic configuration changes have been made.

The health check service lets you monitor the status of internal resources.

To monitor the status of external resources such as LDAP servers and external databases, use the commands described in "*Check External System Status Using REST*" in the *Connectors Guide*.

- "Basic Health Checks"
- "Session Information"
- "Health Check Service"

Basic Health Checks

The health check service reports on the state of the server and outputs this state to the OSGi console and to the log files. The server can be in one of the following states:

- **STARTING** - the server is starting up
- **ACTIVE_READY** - all of the specified requirements have been met to consider the server ready
- **ACTIVE_NOT_READY** - one or more of the specified requirements have not been met and the server is not considered ready
- **STOPPING** - the server is shutting down

To verify the current server state, use the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/ping"
{
  "_id": "",
  "_rev": "",
  "shortDesc": "OpenIDM ready",
  "state": "ACTIVE_READY"
}
```

Session Information

To obtain information about the current IDM session, beyond basic health checks, use the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "openidm-admin",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "internal/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-admin",
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:1",
    "authenticationId": "openidm-admin",
    "id": "openidm-admin",
    "moduleId": "INTERNAL_USER",
    "queryId": "credential-internaluser-query"
  }
}
```

Note

The precise output of this command will depend on the authentication module responsible for authenticating the user. For more information about authentication modules, see "Authentication and Session Modules" in the *Security Guide*

Health Check Service

The configurable health check service verifies the status of the modules and services required for an operational system. During system startup, IDM checks that these modules and services are available and reports on any requirements that have not been met. If dynamic configuration changes are made, IDM rechecks that the required modules and services are functioning, to allow ongoing monitoring of system operation.

+ *Examples of Required Modules*

IDM checks all required modules. Examples of those modules are shown here:

```
"org.forgerock.openicf.framework.connector-framework"  
"org.forgerock.openicf.framework.connector-framework-internal"  
"org.forgerock.openicf.framework.connector-framework-osgi"  
"org.forgerock.openidm.audit"  
"org.forgerock.openidm.core"  
"org.forgerock.openidm.enhanced-config"  
"org.forgerock.openidm.external-email"  
...  
"org.forgerock.openidm.system"  
"org.forgerock.openidm.ui"  
"org.forgerock.openidm.util"  
"org.forgerock.commons.org.forgerock.json.resource"  
"org.forgerock.commons.org.forgerock.util"  
"org.forgerock.openidm.security-jetty"  
"org.forgerock.openidm.jetty-fragment"  
"org.forgerock.openidm.quartz-fragment"  
"org.ops4j.pax.web.pax-web-extender-whiteboard"  
"org.forgerock.openidm.scheduler"  
"org.ops4j.pax.web.pax-web-jetty"  
"org.forgerock.openidm.repo-jdbc"  
"org.forgerock.openidm.repo-ds"  
"org.forgerock.openidm.config"  
"org.forgerock.openidm.crypto"
```

+ *Examples of Required Services*

IDM checks all required services. Examples of those services are shown here:

```
"org.forgerock.openidm.config"  
"org.forgerock.openidm.provisioner"  
"org.forgerock.openidm.provisioner.openicf.connectorinfoprovider"  
"org.forgerock.openidm.external.rest"  
"org.forgerock.openidm.audit"  
"org.forgerock.openidm.policy"  
"org.forgerock.openidm.managed"  
"org.forgerock.openidm.script"  
"org.forgerock.openidm.crypto"  
"org.forgerock.openidm.recon"  
"org.forgerock.openidm.info"  
"org.forgerock.openidm.router"  
"org.forgerock.openidm.scheduler"  
"org.forgerock.openidm.scope"  
"org.forgerock.openidm.taskscanner"
```

You can replace the list of required modules and services, or add to it, by adding the following lines to your `resolver/boot.properties` file. Bundles and services are specified as a list of symbolic names, separated by commas:

- `openidm.healthservice.reqbundles` - overrides the default required bundles.
- `openidm.healthservice.reqservices` - overrides the default required services.
- `openidm.healthservice.additionalreqbundles` - specifies required bundles (in addition to the default list).
- `openidm.healthservice.additionalreqservices` - specifies required services (in addition to the default list).

Note

By default, the server is given 15 seconds to start up all the required bundles and services before system readiness is assessed. This is not the total start time, but the time required to complete the service startup after the framework has started. You can change this default by setting the value of the `servicestartmax` property (in milliseconds) in your `resolver/boot.properties` file. This example sets the startup time to five seconds:

```
openidm.healthservice.servicestartmax=5000
```

Display Installed Modules and Features

On a running instance, you can list the installed modules and their states using the Felix web console at <https://localhost:8443/system/console/bundles>.

As a security precaution you should remove the Felix web console bundle in a production deployment.

