Installing and Configuring Akula

Version 2.5.0
Installing and Configuring Akula

© Copyright Verivo Software, Inc., 2014. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language by any means without the written permission of Verivo Software, Inc. (“Verivo”).

Your license agreement with Verivo, which is included with the product, specifies the permitted and prohibited uses of the product. Any unauthorized duplication or use of this software, in whole or in part, in print, or in any other storage and retrieval system is forbidden.

Neither you nor anyone acting on your behalf, including your employees, acquire any intellectual property or other proprietary rights, including patents, designs, trademarks, copyright or trade-secrets, relating to the contents of this document, including without limitation, software and information, except as otherwise expressly specified in an appropriate license or other mutually agreed upon, written agreement that you may have with Verivo. Any grants not expressly granted herein are reserved.

The trademarks below are trademarks of Verivo or its affiliates in the United States. Verivo may also have common law trademarks that are not listed below. Although we endeavor to keep current, the below list may not be comprehensive.

VERIVO® and AKULA™ and families of related marks, images and symbols are the exclusive properties and trademarks of Verivo. VERIVO® is registered, and AKULA™ is pending registration, with the U.S. Patent and Trademark Office and may be pending or registered in other countries. The BlackBerry, RIM and SureType word mark and logos are owned by Research in Motion Limited. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

Mention of third party companies and products is for informational purposes only and does not constitute an endorsement. Verivo assumes no responsibility with regard to the selection, performance, or use of these products. All understandings, agreements or warranties, if any, take place directly between the vendor and prospective users.

Verivo Software, Inc.
1000 Winter Street
Waltham, MA 02451
781.795.8200
www.verivo.com
## Contents

1. Installing and Configuring the Akula Server
   1.1 Installing the Akula Server ................................. 2
      1.1.1 Akula System Requirements .......................... 2
      1.1.2 Install the Akula Server ........................... 4
         1.1.2.1 Before You Install the Akula Server ....... 4
         1.1.2.2 Set up the Akula Server Environment ...... 4
         1.1.2.3 Specify the Database for the Akula Server .... 7
         1.1.2.4 Configure a JMS Server ........................ 11
         1.1.2.5 Deploy the Akula WAR file on your Java Server .... 14
         1.1.2.6 Install the Akula Command-line Management Utility 19
         1.1.2.7 Install an Akula License .......................... 23
         1.1.2.8 Validate the Akula Server Install ............ 28
      1.1.3 Uninstall the Akula Server ............................ 29
      1.1.4 Offline Installation of an Akula License ............. 30
   1.2 Configuring the Akula Server ................................ 32
      1.2.1 Define Akula Server Administrators ................... 32
      1.2.2 Allow Cross Origin Requests in a Browser App ....... 35
      1.2.3 Configure Logging ..................................... 41
      1.2.4 Using SSL with Akula .................................. 51
      1.2.5 Using the Akula Command-line Management Utility .... 53
   1.3 Upgrading the Akula Server ................................ 60

2. Installing and Using the Akula Management Console
   2.1 Installing the Akula Management Console .................... 62
   2.2 Using the Akula Management Console .......................... 65
      2.2.1 Viewing and Overriding App Properties ............... 67
      2.2.2 Viewing and Overriding Logging Levels .............. 69
      2.2.3 Manage Roles, Groups, and Permissions ............... 69
      2.2.4 Setting Credentials .................................... 71
      2.2.5 Setting Rules to Block Client Sync and Login ....... 71
      2.2.6 Configuring, Sending, and Viewing Push Notifications 72
      2.2.7 Creating and Viewing Actions .......................... 72
Installing and Configuring the Akula Server

Verivo Akula™ is the enterprise mobile app platform that helps businesses reduce the complexities associated with the development, deployment and on-going runtime operational activities of their enterprise mobile apps.

Akula offers centralized control and management of custom apps, along with a flexibility to utilize your company’s choice of tools, frameworks and technologies. Akula’s built-in security and management capabilities provide the level of control and oversight required to enforce even the most stringent corporate policies.

This section describes how to install and configure the Akula Server, and contains the following topics:

- Installing the Akula Server
- Configuring the Akula Server
- Upgrading the Akula Server

Installing the Akula Server

You can install the Akula Server on a Windows, Mac, or Unix system. This section contains the system requirements and installation instructions.

To complete the installation, perform the steps in all the sections, up to and including the section Validate the Akula Server Install.

This section contains the following topics:

- Akula System Requirements
- Install the Akula Server
- Uninstall the Akula Server
- Offline Installation of an Akula License

Akula System Requirements

This topic describes Akula's system requirements:

- Akula Server hosting requirements
- Mobile device requirements for running a client app

Akula Server hosting requirements

To deploy the Akula Server, your server environment must meet the following system requirements:

<table>
<thead>
<tr>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
</table>

© Copyright 2014, Verivo Software, Inc. All Rights Reserved.
### Operating System

- Windows Server 2008 R2 64-bit
- SUSE Enterprise Linux Server 11
- Red Hat Enterprise Linux 6.3

Note: Additional Operating Systems and versions should work as long as the appropriate Java JRE or JDK is available. For example, most recent variations of Linux including Red Hat Enterprise Linux 6.3, though not tested, should run Akula properly. Akula was successfully run in the following environments: Mac OS X 10.8.x, Ubuntu 12.x, and Windows 7 SP1.

### Java SDK

- OpenJDK 1.7 [http://openjdk.java.net/install/](http://openjdk.java.net/install/)

### JEE Server or Servlet Container

- JBoss EAP Version 6.1 from [http://www.jboss.org/overview/](http://www.jboss.org/overview/)
- Apache Tomcat 7 from [http://tomcat.apache.org/download-70.cgi](http://tomcat.apache.org/download-70.cgi)

### Internal database

- SQL Server 2008, 2012
- MySQL 5 InnoDB
- Oracle 11g

Note: This is the internal database used by the Akula Server to store configuration and run time information used by the server. For more information, see Specify the Database for the Akula Server.

### RAM

- 4 GB

### RAM dedicated to the JVM

- 2 GB

### PermGen size

- 500 MB

### Disk space

- 500 MB

---

**Mobile device requirements for running a client app**

To run an Akula client app on a device, the device must meet the requirements listed below.

### Android device requirements

Android devices must run Android v2.3.3 or later.

### iOS device requirements

Devices running iOS must run iOS versions 5.x - 7.0.
JavaScript

Mobile Browsers
- Android native browser on supported versions of Android (CORS supported on v3.0+)
- iOS native browser on supported versions of iOS
- Windows Mobile 7.5+, 8.0+
- Blackberry OS 6+, 7+, 10+ browsers (CORS not supported in OS 6 and 7)

Desktop Browsers
- Internet Explorer versions 8, 9, 10 (CORS is not supported in IE 8 and 9)
- Chrome 24+
- Firefox v17+
- Safari v5.1.0+, v6.0+ on OS X

Cordova

Apache Cordova 3.0.0 from [http://cordova.apache.org/](http://cordova.apache.org/)

Install the Akula Server

Follow the steps in this section to install the Akula Server and to validate that you installed the Akula Server correctly.

This section describes the following topics:
- Before You Install the Akula Server
- Set up the Akula Server Environment
- Specify the Database for the Akula Server
- Configure a JMS Server
- Deploy the Akula WAR file on your Java Server
- Install the Akula Command-line Management Utility
- Install an Akula License
- Validate the Akula Server Install

Before You Install the Akula Server

Before you begin the installation process, you should be aware of the licensing requirements and the default security realm installed with the Akula Server.

About licensing

You will need an Akula license to perform most actions with the Akula Server, including accessing backend data. If you do not currently have an Akula Server license, you can [contact us](mailto:contactus) for information on obtaining one.

About the default security realm

The Akula security mechanism provides user authentication to validate a user's credentials, and role-based authorization to control the user's access to the app and to app functionality. Authentication information is stored in a security realm maintained outside of the Akula Server. For example, you can use Active Directory as your realm.

To get you up and running quickly in a development environment, the Akula Server provides a default security realm.
implemented as a SQLite database in the AKULA_HOME/global/sqlite_directory.db file. This realm contains definitions for two users and for two user groups. After you install the Akula Server, you can use the Akula Command-line Management Utility to log in to the server as the Administrator user to configure the Akula Server, including installing the Akula license.

The default SQLite database stores passwords in clear text and is not meant for production. It is provided to get you up and running quickly in a development environment and should be deleted as part of moving to a production environment.

The following table lists the users, and associated credentials, defined in the SQLite realm:

<table>
<thead>
<tr>
<th>User</th>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>admin</td>
<td>admin</td>
</tr>
<tr>
<td>User</td>
<td>user</td>
<td>user</td>
</tr>
</tbody>
</table>

Each user is assigned to a user group, as the following table shows:

<table>
<thead>
<tr>
<th>Group name</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrators</td>
<td>admin</td>
</tr>
<tr>
<td>Users</td>
<td>admin, user</td>
</tr>
</tbody>
</table>

Set up the Akula Server Environment

The following sections contain instructions on how to set up the Akula Server environment:

- Set the Java memory requirements for the Akula Server
- Set the AKULA_HOME environment variable
- Set up the Akula Server directories
- Copy the Akula Server configuration files

Set the Java memory requirements for the Akula Server

The Akula Server requires a maximum of 512 MBytes of heap space and a maximum of 128 MBytes of PermGen space. You can define this memory requirements by setting the JAVA_OPTS environment variable as shown below:

- **Microsoft Windows:** JAVA_OPTS=-Xmx512m -XX:MaxPermSize=128m
- **UNIX/Mac:** JAVA_OPTS="-Xmx512m -XX:MaxPermSize=128m"

Set the AKULA_HOME environment variable

The AKULA_HOME environment variable specifies where the Akula Server looks for configuration files and for deployed app scopes. You **must** set the value of the AKULA_HOME environment variable before you start the Akula Server.
When setting the AKULA_HOME environment variable:

- A common setting for the AKULA_HOME environment variable is your user folder, meaning the user name of the person who starts the JEE server hosting the Akula Server. For example, set it to \Users\USERNAME\Akula\home.