Alternatively, you can query the enabled features over REST. The feature availability service determines the set of possible features from the active bundles, and provides the following information:

- The name and `_id` of the feature
- Whether the feature is enabled
- If the feature is enabled, the REST endpoint on which that feature can be accessed

You can query the available features on the `info/features` endpoint, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/features?_queryFilter=true"
{
  "result": [
    {
      "_id": "retrieveUsername",
      "name": "retrieveUsername",
      "enabled": false,
      "endpoints": []
    },
    {
      "_id": "identityProviders",
      "name": "identityProviders",
      "enabled": true,
      "endpoints": [
        "identityProviders"
      ]
    },
    {
      "_id": "workflow",
      "name": "workflow",
      "enabled": true,
      "endpoints": [
        "workflow*"
      ]
    },
    {
      "_id": "passwordReset",
      "name": "passwordReset",
      "enabled": false,
      "endpoints": []
    },
    {
      "_id": "registration",
      "name": "registration",
      "enabled": true,
      "endpoints": [
        "selfservice/registration"
      ]
    },
    {
      "_id": "email",
      "name": "email",
      "enabled": false,
      "endpoints": []
    }
  ]
}
```

```
],  
  ...  
}
```


Chapter 9

Run IDM as a Cluster

To ensure that your identity management service remains available in the event of system failure, you can deploy multiple IDM instances in a cluster. In a clustered environment, each instance points to the same external repository.

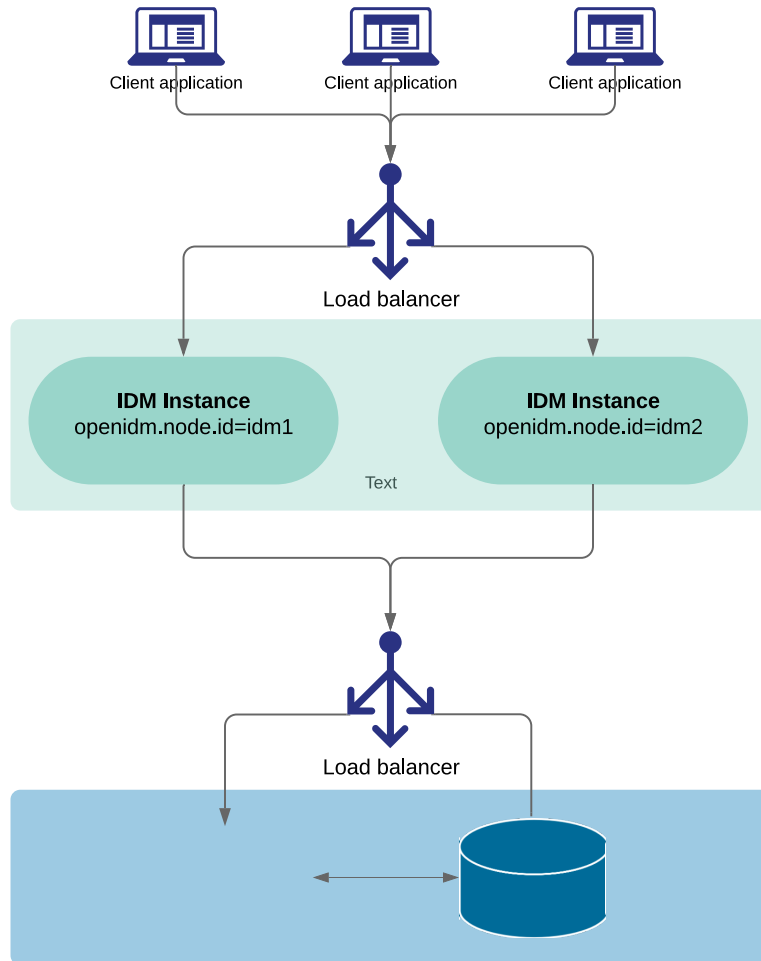
If one instance in a cluster shuts down or fails to check in with the cluster management service, a second instance will detect the failure. For example, if an instance named `instance1` loses connectivity while executing a scheduled task, the cluster manager notifies the scheduler service that `instance1` is not available. The scheduler service then attempts to clean up any jobs that `instance1` was running at that time. Note that clustered instances claim scheduled tasks in a random order. For more information, see "Scheduled Tasks Across a Cluster".

Consistency and concurrency across cluster instances is ensured using multi-version concurrency control (MVCC). MVCC provides consistency because each instance updates only the particular revision of the object that was specified in the update.

All instances in a cluster run simultaneously. When a clustered deployment is configured with a load balancer, the deployment works as an active-active high availability cluster. If the database is also clustered, IDM points to the database cluster as a single system.

IDM requires a single, consistent view of all the data it manages, including the user store, roles, schedules, and configuration. If you can guarantee this consistent view, the number and locations of IDM nodes in a cluster will be limited only by your network latency and other network factors that affect performance.

The following diagram shows an IDM deployment where both the IDM instances and the databases are clustered, and accessed through a load balancer:



This chapter describes the changes required to configure multiple IDM instances in a single cluster. It does not include instructions on configuring the various third-party load balancing options.

Important

A clustered deployment relies on system heartbeats to assess the cluster state. For the heartbeat mechanism to work, you *must* synchronize the system clocks of all the machines in the cluster using a time synchronization service that runs regularly. The system clocks must be within one second of each other. For information on how you can achieve this using the Network Time Protocol (NTP) daemon, see the NTP RFC.

Note that VM guests do not necessarily keep the same time as the host. You should therefore run a time synchronization service such as NTP on every VM.

Configure an IDM Instance as Part of a Cluster

Setting up multiple IDM instances in a cluster involves the following main steps:

1. Ensure that each instance is shut down.
2. Configure each instance to use the same external repository and the same keystore and truststore.
3. Set a unique node ID for each instance.
4. Configure the entire clustered system to use a load balancer or reverse proxy.

To configure an IDM instance as a part of a clustered deployment, follow these steps:

1. Shut down the server if it is running.
2. If you have not already done so, set up a supported repository, as described in "*Select a Repository*".

Each instance in the cluster must be configured to use the same repository, that is, the database connection configuration file (`datasource.jdbc-default.json`) for each instance must point to the same port number and IP address for the database.

Note

The configuration file `datasource.jdbc-default.json` must be the same on all nodes.

In "*Select a Repository*", you will see a reference to a data definition language script file. Do not run that script for each instance in the cluster—run it just once to set up the tables required for IDM.

Important

If an instance is *not* participating in the cluster, it must *not* share a repository with nodes that are participating in the cluster. Having non-clustered nodes use the same repository as clustered nodes will result in unexpected behavior.

3. Specify a unique node ID (`openidm.node.id`) for each instance, in one of the following ways:
 - Set the value of `openidm.node.id` in the `resolver/boot.properties` file of the instance, for example:

```
openidm.node.id = node1
```

- Set the value in the `OPENIDM_OPTS` environment variable and export that variable before starting the instance. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.node.id=node1"
./startup.sh
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dopenidm.node.id=node1
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.0.4"
OpenIDM ready
```

You can set any value for the `openidm.node.id`, as long as the value is unique within the cluster. The cluster manager detects unavailable instances by their node ID.

You *must* set a node ID for each instance, otherwise the instance fails to start. The default `resolver/boot.properties` file sets the node ID to `openidm.node.id=node1`.

4. Set the cluster configuration in the `conf/cluster.json` file.

By default, configuration changes are persisted in the repository so changes that you make in this file apply to all nodes in the cluster.

The default version of the `cluster.json` file assumes that the cluster management service is enabled:

```
{
  "instanceId" : "&{openidm.node.id}",
  "instanceTimeout" : 30000,
  "instanceRecoveryTimeout" : 30000,
  "instanceCheckInInterval" : 5000,
  "instanceCheckInOffset" : 0,
  "enabled" : true
}
```

instanceId

The ID of this node in the cluster. By default, this is set to the value of the instance's `openidm.node.id` that you set in the previous step.

instanceTimeout

The length of time (in milliseconds) that a member of the cluster can be "down" before the cluster manager considers that instance to be in *recovery mode*.

Recovery mode indicates that the `instanceTimeout` of an instance has expired, and that another instance in the cluster has detected that event. The scheduler component of the second instance then moves any incomplete jobs into the queue for the cluster.

instanceRecoveryTimeout

Specifies the time (in milliseconds) that an instance can be in recovery mode before it is considered to be offline.

This property sets a limit after which other members of the cluster stop trying to access an unavailable instance.

instanceCheckInInterval

Specifies the frequency (in milliseconds) that instances check in with the cluster manager to indicate that they are still online.

instanceCheckInOffset

Specifies an offset (in milliseconds) for the check-in timing, when multiple instances in a cluster are started simultaneously.

The check-in offset prevents multiple instances from checking in simultaneously, which would strain the cluster manager resource.

enabled

Specifies whether the cluster management service is enabled when you start the server. This property is set to `true` by default.

Important

If you disable the cluster manager while clustered nodes are running (by setting `"enabled" : false` in an instance's `cluster.json` file), the following happens:

- The cluster manager thread that causes instances to *check in* is not deactivated.
- Nodes in the cluster no longer receive cluster *events*, which are used to broadcast configuration changes when they occur over the REST interface.
- Nodes are unable to detect and attempt to recover failed instances within the cluster.
- Persisted schedules associated with failed instances cannot be recovered by other nodes.