- Always use forward slashes, "/", as the path separator when setting AKULA_HOME, even on Microsoft Windows.

- Make sure that you set AKULA_HOME as a system variable, not as a user variable.

On Windows, use Computer > Properties > Advanced system setting to set the system environment variable as shown below:

AKULA_HOME = c:/Users/nickdanger/Akula/home

On a Mac or UNIX system, you can set the AKULA_HOME environment variable in your .profile file by adding the following command:

export AKULA_HOME=/Users/nickdanger/Akula/home

Note that the .profile file is a hidden file. Use the "ls -la" command to view hidden files. If you have not created a .profile file previously, you can create one.

Alternatively, set the value of AKULA_HOME as a JVM property in your startup script or in your JVM configuration file (jvm.cfg). For example:

-DAKULA_HOME=c:/Users/nickdanger/Akula/home

Restart your Java server after defining this environment variable.

Set up the Akula Server directories

If you are on a Mac or Unix machine, make sure that the Akula Server directories are writable by the user account used to start your Java server.

To set up the Akula Server directories:

1. Under the AKULA_HOME folder, create the following directories:
   a. /apps - Where the Akula Server expands an AKZ file corresponding to an app scope.
   b. /db - The location of the default Akula configuration database.
   c. /deploy - Where you write an AKZ file corresponding to an app scope.
   d. /global - The location of Akula configuration files.
   e. /licenses - The location of Akula license files.
   f. /logs - The default location of Akula log files.

Copy the Akula Server configuration files

The Akula Server requires several configuration files that you must install before starting the server. To get you up and running quickly, Akula ships with a set of default configuration files:
To copy these files:

1. Browse to the location where you unzipped the Akula Server ZIP file. If you have not done so already, download the Akula Server from the Downloads (you must log in to access this page).
2. From the server-sdk/samples/akula-home-template/global dir, copy these files to the AKULA_HOME/global folder.

Where to go next

Now that you have set up the Akula Server environment, go to Specify the Database for the Akula Server.

Specify the Database for the Akula Server

The Akula Server requires a database to store the configuration and run time information used by the server. The Akula Server includes an internal Apache Derby database that you can use to get up and running quickly in a development environment.

While the Apache Derby database is adequate for development work, it is not intended to function in a deployment environment. For deployment, Verivo recommends that you configure the Akula Server to use an external enterprise-class database, such as Microsoft SQL Server, MySQL, or Oracle.

In your deployment environment, you might want multiple Akula Servers running on different machines to share configuration and run time information. Therefore, you can configure multiple Akula Servers to reference a single, shared database.

This page contains the following sections:

- **Supported Databases**
- **Using the Apache Derby Database**
- **When to Use an External Database**
  - Step 1: Create the external database for use by the Akula Server
  - Step 2: Configure an Akula Server to reference the external database
  - Step 3: Start akula.war
- **Using the SQL Server 2012 Database**
  - JDBC Driver and Maven
  - Hibernate Dialect
  - JNDI Configuration
• Sharing the Database across multiple Akula Servers
  • Where to go next

**Supported Databases**

The Akula Server supports connections to JDBC-compliant databases. Databases that have been tested with the Akula Server include:

- Apache Derby
- SQL Server 2008 and 2012
- MySQL 5 InnoDB
- Oracle 11g

However, any database that supports JDBC should work with the Akula Server.

> The Akula Server uses "Hibernate" to communicate with the database. Therefore, your database should have a corresponding Hibernate dialect class. For examples of Hibernate dialect classes, see [http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/dialect/class-use/Dialect.html#org.hibernate.dialect](http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/dialect/class-use/Dialect.html#org.hibernate.dialect).

**Using the Apache Derby Database**

To use the internal Apache Derby database:

1. Stop the Akula Server if it is running.
2. Copy the following settings into the `AKULA_HOME/global/context.properties` file, overwriting all other settings:

```property
# The Derby database uses a JDBC connection.
db_mode = jdbc

# Specify the name of the database.
db_schema = netderby

# This JDBC configuration instructs Akula to use a local, embedded Derby database
db_jdbc_driver = org.apache.derby.jdbc.ClientDriver
db_jdbc_url = jdbc:derby://localhost:${derby_port}/${AKULA_HOME}/db/netderby;create=true
db_jdbc_username = sa
db_jdbc_password = sa

# Hibernate dialect.
hibernate_dialect = org.hibernate.dialect.DerbyTenSevenDialect

# Build database schema if empty automatically.
# Do not enable this property other than for an embedded Derby database.
db_init_if_empty = true

# Embedded Derby properties, used in development mode.
derby_start_server = true
derby_port = 1527
```

3. If reconfiguring the database for an existing Akula installation, restart the Akula Server. Otherwise, go to Configure a JMS Server.

**When to Use an External Database**

For deployment, Verivo recommends that you configure the Akula Server to use an external enterprise-class
Installing and Configuring Akula

database, such as Microsoft SQL Server, MySQL, or Oracle. This section describes how to configure the Akula Server to access the database.

Configuring an external database is a three step procedure:

1. Create the external database on Microsoft SQL Server, MySQL, or Oracle.
2. Configure an Akula Server to reference the external database.
3. If reconfiguring the database for an existing Akula installation, restart the Akula Server. Otherwise, go to Configure a JMS Server.

**Step 1: Create the external database for use by the Akula Server**

Before an Akula Server can connect to an external database, the database must exist.

To create the database:

1. Create a new database for use by the Akula Server. For example, create a database named akuladb.

   You do not have to create any tables in the database. On start up, the akula.war file automatically initializes the database with the necessary tables and table schemas.

**Step 2: Configure an Akula Server to reference the external database**

Edit the AKULA_HOME/global/context.properties file to configure the Akula Server to reference an external database.

The context.properties contains the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_mode</td>
<td>Set to jndi for the external database.</td>
</tr>
<tr>
<td>db_jndi_name</td>
<td>Specifies a JNDI database resource on your JEE server or servlet container that defines the connection information to your database. For example: java:comp/env/jdbc/akula-db</td>
</tr>
<tr>
<td>db_schema</td>
<td>Specifies the database name.</td>
</tr>
<tr>
<td>hibernate_dialect</td>
<td>Specifies the Hibernate dialect class that corresponds to your database.</td>
</tr>
<tr>
<td></td>
<td>For example, if you are using MySQL, the corresponding Hibernate dialect class is org.hibernate.dialect.MySQLDialect.</td>
</tr>
</tbody>
</table>

After editing the context.properties file, you must restart the Akula server.

**Step 3: Start akula.war**
Installing and Configuring Akula

Starting `akula.war` on your JEE server or servlet container initializes the database. If this is the first time the Akula Server has connected to the database, `akula.war` automatically initializes the database with the necessary tables and table schemas.

Using the SQL Server 2012 Database

**JDBC Driver and Maven**

This database was tested with Akula using a jTDS open source SQL Server JDBC driver. While there is an "official" driver available from Microsoft, the jTDS driver appeared to have larger community support and may have better overall JDBC compliance.

This driver is available on Maven:

```xml
<dependency>
    <groupId>net.sourceforge.jtds</groupId>
    <artifactId>jtds</artifactId>
    <version>1.3.0</version>
</dependency>
```

**Hibernate Dialect**

Set the dialect in context.properties:

```properties
# Hibernate properties
hibernate_dialect = org.hibernate.dialect.SQLServer2008Dialect
```

Note that `org.hibernate.dialect.SQLServer2008Dialect` is used in the snippet. A non-versioned "SQLServerDialect" dialect was tried, but produced errors.

**JNDI Configuration**

Ensure that the username specified in the JNDI database resource definition has permissions to the database so that the `akula.war` can modify the database tables and table schemas. For more information, see the example below.

**Example context.properties file**

Shown below is an example context.properties file:
# Use a JNDI connection to connect to your production database.

db_mode = jndi

# Akula DB connection

db_jndi_name = java:comp/env/jdbc/akula-db

# Hibernate dialect property.

hibernate_dialect = org.hibernate.dialect.MySQLDialect

# Specify the database name.

db_schema = akuladb

In this example, you connect the Akula server to a MySQL database in a production environment. Therefore, the hibernate.dialect property references the org.hibernate.dialect.MySQLDialect class.

The akula.war file automatically initializes its database when you first start it. Therefore, ensure that the username specified in the JNDI database resource definition has permissions to the database so that the akula.war file can modify the database tables and table schemas.

The db_jndi_name property is set to java:comp/env/jdbc/akula-db to specify the JNDI database resource. You therefore require a corresponding JNDI database resource definition on your JEE server or servlet container named java:comp/env/jdbc/akula-db.

For example, if you are running the Akula Server on Tomcat, you would add the following resource definition to the Tomcat context.xml file:

```xml
<Resource
   name="jdbc/akula-db"
   auth="Container"
   type="javax.sql.DataSource"
   maxActive="100" maxIdle="30" maxWait="10000"
   username="javauser" password="javaadmin"
   driverClassName="com.mysql.jdbc.Driver"
   url="jdbc:mysql://localhost:3306/akuladb"
/>
```

If you are not using Tomcat, define the JNDI resource as appropriate for your JEE server or servlet container.

**Sharing the Database across multiple Akula Servers**

In a deployment environment, you deploy a single Akula Server on a host computer. To share the configuration database across multiple Akula Servers, you only need to make sure that the JNDI resource definition for each JEE server or servlet container references the same database.