5. Specify how the instance reads configuration changes. For more information, see "How IDM Instances Read Configuration Changes".
6. If you are using scheduled tasks, configure persistent schedules so that jobs and tasks are launched only once across the cluster.
7. Configure each node in the cluster to work with host headers. If you are using a load balancer, adjust the default `jetty.xml` configuration, as described in "Deploy Securely Behind a Load Balancer" in the *Security Guide*

8. Make sure that each node in the cluster has the same keystore and truststore. You can do this in one of the following ways:
 - When the first instance has been started, copy the initialized keystore (`/path/to/openidm/security/keystore.jceks`) and truststore (`/path/to/openidm/security/truststore`) to all other instances in the cluster.
 - Use a single keystore that is shared between all the nodes. The shared keystore might be on a mounted filesystem, a Hardware Security Module (HSM) or something similar. If you use this method, set the following properties in the `resolver/boot.properties` file of each instance to point to the shared keystore:

```
openidm.keystore.location=path/to/keystore
openidm.truststore.location=path/to/truststore
```

For information on configuring IDM to use an HSM device, see "Configuring IDM For a Hardware Security Module (HSM) Device" in the *Security Guide*.

- The configuration file `secrets.json` in the `/path/to/openidm/conf` directory must be the same on all the nodes.
9. Start each instance in the cluster.

Important

The audit service logs configuration changes only on the modified instance. Although configuration changes are persisted in the repository, and replicated on other instances by default, those changes are not logged separately for each instance.

Configuration changes are persisted by default, but changes to workflows and scripts, and extensions to the UI are not. Any changes that you make in these areas must be manually copied to each node in the cluster.

How IDM Instances Read Configuration Changes

IDM can read its configuration from the following locations:

- *Repository*. Each instance reads its configuration from the `configobjects` and `configobjectproperties` tables in a JDBC repository, or from the `ou=config,dc=openidm,dc=forgerock,dc=com` baseDN in a DS repository.
- *Filesystem*. Each instance reads its configuration from the JSON files under its `conf` directory and stores the configuration locally in memory.

In a clustered deployment, file-based configuration changes must be applied manually across all instances.

- *Memory*. The configuration can diverge if an instance is cut from its cluster due to a networking issue or a misconfigured load balancer. In this case, configuration changes made in the repository might not be detected and the configuration in memory will not be updated.

There are two properties in the `conf/system.properties` file that determine how configuration changes are handled for each instance:

openidm.config.repo.enabled

When this property is set to `true`, the instance reads configuration changes from the repository.

The default setting (`# openidm.config.repo.enabled=false`) indicates that the parameter is true. Uncomment that line to prevent the instance from reading configuration changes from the repository.

openidm.fileinstall.enabled

When this property is set to `true`, the instance reads its configuration from the files in its `conf/` directory.

The default setting (`# openidm.fileinstall.enabled=false`) indicates that the parameter is true. Uncomment that line to prevent the instance from reading file-based configuration changes.

Important

Every node in the cluster must have the identical configuration setting. For example, if you set `openidm.config.repo.enabled=true`, `openidm.fileinstall.enabled=false` on one node, you must set exactly the same options on every node in the cluster.

Repository-Based Configuration

Traditional clustered deployments share a *mutable* configuration that is read from a shared repository. The repository initially loads the configuration from the JSON files in the `conf` directory of the first instance that is configured in the cluster. However configuration changes are made, they are written to the repository, and the repository is the authoritative configuration source.

Therefore, a traditional clustered deployment generally has the following configuration:

```
openidm.config.repo.enabled=true
openidm.fileinstall.enabled=false
```

File-Based Configuration

A file-based configuration enables you to store the system configuration in a version-controlled filesystem, and to push a new version out to all nodes when the configuration changes. Using a file-based configuration therefore makes versioning and rolling out new configuration easier than pushing new configuration out over REST.

Container deployments typically require an *immutable* configuration that is read from a file system (such as a Docker image) and stored in memory. The file system is the authoritative configuration source and configuration changes are *not* written to the repository.

Therefore, a container deployment generally has the following configuration:

```
openidm.config.repo.enabled=false  
openidm.fileinstall.enabled=true
```

If file-based configuration is used, you *must* ensure that the configuration across instances remains consistent. Because the file-based configuration is not shared between instances, changes made to one node's configuration must be applied manually to all nodes across the cluster.

By default, IDM polls JSON configuration files in each `conf/` directory for changes. For security reasons, it is generally recommended that you disable automatic polling of configuration files to prevent untested configuration changes from disrupting your identity service.

For information on this parameter, see "Disabling Automatic Configuration Updates" in the *Security Guide*.

Scheduled Tasks Across a Cluster

In a clustered environment, the scheduler service looks for pending jobs and handles them as follows:

- Non-persistent (in-memory) jobs execute only on the node that created it.
- Persistent scheduled jobs are picked up and executed by any available node in the cluster that has been configured to execute persistent jobs.
- Jobs that are configured as persistent but *not concurrent* run on only one instance in the cluster at a time. That job will not run again at the scheduled time, on any instance in the cluster, until the current job is complete.

For example, a reconciliation operation that runs for longer than the time between scheduled intervals will not trigger a duplicate job while it is still running.