**Where to go next**

Now that you have set up the Akula Server database, go to Configure a JMS Server.
Configure a JMS Server

The ability to use push notifications with a JMS server is a built-in feature of the Akula Server. You must specify resource definitions for a JMS server even if you do not use push notification in your apps. If resource definitions are not provided, the Akula Server will not start.

The location and structure of the JMS resource definitions are container-specific. For example, if you use Tomcat as your JEE container, then you define the JMS resources in the context.xml file. For JBoss, you define the resources in the /standalone/configuration/standalone.xml file. The values of the resources are specific to your JMS implementation. For more information, consult your JMS server's documentation.

If using a WebLogic server to send notifications to Android devices, you must also configure a custom host name identifier on that server.

This document contains the following sections:

- Define resources for your JMS server
- Configure a custom host name identifier (WebLogic only)

Define resources for your JMS server

To configure your JMS server with the Akula push service:

1. Stop the Akula Server if it is running.
2. Define the resources listed in the following table:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Physical Name</th>
<th>JNDI Name</th>
<th>web.xml Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS connection factory</td>
<td>N/A</td>
<td>jms/akula-push-cf</td>
<td>&lt;resource-ref&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;description&gt;Akula JMS connection factory.&lt;/description&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jms/akula-push-cf</td>
<td>&lt;res-ref-name&gt;jms/akula-push-cf&lt;/res-ref-name&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>java:jboss/factories/jms/akula-push-cf</td>
<td>&lt;res-type&gt;javax.jms.QueueConnectionFactory&lt;/res-type&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;res-auth&gt;Container&lt;/res-auth&gt;</td>
</tr>
</tbody>
</table>
### GCM JMS queue destination

| akula.push.gcm | jms/akula-push-queue-gcm |

For JBoss, use:

```xml
<resource-env-ref>
    <resource-env-ref-name>jms/akula-push-queue-gcm</resource-env-ref-name>
    <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

### APNS JMS queue destination

| akula.push.apns | jms/akula-push-queue-apns |

For JBoss, use:

```xml
<resource-env-ref>
    <resource-env-ref-name>jms/akula-push-queue-apns</resource-env-ref-name>
    <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

3. If installing Akula for the first time or completing an upgrade, see [4. Deploy the Akula WAR File on Your Java Server](#). Otherwise, if changing the JMS configuration for an existing deployment, start the Akula Server.

For more information on sending push notifications to clients, see [Push Notifications](#).

**Sample resource definitions for a Tomcat server**

The resource definitions needed to use Akula with push notifications on a Tomcat server are added in the `/conf/context.xml` file. The following sample adds connection factory and push service reference definitions:
Configure a custom host name identifier (WebLogic only)

To use the Akula push service to send notifications to Android devices, specify the wildcarded host name verifier as a custom host name identifier using the WebLogic Administration Console. The name of the wildcarded host name verifier class is weblogic.security.utils.SSLWLSWildcardHostnameVerifier.

For more information and instructions for configuring a custom host name identifier, see: [http://docs.oracle.com/cd/E28280_01/apirefs.1111/e13952/taskhelp/security/ConfigureACustomHostNameVerifier.html](http://docs.oracle.com/cd/E28280_01/apirefs.1111/e13952/taskhelp/security/ConfigureACustomHostNameVerifier.html).

For more information on sending push notifications to clients, see [Push Notifications](#).

Where to go next

If installing Akula for the first time or completing an upgrade, see [4. Deploy the Akula WAR File on Your Java Server](#). Otherwise, if changing the JMS configuration for an existing deployment, start the Akula Server.

**Deploy the Akula WAR file on your Java Server**

To install the Akula Server, deploy the akula.war file to your Java application server or servlet container. If you want to install the [Akula Management Console](#), which is also a WAR file, you can do so at the same time.

This topic contains the following sections:

- [Deploy the akula.war file on Tomcat](#)
- [Deploy the akula.war file on JBoss](#)
Installing and Configuring Akula

• **Deploy the akula.war file on Oracle WebLogic**

**Deploy the akula.war file on Tomcat**

Tomcat lets you deploy a WAR file in several different ways, including:

- Startup deployment, where you have to manually start and stop the Tomcat server.
- Hot deployment, where you can deploy the WAR file without restarting Tomcat.
- Tomcat Web Application Manager is a browser-based utility used to manage Tomcat.

The following sections describe all three methods.

For more information on Tomcat, see [http://tomcat.apache.org/download-70.cgi](http://tomcat.apache.org/download-70.cgi).

<table>
<thead>
<tr>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>If this is the first time you have installed and used Tomcat, you might find it easiest to use startup deployment.</td>
</tr>
<tr>
<td>You must have the necessary system permissions to install and start Tomcat on your machine.</td>
</tr>
</tbody>
</table>

**Using startup deployment on Tomcat**

Startup, or folder, deployment requires that you restart Tomcat after you deploy a WAR file. Use the following procedure to install the akula.war file on the Apache Tomcat application server.

**To use startup deployment:**

1. Stop the Tomcat server by using the command `shutdown` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `shutdown.bat`
   - On Mac/Unix, run `shutdown.sh`
2. Browse to the `server-sdk/bin` directory where you unzipped the Akula Server ZIP file.
3. Deploy the `server-sdk/bin/akula.war` file by copying it to the webapps directory of your Tomcat installation. Optionally, you can copy the AkulaManagementConsole.war file to the same directory to deploy the Akula Management Console at the same time. If the `unpackWARs` option is set to `true`, the default, in the Tomcat `conf/server.xml` file, Tomcat automatically expands the WAR file when it starts. Otherwise, Tomcat runs the app directly from the WAR file.
4. Start the Tomcat server by using the command `startup` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `startup.bat`
   - On Mac/Unix, run `startup.sh`

**Using hot deployment on Tomcat**

With hot deployment, you can redeploy a WAR file or edit an existing application without restarting Tomcat.

To use hot deployment to deploy the akula.war file, make sure that the `autoDeploy` option is set to `true`, the default, in the Tomcat `conf/server.xml` file to enable hot deployment. Then, copy the akula.war file to the Tomcat webapps directory.

If the `unpackWARs` option is also set to `true`, the default, in the Tomcat `conf/server.xml` file, Tomcat automatically expands the akula.war file.

**To use hot deployment:**
1. Browse to the server-sdk/bin directory where you unzipped the Akula Server ZIP file.
2. Deploy the server-sdk/bin/akula.war file by copying it to the webapps directory of your Tomcat installation. Optionally, you can copy the AkulaManagementConsole.war file to the same directory to deploy the Akula Management Console at the same time. If the unpackWARs option is set to true in the Tomcat conf/server.xml file, you do not have to expand the akula.war file. Tomcat will automatically expand it when it starts.

Using the Tomcat Web Application Manager

The Tomcat Web Application Manager is a console that lets you stop, deploy, undeploy, and reload an application.

To use the Tomcat Web Application Manager, you must create a user in the Tomcat conf/tomcat-users.xml file that has the proper role, as the following example shows:

```xml
<role rolename="manager-gui"/>
<user username="tcAdmin" password="tcAdminPW" roles="manager-gui"/>
```

After creating the user, restart Tomcat and then you can access the Tomcat Web Application Manager at the following URL:

http://localhost:8080/manager/html

Port 8080 is the default port, but it can be different, depending on your configuration.

If the Akula Server has been deployed correctly, it will appear in the list of available apps along with the default applications, as the following example shows:

![Tomcat Web Application Manager](image)

Optionally, you can also use the Tomcat Web Application Manager to deploy the Akula Management Console.

Deploy the akula.war file on JBoss

This section describes how to deploy the akula.war file on JBoss EAP Version 6.1. This procedure assumes that you are running JBoss in standalone mode, which is typical for a development environment. If you are running JBoss in domain mode, see the JBoss documentation for deployment information. As described in the steps below, the
database name is specified in the JBoss standalone.xml file; the name specified as the db_schema (database name) in the Akula context.properties file is not used.

This procedure describes starting and stopping JBoss from the command line. You can also start and stop JBoss, and deploy WAR files by using the JBoss Management Console.

Use the following procedure to deploy the akula.war file on JBoss in standalone mode:

1. Map the IP address to the machine name in your Java server's hosts file. On Linux and Mac systems, the file is in the /etc directory. On a Windows Server 2008 system, it is in the C:\Windows\System32\Drivers\etc directory.
2. Ensure that you have already set the JBOSS_HOME environment variable to the home directory of JBoss.
3. Change to the /standalone/configuration directory.
4. Open the standalone.xml file and add the following Akula JNDI data sources:

   pool-name="akuladb" jndi-name="java:/comp/env/jdbc/akula-db"
   pool-name="akula-dev-db" jndi-name="java:/comp/env/jdbc/akula-dev-db"

View full sample:

```xml
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource
      jta="false" jndi-name="java:/comp/env/jdbc/akula-db" pool-name="akuladb"
      enabled="true" use-ccm="false">
      <connection-url>jdbc:sqlserver://akulasql.vov.com:1433;db=ValidateSDK<
      /connection-url>

      <driver-class>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-class>
      <driver>sqljdbc</driver>
      <security>
        <user-name>sdkuser</user-name>
        <password>sdk!password</password>
      </security>
      <validation>
        <validate-on-match>false</validate-on-match>
        <background-validation>false</background-validation>
      </validation>
      <statement>
        <share-prepared-statements>false</share-prepared-statements>
      </statement>
    </datasource>
  </datasources>
</subsystem>
```
5. Change to the JBoss /bin directory.
6. Run ./standalone.sh to start the JBoss server in standalone mode. This command uses the standalone.xml file for running the JBoss server instance.
   To access JBoss on a remote machine, specify the -b parameter to standalone.sh script with the IP address of the machine:
   
   /standalone.sh -b 172.17.3.201

7. Copy the /server-sdk/bin/akula.war file from the location where you installed Akula to JBOSS_HOME/standalone/deployments directory. Optionally, you can also copy the AkulaManagementConsole.war file at the same time to deploy the Akula Management Console.