In clustered environments, the scheduler service obtains an `instanceID`, and check-in and timeout settings from the cluster management service (defined in the `project-dir/conf/cluster.json` file).

IDM instances in a cluster claim jobs in a random order. If one instance fails, the cluster manager automatically reassigns unstarted jobs that were claimed by that failed instance.

For example, if instance A claims a job but does not start it, and then loses connectivity, instance B can claim that job. If instance A claims a job, starts it, and then loses connectivity, other instances in the cluster cannot claim that job. If the failed instance does not complete the task, the next action depends on the `misfire policy`, defined in the scheduler configuration.

You can override this behavior with an external load balancer.

If a liveSync operation leads to multiple changes, a single instance processes all changes related to that operation.

Because all nodes in a cluster read their configuration from a single repository, you must use an instance's `resolver/boot.properties` file to define a specific scheduler configuration for that instance. Settings in the `boot.properties` file are not persisted in the repository, so you can use this file to set different values for a property across different nodes in the cluster.

For example, if your deployment has a four-node cluster and you want only two of those nodes to execute persisted schedules, you can disable persisted schedules in the `boot.properties` files of the remaining two nodes. If you set these values directly in the `scheduler.json` file, the values are persisted to the repository and are therefore applied to all nodes in the cluster.

By default, instances in a cluster are able to execute persistent schedules. The setting in the `boot.properties` file that governs this behaviour is:

```
openidm.scheduler.execute.persistent.schedules=true
```

To prevent a specific instance from claiming pending jobs, or processing clustered schedules, set `openidm.scheduler.execute.persistent.schedules=false` in the `boot.properties` file of that instance.

Caution

Changing the value of the `openidm.scheduler.execute.persistent.schedules` property in the `boot.properties` file changes the scheduler that manages scheduled tasks on that node. Because the persistent and in-memory schedulers are managed separately, a situation can arise where two separate schedules have the same schedule name.

For more information about persistent schedules, see "Configure Persistent Schedules" in the *Schedules Guide*.

Manage Nodes in a Cluster

You can managed clusters and nodes over the REST interface, or using the Admin UI.

Manage Nodes Over REST

You can manage clusters and individual nodes over the REST interface, at the endpoint `openidm/cluster/`. These sample REST commands demonstrate the cluster information that is available over REST:

+ *Display the Nodes in the Cluster*

The following REST request displays the nodes configured in the cluster, and their status.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/cluster?_queryFilter=true"
{
  "result": [
    {
      "_id": "node1",
      "state": "running",
      "instanceId": "node1",
      "startup": "2017-09-16T15:37:04.757Z",
      "shutdown": ""
    },
    {
      "_id": "node2",
      "state": "running",
      "instanceId": "node2",
      "startup": "2017-09-16T15:45:05.652Z",
      "shutdown": ""
    }
  ]
}
```

+ Check Node Status

To check the status of a specific node, include its node ID in the URL, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/cluster/node1"
{
  "_id": "node1",
  "instanceId": "node1",
  "startup": "2017-09-16T15:37:04.757Z",
  "shutdown": "",
  "state": "running"
}
```

Manage Nodes Using the Admin UI

The Admin UI provides a status widget that enables you to monitor the activity and status of all nodes in a cluster.

To add the widget to a Dashboard, click Add Widget then scroll down to System Status > Cluster Node Status and click Add.

The cluster node status widget shows the current status and number of running jobs of each node.

Select Status to obtain more information on the latest startup and shutdown times of that node. Select Jobs to obtain detailed information on the tasks that the node is running.

The widget can be managed in the same way as any other dashboard widget. For more information, see "Manage Dashboards" in the *Setup Guide*.

Remove Nodes

IDM automatically addresses cluster node removal. Shut down the instance to remove, and IDM no longer utilizes the node.

Considerations when removing an existing node from the cluster:

1. Ensure a clustered reconciliation is not taking place when removing the node as this could cause unexpected behavior.
2. Remnants of the node will still exist in the chosen IDM database. For example, you may see the removed node still appearing in the cluster node status widget in the Dashboard (for more context on this, refer to [Manage nodes using the admin UI](#)).

+ Example using DS

1. Shut down all instances of IDM.
2. Delete the old instance record from DS:
 - Locate the DN of the node using the `ldapsearch` command:

```
./ldapsearch -D "uid=admin" -w password -h localhost -p 1389 -b "dc=openidm,dc=forgerock,dc=com" "uid=node1"
```

```
dn: uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com
objectClass: uidObject
objectClass: fr-idm-generic-obj
objectClass: top
fr-idm-json:
{"recoveryAttempts":1,"detectedDown":"0000001660588116038","type":"state","recoveryFinished":"0000001660588116038"}
uid: node1
```

Note

The name of the node instance can be found in the `openidm/resolver/boot.properties` file by the value of the `openidm.node.id` attribute.