   If the autoDeploy attribute is true, the default value, JBoss automatically deploys the akula.war file and create server.war.deployed file. If the autoDeploy attribute is false, you have to restart JBoss.

   If you see any errors during deployment, check the JBoss logs/server.log and boot.log files.
Deploy the akula.war file on Oracle WebLogic

This section describes how to deploy the akula.war file on Oracle WebLogic 12.1.

You typically only deploy the Akula Server on WebLogic in a production environment. Therefore, you should ensure that you have already configured an external JNDI database for use by the Akula Server, instead of using the default Derby database. For more information on configuring a JNDI database, see Specify the Database for the Akula Server.

Use the following procedure to deploy the akula.war file on WebLogic:

1. Use the WebLogic domain configuration tool at `$MW_HOME/wlserver/common/bin/config.sh` to create a domain for the Akula Server. For example, when used to create a domain named “akula”, this tool creates a new directory named `$MW_HOME/user_projects/domains/akula`.
2. Copy the `server-sdk/bin/akula.war` file from the location where you installed Akula to the `$MW_HOME/user_projects/domains/akula/autodeploy` directory. Optionally, you can copy the `AkulaManagementConsole.war` file to deploy the Akula Management Console at the same time.
3. Create a new directory in the “akula” domain named `$MW_HOME/user_projects/domains/akula/akula_jars`.
4. Download the 3rd party JAR files from the following link: jaxb-jars.zip
5. Extract the downloaded JAR files to the `$MW_HOME/user_projects/domains/akula/akula_jars` directory. These JAR files are required by the Akula Server and must be loaded before the akula.war file loads.
6. Edit the `startweblogic.sh` file in the “akula” domain to add the `akula_jars` directory to the PATCH_CLASSPATH environment variable. This file should contain the following line:

   ```
   export PATCH_CLASSPATH=${DOMAIN_HOME}/akula_jars/*:$PATCH_CLASSPATH
   ```

   where `DOMAIN_HOME` is an environment variable created automatically by the WebLogic `configure.sh` command and points to the home directory of the "akula" domain.
7. Start WebLogic.

Where to go next

If installing Akula for the first time, see Install the Akula Command-line Management Utility. Otherwise, see Configuring the Akula Server.

Install the Akula Command-line Management Utility

The Akula Command-line Management Utility is a command-line utility that lets you configure the Akula Server and app scopes running on the server. The Akula Command-line provides a simple way to write HTTP requests against the REST endpoints on the Akula Server used to manage app scopes. For a complete list of the REST endpoints, see REST API Reference.

For all three supported platforms, Akula ships the Akula Command-line Management Utility as a Python application. For Microsoft Windows only, Akula also ships the Akula Command-line Management Utility as the akula.exe file.

This topic contains the following sections:

- Use the Akula Command-line Management Utility executable on Microsoft Windows
- Install the Akula Command-line Management Utility on Windows, Mac, and Linux
- Use the Akula Command-line Management Utility on MacOS or Linux
Use the Akula Command-line Management Utility executable on Microsoft Windows

For Microsoft Windows only, Akula also ships the Akula Command-line Management Utility as the akula.exe file in the `server-sdk/tools/mgmt-util/bin` folder.

To use the Akula Command-line Management Utility executable on Windows:

1. Locate the `server-sdk/tools/mgmt-util/bin` folder in the location where you unzipped the Akula Server ZIP file. If you have not done so already, download the Akula Server from the Downloads (log in is required).
2. Add the `server-sdk/tools/mgmt-util/bin` folder to your system path, or change to the `server-sdk/tools/mgmt-util/bin` folder.
3. Make sure the Akula Server is running.
4. Open a command window.
5. Invoke the Akula Command-line. Include the URL of the Akula Server on invocation, as the following example shows:

   ```
   akula.exe http://your.server.com/akula
   ```

   For example, if you deployed the Akula Server on Tomcat, use the command:

   ```
   akula.exe http://localhost:8080/akula
   ```

   If the Akula Command-line starts successfully and connects to the Akula Server, you see the Akula Command-line's prompt:

   ```
   >
   ```

   Enter commands at the prompt. For example, enter `help` to see usage information. For more information on using the Akula Command-line, see Using the Akula Command-line Management Utility.

6. If you deployed the default Akula Server configuration files, as described on Set up the Akula Server Environment, then log in to the Akula Server using the default Administrator account:

   ```
   >login admin admin
   ```

   Otherwise, you can log in with the credentials defined by your own security realm.

Install the Akula Command-line Management Utility on Windows, Mac, and Linux

Akula ships the Akula Command-line Management Utility as a Python application and (for Windows only) as an executable in the `/server-sdk/tools/mgmt-util` folder. You must install Python 2.7.4 and several other utilities to use the Akula Command-line Management Utility.

Installation requirements

Before you can install the Akula Command-line, you must ensure that the Akula Server is installed and configured, and that you have Python 2.7.4 installed on your machine. The following table lists the system requirements for the Akula Command-line:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Microsoft Windows</th>
<th>Mac</th>
<th>Unix</th>
</tr>
</thead>
</table>

© Copyright 2014, Verivo Software, Inc. All Rights Reserved.
### Install Python 2.7.x

The Akula Command-line requires that you have installed Python 2.7.4. If you are running the Akula Command-line on a Mac, you should already have Python installed. Verify that your version is Python 2.7.4.

To test your Mac to see if Python is installed, open a terminal window and enter the following command:

```
> python
```

If Python is installed, you will see the Python version number and other information in the terminal window.

If you have to install Python 2.7.4, you can download and install it from:

[http://www.python.org/download/releases/2.7.4/](http://www.python.org/download/releases/2.7.4/)

On Microsoft Windows, make sure to install the 32-bit version of Python to work with EasyInstaller, even if you are on a 64-bit version of Windows.

### Install EasyInstaller, PIP, PyReadline, and Readline

PyReadline (Windows) and Readline (Mac) provide command-line completion so that you can enter the first part of a command in the Akula Command-line, and then hit the Tab key to complete the command. Pip and EasyInstaller are utilities that you use to install PyReadline and Readline.

Microsoft Windows requires that you install PyReadline to use the Akula Command-line. On a Mac, Readline is optional, and on Unix it should already be installed with Python.

**To install PyReadline on Windows:**

1. Download the EasyInstaller **setuptools-0.6c11.win32-py2.7.exe** file for Windows from [http://pypi.python.org/pypi/setuptools](http://pypi.python.org/pypi/setuptools).
2. Run the **setuptools-0.6c11.win32-py2.7.exe** file to install EasyInstaller.
3. Download the latest version of pip from [http://pypi.python.org/pypi/pip#downloads](http://pypi.python.org/pypi/pip#downloads). The version 1.3.1 file is named **pip-1.3.1.tar.gz**.
4. Uncompress the **pip-1.3.1.tar.gz** file to a temporary location. By default, the top-level folder of the
4. Uncompressed files is named **pip-1.3.1**.
5. Copy all of the files and directories under **pip-1.3.1** to your top-level Python folder. For example, copy them to C:\Python27.
6. Change directories to your Python folder. For example, change folder to C:\Python27.
7. Run the following command from the Python folder to install pip:

   ```
   python setup.py install
   ```

8. Add the following directories to your system **Path** environment variable:

   ```
   c:\Python27
   c:\Python27\Scripts
   ```

   Modify this path as necessary for your Python installation. On Windows, you have to close and reopen your command window for your path change to take effect.

9. Run the following command from the Python folder to install PyReadline:

   ```
   easy_install pyreadline
   ```

**To install Readline on a Mac:**

Readline is optional on a Mac. Install it only if you want command-line completion to work with the Akula Command-line.

You use EasyInstaller to install Readline. EasyInstaller should already be installed on your Mac.

1. In a terminal window, enter the following command to install Readline:

   ```
   sudo easy_install readline
   ```

**Use the Akula Command-line Management Utility on MacOS or Linux**

The Akula Command-line ships as part of Akula. To start the Akula Command-line on Linux or MacOS:

1. Locate the `server-sdk/tools/mgmt-util/src` folder in the location where you unzipped the Akula Server ZIP file.
2. Open the `server-sdk/tools/mgmt-util/src/akula.py` file in a text editor.
3. Edit the first line of the akula.py file to specify the location where you installed Python 2.7.4. By default, the line appears as below:

   ```
   #!/usr/bin/python
   ```

   For example, if you installed Python to the `/usr/bin/python27` folder, edit this line as shown below:

   ```
   #!/usr/bin/python27
   ```

4. Make sure the Akula Server is running.
5. Add the `server-sdk/tools/mgmt-util/src` folder to your system path, or change to the `server-sdk/tools/mgmt-util/src` folder.
6. Open a command window.
7. Invoke the Akula Command-line. Include the URL of the Akula Server on invocation, as the following example shows:
akula.py http://your.server.com/akula

where akula.py is the Python file for the Akula Command-line.

For example, if you deployed the Akula Server on Tomcat, use the command:

akula.py http://localhost:8080/akula

If the Akula Command-line starts successfully and connects to the Akula Server, you see the Akula Command-line's prompt:

> 

Enter commands at the prompt. For example, enter help to see usage information. For more information on using the Akula Command-line, see Using the Akula Command-line Management Utility.

8. If you deployed the default Akula Server configuration files, as described on Set up the Akula Server Environment, then log in to the Akula Server using the default Administrator account:

> login admin admin

Otherwise, you can log in with the credentials defined by your own security realm.

Where to go next

Now that you have installed the Akula Command-line, go to Install an Akula License.

Install an Akula License

You require an Akula license to perform most actions with the Akula Server, including accessing backend data. If you do not currently have an Akula Server license, you can contact us for information on obtaining one.