- Delete the node record:

```
./ldapdelete -D "uid=admin" -w password -h localhost -p 1389 -b "dc=openidm,dc=forgerock,dc=com"
"uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com"

# DELETE operation successful for DN
uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com
```

3. Start all operational IDM nodes.

+ Example using Oracle database

1. Shut down all instances of IDM.
2. Delete references to the old instance from `clusterobjects` and the associated rows from `clusterobjectproperties` in the database:

- Locate the name of the node, in this case `node1`:

```
SELECT * FROM clusterobjects;
```

id	objecttypes_id	objectid	rev	fullobject
1	2	node1	43789	{"redacted"}

- Delete the references of the old node from the `clusterobjects` table:

```
DELETE FROM openidm.clusterobjects WHERE objectid = 'node1';
```

clusterobjects_id	propkey	proptype	propvalue
1	/recoveryAttempts	java.lang.Integer	0
1	/_rev	java.lang.String	43790
1	/detectedDown	java.lang.String	000000

- Delete references of the old node from the `clusterobjectproperties` table:

```
DELETE FROM openidm.clusterobjectproperties WHERE clusterobjects_id = 'node1';
```

- Start all operational IDM nodes.

Appendix A. Host and Port Information

To change the default IDM hostname or listening ports, edit the applicable entry in `openidm/resolver/boot.properties`:

```
openidm.port.http=8080
openidm.port.https=8443
openidm.port.mutualauth=8444
openidm.host=localhost

openidm.auth.clientauthonlyports=8444
```

8080

HTTP access to the REST API, requiring IDM authentication. This port is not secure, exposing clear text passwords and all data that is not encrypted. This port is therefore not suitable for production use.

8443

HTTPS access to the REST API, requiring IDM authentication

8444

HTTPS access to the REST API, requiring SSL mutual authentication. Clients that present certificates found in the truststore (`openidm/security/`) are granted access to the system.

Appendix B. Property Files

This section lists the `*.properties` files used to configure IDM. Apart from the `boot.properties` file, these files are located in your project's `conf/` directory. This section does not include the `*.properties` files associated with ICF connectors.

Important

After making changes to any `*.properties` file, you must restart IDM for the changes to take effect.

+ `boot.properties`

The `boot.properties` file is the property resolver file, used for property substitution, and is located in the `/path/to/openidm/resolver` directory. Generally, this file lets you set variables that are used in other configuration files, including `config.properties` and `system.properties`.

+ `config.properties`

The `config.properties` file is used for two purposes:

- To set OSGi bundle properties.
- To set Apache Felix properties, and plugin bundles related to the Felix web console.

For more information about each item in `config.properties`, see the following documentation: *Apache Felix Framework Configuration Properties*.

+ `logging.properties`

The `logging.properties` file configures JDK logging for IDM.

+ `system.properties`

The `system.properties` file is used to bootstrap java system properties such as:

- Jetty log settings, based on the Jetty container bundled with IDM. IDM bundles Jetty version 9.4.22.
- Cluster configuration.
- Quartz updates, as described in *Quartz Best Practices* documentation.
`org.terracotta.quartz`
- A common transaction ID, as described in "Configure the Audit Service" in the *Audit Guide*.

Appendix C. Embedded Jetty Configuration

IDM includes an embedded Jetty web server.

To configure the embedded Jetty server, edit `openidm/conf/jetty.xml`. IDM delegates most of the connector configuration to `jetty.xml`. OSGi and PAX web specific settings for connector configuration therefore do not have an effect. This lets you take advantage of all Jetty capabilities, as the web server is not configured through an abstraction that might limit some of the options.

The Jetty configuration can reference configuration properties (such as port numbers and keystore details) from your `resolver/boot.properties` file.

- "IDM Configuration Properties in Jetty"
- "Jetty Default Settings"
- "Register Additional Servlet Filters"
- "Enable and Disable Secure Protocols and Cipher Suites"
- "Adjust Jetty Thread Settings"
- "Gzip Compression for HTTP Responses"

IDM Configuration Properties in Jetty

IDM exposes a `Param` class that you can use in `jetty.xml` to include IDM-specific configuration. The `Param` class exposes Bean properties for common Jetty settings and generic property access for other, arbitrary settings.

Explicit Bean Properties

To retrieve an explicit Bean property, use the following syntax in `jetty.xml`:

```
<Get class="org.forgerock.openidm.jetty.Param" name="<bean property name>"/>
```

For example, to set a Jetty property for keystore password:

```
<Set name="password">  
  <Get class="org.forgerock.openidm.jetty.Param" name="keystorePassword"/>  
</Set>
```

Also see the bundled `jetty.xml` for further examples.

The following explicit Bean properties are available; they map either to the `boot.properties` in the `openidm/resolver/` subdirectory, or the `secrets.json` file in your project's `conf/` subdirectory.

port

Maps to `openidm.port.http`

port

Maps to `openidm.port.https`

port

Maps to `openidm.port.mutualauth`

keystoreType

Maps to `mainKeyStore storeType`

keystoreProvider

Maps to `mainKeyStore providerName`

keystoreLocation

Maps to `mainKeyStore file`

keystorePassword

Maps to `mainKeyStore storePassword`

truststoreLocation

Maps to `mainTrustStore file`

truststorePassword

Maps to `mainTrustStore storePassword`

Generic Properties

```
<Call class="org.forgerock.openidm.jetty.Param" name="getProperty">  
  <Arg>org.forgerock.openidm.some.sample.property</Arg>  
</Call>
```

Jetty Default Settings

By default the embedded Jetty server uses the following settings.

- The HTTP, SSL, and Mutual Authentication ports defined in IDM
- The same keystore and truststore settings as IDM
- Trivial sample realm, `openidm/security/realm.properties` to add users

The default settings are intended for evaluation only. Adjust them according to your production requirements.

Register Additional Servlet Filters

You can register generic servlet filters in the embedded Jetty server to perform additional filtering tasks on requests to or responses from IDM. For example, you might want to use a servlet filter to protect access to IDM with an access management product. Servlet filters are configured in files named `openidm/conf/servletfilter-name.json`. These servlet filter configuration files define the filter class, required libraries, and other settings.

A sample servlet filter configuration is provided in the `servletfilter-cors.json` file in the `/path/to/openidm/conf` directory.

The sample servlet filter configuration file is shown below:

```
{
  "classPathURLs" : [ ],
  "systemProperties" : { },
  "requestAttributes" : { },
  "scriptExtensions" : { }.
  "initParams" : {
    "allowedOrigins" : "https://localhost:&{openidm.port.https}",
    "allowedMethods" : "GET,POST,PUT,DELETE,PATCH",
    "allowedHeaders" : "accept,x-openidm-password,x-openidm-noseession,
      x-openidm-username,content-type,origin,
      x-requested-with",
    "allowCredentials" : true,
    "chainPreflight" : false
  },
  "urlPatterns" : [
    "/"*
  ],
  "filterClass" : "org.eclipse.jetty.servlets.CrossOriginFilter"
}
```

The sample configuration includes the following properties:

classPathURLs

The URLs to any required classes or libraries that should be added to the classpath used by the servlet filter class

systemProperties

Any additional Java system properties required by the filter

requestAttributes

The HTTP Servlet request attributes that will be set when the filter is invoked. IDM expects certain request attributes to be set by any module that protects access to it, so this helps in setting these expected settings.

scriptExtensions

Optional script extensions to IDM. Currently only `augmentSecurityContext` is supported. A script that is defined in `augmentSecurityContext` is executed after a successful authentication request. The script helps to populate the expected security context. For example, the login module (servlet filter) might select to supply only the authenticated user name, while the associated roles and user ID can be augmented by the script.

Supported script types include `"text/javascript"` and `"groovy"`. The script can be provided inline (`"source":script source`) or in a file (`"file":filename`). The sample filter extends the filter interface with the functionality in the script `script/security/populateContext.js`.

filterClass

The servlet filter that is being registered

The following additional properties can be configured for the filter:

`httpContextId`

The HTTP context under which the filter should be registered. The default is `"openidm"`.

`servletNames`

A list of servlet names to which the filter should apply. The default is `"OpenIDM REST"`.

`urlPatterns`

A list of URL patterns to which the filter applies. The default is `["/*"]`.

`initParams`

Filter configuration initialization parameters that are passed to the servlet filter `init` method. For more information, see <http://docs.oracle.com/javaee/5/api/javax/servlet/FilterConfig.html>.

Enable and Disable Secure Protocols and Cipher Suites

The Jetty configuration for inbound connections to IDM supports a number of protocols and cipher suites.

Enabled *protocols* are explicitly listed in the `includedProtocols` list in the `conf/jetty.xml` file. Only `TLSv1.2` and `TLSv1.3` are enabled by default:

```
...
<Array id="includedProtocols" type="java.lang.String">
  <!-- Only support TLS v1.2 and v1.3 -->
  <Item>TLSv1.2</Item>
  <Item>TLSv1.3</Item>
</Array>
...
```

To disable a particular protocol, remove it from the `includedProtocols` list. To add support for a weaker protocol, add it to the list, for example:

```
...
<Array id="includedProtocols" type="java.lang.String">
  <Item>TLSv1.2</Item>
  <Item>TLSv1.3</Item>
  <Item>SSLv3.0</Item>
</Array>
...
```

Important

It is highly recommended that you do not enable weaker protocols such as SSL, and TLS versions prior to 1.2. These protocols use outdated algorithms and are generally considered insecure.