After you obtain a license, install it on the Akula Server in one of two ways:

1. **Online installation** - Requires that your Akula Server has an external internet connection so that it can connect to the Akula Licensing Server. You perform this process by using the Akula Command-line Management Utility.

2. **Offline installation** - If your Akula Server is behind a firewall, or has no external internet connection, you use the offline installation procedure to install the license. In this scenario, you obtain a licensing file from Verivo and deploy it directly on your Akula Server. This procedure is normally only performed in a production environment. For more information, see Offline Installation of an Akula License.

This document describes how to install a license, and contains the following sections:

- About licensing
- Configure proxy settings for online licensing server
- Online license generation and installation
- Managing installed licenses and users
  - Where to go next

About licensing

Akula Server licenses are represented by an AKL file in the AKULA_HOME\licenses directory of your Akula Server.
The \texttt{AKULA\_HOME/licenses} directory can contain multiple licenses, but only one is considered to be the active license.

A license can have the following possible states:

1. **Available** - A purchased or evaluation license that resides on the Verivo Licensing Server. You must activate the license and download it before you can install it on the Akula Server.
2. **Activated** - A license that has been downloaded to an Akula Server and contains information that ties it to that specific Akula Server.
3. **Active** - The single activated license currently in use on the Akula Server. Every license in \texttt{AKULA\_HOME/licenses} is activated, meaning it has been tied to a specific Akula Server, but only one license can be active. The active license is always the last valid one you installed. Therefore, if you get an evaluation license, then upgrade to a full year license, they both are activated but the full year license is the active license.
4. **Expired** - A license that is no longer valid.

If you do not currently have an Akula Server license, you can request an evaluation license at [https://info.verivo.com/akula-free-trial.html](https://info.verivo.com/akula-free-trial.html).

**Configure proxy settings for online licensing server**

When using online licensing, the Akula Server must have an external internet connection so that it can connect to the Akula Licensing Server. In some hosting configurations, the JEE server or servlet container hosting the akula.war file might use a proxy server. In that case, you must specify information about the proxy server to the Akula Server.

As an alternative, use the offline licensing procedure described in [Offline Installation of an Akula License](#).

- This procedure is only required when the JEE server or servlet container uses a proxy server. Otherwise, you can skip this procedure.

The way you pass information about the proxy server to the Akula Server is based on the JEE server or servlet container:

- **Tomcat**: Specify the proxy information by setting `JAVA_OPTS` for all Tomcat instances, or edit `setenv.sh` for a specific instance.
- **JBoss**: Specify the proxy information by setting `JAVA_OPTS` for all JBoss instances, or edit `standalone.sh` for a specific instance.
- **WebLogic**: Specify the proxy information by setting `JAVA_OPTIONS`.

The proxy settings that you configure include:

- For **HTTP**: `httpProxyHost` and `httpProxyPort`
- For **HTTPS**: `httpsProxyHost` and `httpsProxyPort`

Note that the `httpsProxyUser` and `httpsProxyPassword` proxy settings are not supported. Therefore, the proxy server does not require authentication.

For example, you can set `JAVA_OPTS` as shown below for Tomcat or JBoss:
Installing and Configuring Akula

JAVA_OPTS="-Xms1024m -Xmx1024m -XX:MaxPermSize=512M
-Dhttp.proxyHost=localhost -Dhttp.proxyPort=80
-Dhttps.proxyHost=localhost -Dhttps.proxyPort=80"

For WebLogic, edit startWebLogic.sh under home/user_projects/domains/akula to set JAVA_OPTIONS, assuming "akula" is the WebLogic domain name, as shown below:

export JAVA_OPTIONS=" -Dhttp.proxyHost=localhost -Dhttp.proxyPort=80
- Dhttps.proxyHost=localhost -Dhttps.proxyPort=80"

Online license generation and installation

Use the Akula Command-line Management Utility to install a license. To install a license using online installation:

1. Make sure that you have registered for a license. If you do not already have a license, you can request an evaluation at https://info.verivo.com/Akula-20-Evaluation-Registration.html.
2. Start the Akula Server.
3. For Microsoft Windows only, if you are using the akula.exe file, invoke the Akula Command-line and include the URL of the Akula Server on invocation, as the following example shows:

   akula.exe http://your.server.com/akula

For all platforms, use Python to invoke the Akula Command-line from the command line. Include the URL of the Akula Server on invocation, as the following example shows:

   akula.py http://your.server.com/akula

where akula.py is the Python file for the Akula Command-line.

4. Log in as a user with administration privilege for the Akula Server.

   > login adminUserName adminPWord

For example, if you installed the default SQLite security realm or if you installed the Akula Evaluation Installer, log in as the administrator using the username and password of admin:
5. Use the `loginlicense` command of the Akula Command-line to log in to the Verivo Licensing Server using the same credentials that you use to log in to the Verivo DevCenter:

   ```
   > loginlicense username@myco.com myPassWord
   ```

6. Use the `availablelicenses` command of the Akula Command-line to list all of your available licenses and the associated license keys.

   ```
   > availablelicenses
   ```

7. Use the `activatelicence` command of the Akula Command-line to activate your license by passing the license key as shown below. The license file is written to the `AKULA_HOME/licenses` directory of your Akula Server.

   ```
   > activatelicence AKxxx-xxxx-xxx-xxx
   ```

   Be sure to replace the XXXs with your actual key.


   ```
   > logoutlicense
   ```

   If you later want to update your license, use the same procedure to install the updated license from the Verivo Licensing Server.

**Managing installed licenses and users**

Use the Akula Command-line Management Utility to manage local installed licenses and also licensed users. The
Akula Command-line contains the following commands for managing licenses and users:

<table>
<thead>
<tr>
<th>Category</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online licensing</td>
<td>loginlicense &lt;username&gt;</td>
<td>Log in to the Verivo Licensing Server using the same username and password that you use to log in to the Verivo DevCenter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>availablelicenses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>List the available Akula licenses available to the user currently logged in to the Verivo Licensing Server.</td>
</tr>
<tr>
<td></td>
<td>activate &lt;key&gt;</td>
<td>Activate one of your available Akula licenses. Activating a license copies the license to a file in the AKULA_HOME\licenses directory of your Akula Server.</td>
</tr>
<tr>
<td></td>
<td>logoutlicense</td>
<td>Log in to the Verivo Licensing Server.</td>
</tr>
<tr>
<td>Offline licensing</td>
<td>createlicenserequest &lt;file&gt;</td>
<td>Create an offline license activation request. This command writes to the specified file the information necessary to pass to the Verivo Licensing Server to activate a license.</td>
</tr>
<tr>
<td></td>
<td>installlicense &lt;file&gt;</td>
<td>Install an Akula license file on the Akula Server. Installing the license writes the specified license file to the AKULA_HOME\licenses directory.</td>
</tr>
<tr>
<td></td>
<td>installedlicenses</td>
<td>List all the installed licenses for the Akula Server. While you can have multiple installed and activated licenses, only one license can be active at a time. This command can return multiple licenses if you have multiple license files in the AKULA_HOME\licenses directory.</td>
</tr>
<tr>
<td></td>
<td>active &lt;key&gt;</td>
<td>List the active license for the Akula Server. The active license is the most recently added license to the AKULA_HOME\licenses directory.</td>
</tr>
<tr>
<td>User management</td>
<td>licensedusers</td>
<td>List the licensed users for the Akula Server. A licensed user is identified by their username, or principal, and the realm in which their user credentials are stored.</td>
</tr>
</tbody>
</table>
## Where to go next

Now that you have installed a license, go to Validate the Akula Server Install.

### Validate the Akula Server Install

You can validate that you have installed the Akula Server correctly by deploying and running the selftest.akz file. The selftest.akz file represents an app scope on the Akula Server.

This topic contains the following sections:

- [Validate your installation](#)
- Troubleshooting the Akula Server installation
- Where to go next

### Validate your installation

To validate your Akula Server installation:

1. Ensure that you have set the AKULA_HOME environment variable that specifies the location of a deployed app scope. For more information, see Set up the Akula Server Environment.
2. If the Akula Server is running, stop it.
3. Browse to the location where you unzipped the Akula Server ZIP file.
4. Locate the selftest.akz file in the /server-sdk/samples/projects/selftest/target directory where you unzipped the Akula Server ZIP file.
5. Copy the selftest.akz file to the AKULA_HOME/deploy directory to deploy the AKZ file on your Akula Server.
7. Open the following URL in a browser:

   http://localhost:8080/akula/SelfTest/data/SelfTest

   In this URL, **akula** is the name of the deployed Akula Server, **selftest** is the name of the deployed app scope on the Akula Server, and **data/SelfTest** is the name of the endpoint exposed by the selftest app scope that

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deletelicensedusers</td>
<td>Clear all the licensed users from the Akula Server. This reduces the number of authenticated users counted against the license to zero. Any user currently logged in will stay logged in.</td>
</tr>
<tr>
<td>deletelicenseduser</td>
<td>Clear a licensed user from the Akula Server. This might be necessary when you reach the limit of users allowed by your license. If the removed user is currently logged in, they remain logged in.</td>
</tr>
</tbody>
</table>

- **realm** is the realm name containing the user's credentials.
- **principal** is the username of the user to clear.
you can access.

This URL works if you are running the Akula Server locally on your machine. If the Akula Server is running remotely, open the following URL:

http://yourJavaServerURL/akula/selftest/data/SelfTest

If everything is installed correctly, you see the following message:

Akula Server Self-Test: PASSED

Troubleshooting the Akula Server installation

If you are unable to connect to the selftest.akz app scope, you can use the following steps to troubleshoot the problem:

1. Make sure that you set the `AKULA_HOME` as a system environment variable. For more information, see Set up the Akula Server Environment.
2. Ensure that you are copying AKZ files and server configuration files to the correct location as defined by the `AKULA_HOME` environment variable. For more information, see Set up the Akula Server Environment.
3. Make sure the selftest.akz file was deployed from the `AKULA_HOME/deploy` directory to the `AKULA_HOME/apps` directory. If the selftest.akz file was successfully deployed, you should see a new directory named `AKULA_HOME/apps/selftest`. If the selftest.akz file was not successfully deployed:
   a. On Unix, make sure that the directory, and subdirectories, specified by the `AKULA_HOME` environment are writable by the user who started your Java server.
   b. Restart your Java server.
4. Make sure you set the `JAVA_OPTS` environment variable to define the memory requirements of the Akula Server. For more information, see Set up the Akula Server Environment.
5. Verify that the JMS server resource definitions are correct, see Configure a JMS Server.
6. On a Mac, if localhost fails to resolve, edit the `/etc/hosts` file to include the following mapping for localhost:

   127.0.0.1  localhost UserName

7. Lastly, restart your Java server.

If you do not see the SelfTest.vas file deployed to the `AKULA_HOME/apps` directory after copying it into the `AKULA_HOME/deploy` directory, refer to this KB article:

http://support.verivo.com/entries/23339107-Vas-files-are-not-unpacking-in-apps-directory-after-placing-them-in-deploy-

Where to go next

After you have validated the Akula Server installation, you can:

1. Work through the Akula samples and tutorials at Getting Started.
2. If you are configuring the Akula Server for a development or deployment environment, proceed to Configuring the Akula Server.

You can also download and install the Akula client SDK for your environment. Akula provides a separate SDK for building Android, iOS, and JavaScript client apps. For more information on installing the client SDKs, see Using the Akula SDKs.

Uninstall the Akula Server

To uninstall the Akula Server, remove the akula.war file and the deployed server application from your Java server.
This document contains the following sections:

- Uninstall the Akula Server on Tomcat
- Uninstall the Akula Server on JBoss
- Uninstall the Akula Server on Oracle WebLogic

### Uninstall the Akula Server on Tomcat

Use the following procedure to uninstall the akula.war file on the Apache Tomcat application server:

1. Stop the Tomcat server by using the command `shutdown` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `shutdown.bat`
   - On Mac/Unix, run `shutdown.sh`
2. Delete the akula.war file from the Tomcat webapps folder.
3. Delete the akula application from the webapps directory.
4. Start the Tomcat server by using the command `startup` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `startup.bat`
   - On Mac/Unix, run `startup.sh`

### Uninstall the Akula Server on JBoss

Use the following procedure to uninstall the akula.war file on JBoss:

1. Delete the akula.war file from the `JBOSS_HOME/standalone/deployments` directory.

   If the `autoDeploy` attribute is `true`, the default value, JBoss automatically undeploys the akula.war file. If the `autoDeploy` attribute is `false`, you have to restart JBoss.

### Uninstall the Akula Server on Oracle WebLogic

Use the following procedure to uninstall the akula.war file on WebLogic:

1. Use the WebLogic Server Administration Console to stop and then to delete the "akula" domain.

### Offline Installation of an Akula License

If your Akula Server is behind a firewall, or has no external internet connection, you use the offline installation procedure to install the Akula Server license. In this scenario, you obtain a license file from Verivo and deploy it directly onto your Akula Server. This procedure is normally only performed in a production environment. For more information on licensing, including the procedure for online license installation, see `Install an Akula License`.

#### Offline license generation and installation

1. Make sure that you have registered for a license. If you do not already have a license, you can contact us for information on obtaining one.

2. Start the Akula Server.
3. For Microsoft Windows only, if you are using the akula.exe file, invoke the Akula Command-line and include the URL of the Akula Server on invocation, as the following example shows:

```
akula.exe http://your.server.com/akula
```

For all other platforms, use Python to invoke the Akula Command-line from the command line. Include the URL of the Akula Server on invocation, as the following example shows:

```
python akula.py http://your.server.com/akula
```

where akula.py is the Python file for the Akula Command-line.

4. Log in as a user with administration privilege for the Akula Server.

```
> login adminUserName adminPWord
```

For example, if you installed the default SQLite security realm, log in as the administrator using the username and password of admin:

```
> login admin admin
```

5. Use the createlicenserequest command of the Akula Command-line to generate the information required by the Verivo Licensing Server to activate a license and to write that information to a file:

```
> createlicenserequest licenserequest.txt
License request created at 'licenserequest.txt'.
You may now retrieve your license from the Akula licensing server. Use the 'installlicense' command to install this license to this Akula server.
```

In this example, the information is written to the licenserequest.txt file in the current directory.

6. Copy the licenserequest.txt file to a machine that has an external internet connection.
7. Log in to the Verivo License Activation page at https://licensing.verivo.com/offlineLicensing/#/login to activate the license. Use your Verivo DevCenter user name and password to log in to the Verivo License Activation page.

8. In the list of licenses, select a license labelled Available for Activation in the Activation Date column.

9. Select the Choose File button, and then select the licenseinfo.txt file that you generated previously.

10. Select the Start Upload button.

11. After the upload completes, copy all the text from the License Object to be copied box into an AKL file, such as mylicense.akl.

12. Copy the AKL file with the license information onto the machine hosting the Akula Server.

13. Start the Akula Command-line Management Utility and log in as a user with administration privilege for the Akula Server.

14. Use the installlicense command of the Akula Command-line to copy the license to the AKULA_HOME\licenses directory and to make the license the active license:

   ```
   > installlicense mylicense.akl
   ```

   If you later want to update your license, use the installlicense command to install a different license file.

Configuring the Akula Server

This section describes the following topics:

- Define Akula Server Administrators
- Allow Cross Origin Requests in a Browser App
- Configure Logging
- Using SSL with Akula
- Using the Akula Command-line Management Utility

Define Akula Server Administrators

When you install the Akula Server, you also install a predefined server app scope that exposes REST endpoints used to configure the server and to configure app scopes running on the server. You use many of these endpoints to configure the authorization mechanism. For example, use these endpoints to create and configure roles, associate permissions with roles, and access information about security realms.

You access these endpoints by making an HTTP request to a URL controlled by the server app scope, or by using the Akula Command-line Management Utility. The URL of these endpoints contains the word server, as the following example shows for the URL to delete a role:

   `http://serverURL/akula/server/manage/scopes/{scope}/roles/{role}`
Every endpoint controlled by the server app scope has predefined permissions that a user is required to have in order to make a request to the endpoint. For example, the endpoint shown above requires that the user be in a role that has the Edit_Roles permission. If the user's role does not have the Edit_Roles permission, a request to that endpoint fails.

At Akula install time, if you use the default configuration files described in Set up the Akula Server Environment, then you have a default administrator configured. Otherwise, no users have the necessary permissions to access the server app scope. Therefore, any request to its endpoints fails.

Additionally, there are REST endpoints associated with individual app scopes. Many of these endpoints are used by a client apps to access a data source or to perform other actions controlled by the server app scope. However, some of these endpoints might only be available to administrators and should not be accessible to normal users.

### Configuring Akula Server administrators

Configure Akula Server administrators in two ways:

- Specify a user group that functions as the Akula Server administrators. Server administrators have the necessary permission to perform any action on any endpoint for any server app scope controlled by the Akula Server.

  The user group must be defined in your authentication realm (such as an Active Directory server). All users in that group have the necessary permissions to function as a server administrator. The procedure for specifying a user group that functions as the Akula administrators is described below.

- Associate individual administrator permissions with a role.

  You typically associate individual administrator permissions to a role to give the users in that role limited administrator privileges. For example, associate the Edit_Logs permission with a role to let the users in that role change the logging level of an Akula logger. For information on how to associate a permission with a role, see Authorizing Users.

### Specifying a user group as the Akula Server administrator

To specify the user group that functions as the Akula Server administrator, edit the AKULA_HOME/global/properties.xml file. The Akula Server reads the properties.xml file at start up. Therefore, after editing this file, you have to restart the Akula Server.

Set the following properties in the properties.xml file to specify the administrator group:

<table>
<thead>
<tr>
<th>Akula property</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
akula_superrole_enabled | Set to `true` to enable the administrator role. Set it to `false` to disable the role.

You typically leave the administrator group active at development time. When you deploy the Akula Server to a live site, and after you have configured roles and permission on the live site, you then disable the server administrator group to prevent unintended access to these services.

If it is necessary to expose administrator endpoints on a deployed Akula Server, use the authorization mechanism to individually allow access to that endpoint. For more information, see [Authorizing Users](#).

| akula_superrole_name | The name of the role corresponding to administrators. The Akula Server creates this role for you.

| akula_superrole_realm | The realm containing the user group for the administrators. The name of the realm is determined by how you configured the connection to the realm. For more information, see [Using Active Directory Server as an Authentication Realm](#).

| akula_superrole_group | The user group on the realm containing all administrator users.

The example below shows the default values of the `isClientProperty`, `overridable`, and `type` attributes for the administrator properties. Do not modify the `isClientProperty`, `overridable`, and `type` attributes of an "akula_" app property from their default values. The Akula Server has an internal representations of these attributes that will not be valid (or respected) if the attributes are changed.