Enabled *cipher suites* for each protocol are listed in the `includedCipherSuites` list in `conf/jetty.xml`:

```

...
<Array id="includedCipherSuites" type="java.lang.String">
  <!-- TLS 1.3 cipher suites -->
  <Item>TLS_AES_128_GCM_SHA256</Item>
  <Item>TLS_AES_256_GCM_SHA384</Item>
  <Item>TLS_CHACHA20_POLY1305_SHA256</Item>

  <!-- TLS 1.2 cipher suites -->
  <Item>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</Item>
  <Item>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</Item>
  <Item>TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256</Item>
  <Item>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</Item>
  <Item>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</Item>
  <Item>TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256</Item>
  <Item>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</Item>
  <Item>TLS_DHE_RSA_WITH_AES_128_GCM_SHA256</Item>
</Array>
...

```

To add support for additional cipher suites, add them as `<Item>`s in this list.

Adjust Jetty Thread Settings

To change the Jetty thread pool settings, add the following excerpt to your project's `conf/config.properties` file:

```

# Jetty maxThreads (default 200)
org.ops4j.pax.web.server.maxThreads=${org.ops4j.pax.web.server.maxThreads}
# Jetty minThreads (default 8)
org.ops4j.pax.web.server.minThreads=${org.ops4j.pax.web.server.minThreads}
# Jetty idle-thread timeout milliseconds (default 60000)
org.ops4j.pax.web.server.idleTimeout=${org.ops4j.pax.web.server.idleTimeout}

```

To override these defaults, set a corresponding `OPENIDM_OPTS` variable when you start IDM. For example:

```

export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dorg.ops4j.pax.web.server.maxThreads=768"
/path/to/openidm/startup.sh
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m -Dorg.ops4j.pax.web.server.maxThreads=768
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
-> OpenIDM version "7.0.4" (revision: unknown)
OpenIDM ready

```

Important

You cannot use property substitution to set these properties.

You cannot adjust Jetty's thread settings in the `conf/jetty.xml` file. If you uncomment the excerpt of `jetty.xml` that starts with `<!--<Arg name="threadpool">...`, you'll see errors in the IDM log.

Gzip Compression for HTTP Responses

IDM uses the Jetty Gzip handler to compress HTTP responses. The default Gzip handler configuration, in `conf/jetty.xml`, is as follows:

```
...
  <Call name="insertHandler">
    <Arg>
      <!-- https://www.eclipse.org/jetty/documentation/9.4.x/gzip-filter.html -->
      <New id="GzipHandler" class="org.eclipse.jetty.server.handler.gzip.GzipHandler">
        <Set name="minGzipSize"><Property name="jetty.gzip.minGzipSize" default="2048"/></Set>
        <Set name="compressionLevel"><Property name="jetty.gzip.compressionLevel" default="-1"/></
Set>
        <Set name="inflateBufferSize"><Property name="jetty.gzip.inflateBufferSize" default="0"/
></Set>
        <Set name="syncFlush"><Property name="jetty.gzip.syncFlush" default="false" /></Set>
        <Set name="excludedAgentPatterns">
          <Array type="String">
            <!-- IE 6 has known bugs related to GZIP compression -->
            <Item><Property name="jetty.gzip.excludedUserAgent" default=".*MSIE.6\.0.*"/></
Item>
          </Array>
        </Set>
        <Set name="includedMethodList"><Property name="jetty.gzip.includedMethodList" default="GET" /></Set>
        <Set name="excludedMethodList"><Property name="jetty.gzip.excludedMethodList" default="" /
></Set>
      </New>
    </Arg>
  </Call>
...
```

Adjust this configuration if the default does not suit your deployment. Configuration properties are as follows:

`minGzipSize`

Content is compressed only if the content length is unknown or is greater than the `minGzipSize`. By default, content is compressed only if the response is greater than 2048MB.

`compressionLevel`

The compression level (1-9, with 1 being the fastest compression speed but a lower compression ratio, and 9 being the highest compression ratio but lowest compression speed. The default configuration sets the compression level to -1, which indicates that the application should use the default. The default Gzip handler uses level 6, favoring higher compression over speed.

inflateBufferSize

Number of bytes in the request decompression buffer. The default setting is `-1`, which disables this feature. Use this feature only if you want to compress large POST/PUT request payloads. Be aware that this setting exposes a potential Zip bomb risk.

syncFlush

By default, this setting is `false`, which lets the deflater determine how much data to accumulate, before it produces output. This achieves the best compression. When `true`, this setting forces flushing of the buffer of data to compress. This can result in poor compression.

excludedAgentPatterns

A list of regex patterns for User-Agent names. Requests from these names should not be compressed.

includedMethodList

A list of HTTP methods to compress. By default, only GET requests are compressed.

excludedMethodList

A list of HTTP methods that should not be compressed.

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see " <i>Correlating Source Objects With Existing Target Objects</i> " in the <i>Synchronization Guide</i> .
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a correlation query , a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of assignment . A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo , "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the <code>JWT_SESSION</code> authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the <i>Object Modeling Guide</i> .
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object

A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.

target object

In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.