For example:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<properties-provider xmlns="http://akula.verivo.com/schema/server">
  <properties>
    <akula-property key="akula_superrole_enabled"
        description="Enable the super role"
        display="Super Role Enabled"
        isClientProperty="false"
        overridable="false"
        type="boolean">true</akula-property>
    <akula-property key="akula_superrole_name"
        description="Name of the super role"
        display="Super Role Name"
        isClientProperty="false"
        overridable="false"
        type="string">superuser</akula-property>
    <akula-property key="akula_superrole_group"
        description="Name of the group associated with the super role"
        display="Super Role Group"
        isClientProperty="false"
        overridable="false"
        type="string">Administrators</akula-property>
    <akula-property key="akula_superrole_realm"
        description="Name of the realm that the super role's group is a member of"
        display="Super Role Realm"
        isClientProperty="false"
        overridable="false"
        type="string">SqliteRealm</akula-property>
  </properties>
</properties-provider>

The `description` and `display` attributes of the `<akula-property>` tag are optional and default to null. The `isClientProperty` and `overridable` properties are also optional, but have a default value of `true`. In this example, you specify `isClientProperty` as `false` so that the property is not sent to the client on log in, and set `overridable` to `false` so that the property cannot be changed at the app-scope level at run time. For more information on these attributes, see Using App Properties.

### Allow Cross Origin Requests in a Browser App

A client app written using the Akula Client SDK for JavaScript runs in a browser on a mobile device. Browsers enforce a cross-origin policy which means that a script running on a page originating from one site is only allowed to access scripts on other pages originating from that same site. The script cannot access a script on a page originating from a different site.

For two sites to be considered to be matching, the protocol (such as HTTP), hostname, and port number must match.
For example, you serve a web page containing JavaScript from the site http://publicSite.com. That page can then access scripts on other pages originating from http://publicSite.com. However, if the page attempts to access a script on a page located at http://myServer.com, the browser prohibits the request.

The way the browser determines if the cross-origin request is allowed is to send an HTTP OPTIONS request to the destination site. The destination responds to the OPTIONS request to indicate that the request is allowed or not.

**Cross origin request to the Akula Server**

You can run into the cross-origin policy issue when using the Akula JavaScript SDK to write a client app that runs in a browser on a mobile device. For example, your environment deploys the Akula Server at http://myServer.com/akula. However, your public facing web site that serves the app is located at http://publicSite.com. Therefore, because of the cross-origin policy restrictions in a browser, the client app cannot access JavaScript served by http://myServer.com/akula.

**Configuring cross origin requests**

You can configure the Akula Server to allow cross origin requests by editing the AKULA_HOME/global/properties.xml file. This technique is referred to as Cross-Origin Resource Sharing (CORS). The Akula Server reads the properties.xml file at startup. Therefore, after editing this file, you have to restart the Akula Server.

To configure Akula for cross origin requests:

1. Edit the provided properties.xml file to allow license activation from CORS-enforcing clients by adding the type X-Lic-Session-Id to the list of supported headers in the akula_cors_supportedHeaders property.

   ```xml
   <akula-property key="akula_cors_supportedHeaders"
       description="Specify what headers are allowed by CORS"
       display="CORS Supported Headers"
       isClientProperty="false"
       overridable="true"
       type="string">
       Content-Type, Accept, X-Lic-Session-Id, X-Ak-Session-Id, X-Ak-Device-ID, Origin
   </akula-property>
   ```

2. Specify the list of sites allowed to access the Akula Server across origins. In the properties.xml file, use the following predefined app properties to specify a list of sites that are allowed to access the Akula Server across origins:

   To specify the list of sites allowed to access the Akula Server, edit the AKULA_HOME/global/properties.xml file. The Akula Server reads the properties.xml file at startup. Therefore, after editing this file, you have to restart the Akula Server.

   In the properties.xml file, use the following predefined app properties to specify a list of sites that are allowed to access the Akula Server across origins:

<table>
<thead>
<tr>
<th>Property</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>akula_cors_allowCredentials</td>
<td>Used as part of a response to the HTTP OPTIONS request to determine if access to the requested origin is allowed. If true, it indicates that the actual request can be made using credentials, such as cookies, HTTP authentication, and client-side SSL certificates. If false, the actual request is made without credentials. Note that GET requests are not validated. Therefore, if a request is made for a resource with credentials, and if this header is not returned with the resource, the response is ignored by the browser and not returned to web content.</td>
</tr>
<tr>
<td>akula_cors_allowOrigin</td>
<td>A space-delimited list of sites allowed to access the Akula Server. Specify a value of &quot;*&quot; to allow any site to access the Akula Server.</td>
</tr>
<tr>
<td>akula_cors_exposedHeaders</td>
<td>A comma-delimited list of custom headers that are whitelisted by the Akula Server to allow browsers to access.</td>
</tr>
<tr>
<td>akula_cors_supportedMethods</td>
<td>A comma-delimited list of the allowed HTTP request methods to the Akula Server. Method include: GET, POST, PUT, DELETE, HEAD, OPTIONS, and PATCH.</td>
</tr>
<tr>
<td>akula_cors_supportedHeaders</td>
<td>A comma-delimited list of custom headers that can be used in cross-origin requests. A cross-origin request fails if it contains a custom header that is not listed by this property. Many HTTP requests to the Akula Server pass a token in the headers that identifies an authenticated user to the server. The name of the header is X-Ak-Session and should be included in the list of supported headers.</td>
</tr>
<tr>
<td>akula_cors_maxAge</td>
<td>Before a browser allows a cross-origin request, it first sends an HTTP OPTIONS request to the destination site to determine if the destination supports the request. The browser then caches the response from the destination. This property specifies the time, in seconds, before the cached response is cleared and the browser has to make another OPTIONS request to the destination.</td>
</tr>
</tbody>
</table>

The example below shows the default values of the isClientProperty, overridable, and type attributes for the cross origin request properties. Do not modify the isClientProperty, overridable, and type attributes of an "akula_" app property from their default values. The Akula Server has an internal representations of these attributes that will not be valid (or respected) if the attributes are changed.
The following example shows a properties.xml file that lists three sites that can access the Akula Server using the HTTP GET, POST, PUT, DELETE, and OPTIONS methods:
The description and display attributes of the <akula-property> tag are optional and default to null. The is
ClientProperty and overridable properties are optional and have a default value of true. In this example, you specify isClientProperty as false so that the property is not sent to the client on log in, and set overridable to true so that the property can be set for an app scope. For more information on these attributes, see Using App Properties.

To allow any site to make a request to the Akula Server using any HTTP method, use the following settings:
<xml version="1.0" encoding="UTF-8" standalone="yes"?>
<properties-provider xmlns="http://akula.verivo.com/schema/server">
  <properties>
    <akula-property key="akula_cors_allowOrigin"
        description="Specify what origins are allowed by CORS"
        display="CORS Allow Origin"
        isClientProperty="false"
        overridable="true"
        type="string">*</akula-property>

    <akula-property key="akula_cors_supportedMethods"
        description="Specify what methods are allowed by CORS"
        display="CORS Supported Methods"
        isClientProperty="false"
        overridable="true"
        type="string">GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH</akula-property>

    <akula-property key="akula_cors_supportedHeaders"
        description="Specify what headers are allowed by CORS"
        display="CORS Supported Headers"
        isClientProperty="false"
        overridable="true"
        type="string">Content-Type, Accept, X-Lic-Session-Id, X-Ak-Device-Id, X-Ak-Session-Id, Origin</akula-property>

    <akula-property key="akula_cors_exposedHeaders"
        description="Specify what headers are exposed by CORS"
        display="CORS Exposed Headers"
        isClientProperty="false"
        overridable="true"
        type="string">customHeader1, customHeader2</akula-property>

    <akula-property key="akula_cors_allowCredentials"
        description="Specify CORS allows credentials"
        display="CORS Allow Credentials"
        isClientProperty="false"
        overridable="true"
        type="string">true</akula-property>

    <akula-property key="akula_cors_maxAge"
        description="Specify the max age for CORS"
        display="CORS Max Age"
        isClientProperty="false"
        overridable="true"
        type="string">3600</akula-property>
  </properties>
</properties-provider>

Configure Logging
The Akula Server has built-in support for the Logback logging framework, the successor to the open-source log4j project, which enables you to capture logging messages generated from the Akula Server and from your code running on the Akula Server. Because the Akula Server relies on Logback, you can use the Logback documentation to find out much of the information that you need to to work with logging in Akula.

As you build and deploy your own app scopes on the Akula Server, you can dispatch your own logging messages for capture. For example, you write a custom module and deploy it as part of an AKZ file. To help you debug your module, you dispatch logging messages from different areas of your code.

Akula includes two REST APIs that use Logback. The logging REST API enables you to set the logging level of built-in Akula loggers. The auditing REST API enables you to to create log files containing only information for specified endpoints during a requested time frame. You can use the default settings for auditing, or you can use appender and layout files to create custom reports, as described in this topic.

This document contains the following sections:

- Default logging
- About Logback
- About configuring logging
- Configuring loggers
- Configuring the root logger
- Configuring appenders and layouts

In addition to the topics described here, you can also create a custom logger within your Akula routes. For more information, see Logging Within Routes.

### Default logging

When you install the Akula Server, logging is enabled by default. All default logging messages are written to the AKU_LA_HOME/logs/akula.log file.

By default, the following messages are written to the akula.log file:

- All Akula INFO messages
- All ERROR messages from the internal Akula Server components, such as Hibernate
- All WARN messages

Default logging is controlled by the WEB-INF/classes/logback.xml file in the deployed Akula WAR file on your Java EE server.

### About Logback

Logback is composed of three main components:

- **Loggers**: A hierarchical collection of components that determines the area of the product from which to capture logging messages, and the log level of the messages to capture.
- **Appenders**: Specifies the destination of a logging message, such as a file, database, console, or other. Each logger is associated with one or more appenders. The Akula logging mechanism supports third-party adapters that conform to the Logback requirements.
- **Layouts**: Specifies the format of the logging message.

### About configuring logging

To configure logging, create logback.xml files that specify the loggers, appenders, and layout used by the logging
mechanism. These XML files are read when the Akula Server starts. If you want to change these files, you must restart the Akula Server.

For example, you might configure the logging mechanism to capture:

- All DEBUG messages for a single app scope and write the messages to the console.
- All DEBUG messages from the Akula security code for a single app scope and write the messages to a file.
- All WARN messages for all app scopes and write the messages to the console.

You can dynamically control the logging mechanism by using the Akula Command-line Management Utility, or by making HTTP request directly to the REST endpoints that control logging. Dynamic control lets you change the logging level of a previously-defined logger at run time.

For example, you use the XML configuration files to create a logger to capture all WARN messages. However, during application testing, you detect an issue and now want to capture all DEBUG messages. Because the Akula Server supports dynamic logging configuration, you can change logging levels without restarting the Akula Server. After you resolve the issue, you change back to logging only WARN messages.

Configuring logging for different app scopes

The Akula logging mechanism lets you control logging at the Akula Server level for all deployed app scopes, or individually for specific app scopes:

- **Server level**
  Configure logging at the Akula Server level to capture logging messages from any app scope deployed on the Akula Server. To configure logging at the Akula Server level, create a `logback.xml` file in the `AKULA_HOME/aktl` directory.

- **App scope level**
  Configure logging for an individual app scope so that you can capture logging messages only from that app scope, where an app scope corresponds to a deployed AKZ file. To configure logging at the app-scope level, create a `logback.xml` file in the `app_root/AKZ-INF/config/scope` directory.

You might run into a situation where the Akula Server WAR file does not start correctly or the server is not able to deploy the AKZ file for an app scope. If the Akula Server cannot deploy an AKZ file, then any logging defined by the AKZ file does not occur.

In this situation, you can edit the logback.xml file for the deployed Akula WAR file. This logback.xml file is located in WEB-INF/classes directory of the deployed Akula WAR file. Any loggers defined in WEB-INF/classes/logback.xml are not managed by the Akula REST endpoints. However, these loggers can dispatch messages before the Akula AKZ loader is active and can therefore help to debug AKZ deployment issues.

Configuring loggers

Configure loggers to control which logging messages to capture and to specify where to write the messages. The name of the logger defines the area of the Akula Server from which you want to capture logging messages, where the name corresponds to a package name of the Akula Server API.

Logger names are case-sensitive and use a dot ("." ) delimited syntax that matches the Akula package names.

For example, to capture DEBUG logging messages from all Akula classes and write them to the console, create a logger in the logback.xml file as shown below:
<configuration>
    <appender name="Logger_All" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>
    <logger name="com.verivo.akula" level="DEBUG">
        <appender-ref ref="Logger_All"/>
    </logger>
</configuration>

Put this file in the AKULA_HOME/global directory to capture DEBUG logging messages from all deployed app scopes, or in the app_scope/AKZ-INF/config/scope directory to capture messages in just that app scope.

To capture DEBUG logging messages dispatched by all classes in the com.verivo.akula.security.* package and write them to a file, create a logger in the logback.xml file as shown below:

<configuration>
    <appender name="Logger_Security" class="ch.qos.logback.core.FileAppender">
        <file>${AKULA_HOME:-.}/logs/data.log</file>
        <append>true</append>
        <encoder>
            <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
        </encoder>
    </appender>
    <logger name="com.verivo.akula.security" level="DEBUG">
        <appender-ref ref="Logger_Security"/>
    </logger>
</configuration>

Notice that the appender uses the Logback ch.qos.logback.core.FileAppender class to specify to write messages to a file.

The <file> attribute of the appender specifies the location of the log file relative to the AKULA_HOME environment variable. In this example, the destination of the log file is the AKULA_HOME/logs directory. If you omit the AKULA_HOME environment variable from the <file> attribute, the file is written by default to the root directory of your JVM.

For more information on the AKULA_HOME environment variable, see Set up the Akula Server Environment.

To capture DEBUG logging messages dispatched by all classes in the com.verivo.akula.security.* package and capture WARN logging messages from a single class named named AKAuthService, create a logger in the logback.xml file as shown below:
### Using the default loggers

By default, the Akula Server defines several default loggers and writes the output of these default loggers to files in the `AKULA_HOME/logs` directory. For more information on setting `AKULA_HOME`, see [Set up the Akula Server Environment](#).

These default log file includes the following:

<table>
<thead>
<tr>
<th>Log file</th>
<th>Usage</th>
<th>Logging configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>akula.log</td>
<td>Contains INFO logging messages from com.verivo.akula and from other services running on your Java EE server. This log file is useful to debug server problems if the Akula Server is not starting correctly, the Akula Server is not able to load an AKZ file, or the Akula Server is unable to activate other loggers.</td>
<td>WEB-INF/classes/logback.xml in the deployed Akula WAR file on your Java EE server.</td>
</tr>
</tbody>
</table>
As part of installing Akula, you copied the logback.xml file to the \texttt{AKULA_HOME/global} directory. If you omitted that step during installation, you will not see the security.log, data.log, and mgmt.log files in the \texttt{AKULA_HOME/logs} directory. However, you will see the akula.log file because that file is configured by the logback.xml file built into the Akula Server.

### Setting logging message levels

The Akula logging mechanism supports all Logback logging levels, including:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- ALL
- OFF

Notice in the examples above that all logger definitions use the \texttt{level} attribute of the \texttt{<logger>} tag to set the logging level.

You can change logging levels dynamically at runtime by using the Akula Command-line, or by making HTTP request directly to the REST endpoints that control logging. To use REST endpoints, see \texttt{Logging REST API}.

In the Akula Command-line, use the following commands:

- \texttt{logging} - Returns the loggers defined for a specific app scope. Specify \texttt{server} as the app scope to see all loggers defined at the server level.
- \texttt{setlogginglevel} - Sets the logging level for a logger for a specific app scope.

For more information on the Akula Command-line, see \texttt{Using the Akula Command-line Management Utility}.

For example, to see all loggers defined for the app scope called App1, use the following command:
The output of the command is shown below:

```
Loggers:
  Logger:
    Name: com.verivo.akula.security
    Config:
      Level: INFO
  Logger:
    Name: com.verivo.akula
    Config:
      Level: INFO
  Logger:
    Name: com.verivo.akula.data
    Config:
      Level: INFO
```

To set the logging level for the logger named com.verivo.akula.data to DEBUG for the App1 app scope, use the following command:

```
> setlogginglevel App1 com.verivo.akula.data DEBUG
```

You then run the `logging` command again and see the following output. Notice that the logging level of `com.verivo.akula.data` is now DEBUG:

```
> logging App1
Loggers:
  Logger:
    Name: com.verivo.akula.security
    Config:
      Level: INFO
  Logger:
    Name: com.verivo.akula
    Config:
      Level: INFO
  Logger:
    Name: com.verivo.akula.data
    Config:
      Level: DEBUG
```

For more information on the Akula Command-line, see Using the Akula Command-line Management Utility.

**Logger hierarchy**

Loggers are hierarchical. That is, loggers have a parent/child relationship where children can inherit attributes of their parents or override attributes.

For example, you create a logger for the `com.verivo.akula.*` package and specify the logging level as WARN,
as shown below:

```xml
<configuration>
    <appender name="Logger_All" class="ch.qos.logback.core.ConsoleAppender">
        ...
    </appender>

    <logger name="com.verivo.akula" level="WARN">
        <appender-ref ref="Logger_Security" />
    </logger>

    <logger name="com.verivo.akula.security.authentication.AKAuthService">
        <appender-ref ref="Logger_Security" />
    </logger>
</configuration>
```

This example also defines a logger for the AKAuthService class, but omits a logging level specification. The logger for AKAuthService then inherits the logging level specification from its nearest ancestor, in this example from the logger named `com.verivo.akula.*`.

### Configuring the root logger

The root logger is a predefined logger in Logback. Configure the root logger by using the `<root>` tag in the logging configuration file, rather than the `<logger>` tag.

The root logger has a predefined name of "ROOT" so do not specify a name attribute when you configure it. You can only specify the level of the logger.

### Configuring appenders and layouts

Appenders and layouts control the destination of a logging message, and the format of the logging message.

When you define the message format, you can use the regular Akula Mapped Diagnostic Context (MDC) keys or the Auditing REST API MDC keys. To use the Auditing REST API keys, you must disable auditing on the server and include the following Auditing REST API Appender node in your appender:

```xml
<filter class="com.verivo.akula.audit.filter.AuditFilter" />
```

You can use any predefined Logback appenders in Akula, such as the following:

- `ch.qos.logback.core.OutputStreamAppender` - Appends logging messages to an output stream.
- `ch.qos.logback.core.ConsoleAppender` - Appends logging messages to the console.
- `ch.qos.logback.core.FileAppender` - Appends logging messages to a file.

You can use the predefined Logback layouts, such as:

- `ch.qos.logback.classic.PatternLayout` - The default layout formats a logging message as a String.
- `ch.qos.logback.classic.html.HTMLLayout` - Formats logging messages as HTML.
- `ch.qos.logback.classic.log4j.XMLLayout` - Formats logging messages as XML in a log4j.dtd compliant
format.

You can also use your own custom appenders and layouts. For more information, see Appenders and Layouts in the Logback documentation.

### Configuring MDC context keys in an appender

A Mapped Diagnostic Context (MDC) distinguishes interleaved log output from different sources. Logback has built-in support for MDC context tags, and the Akula Server defines custom MDC context keys that you can use when configuring an appender.

Use the Akula MDC context keys in the `<pattern>` attribute of an appender, as the following example shows:

```
<pattern>
  %d %-5level [%X{AppName}] [%X{req.userAgent}] [%thread] [%logger{0}] %msg%n
</pattern>
```

This example uses two Akula MDC context keys: `AppName` and `req.userAgent`. The `AppName` context key adds the name of the server scope, corresponding to the name of the deployed AKZ file, to the logging message. The `req.userAgent` context key adds information about the browser to the logging message.

You specify an Akula MDC context key in the appender with the following syntax:

`[%X{context_key}]`

For example:

`[%X{AppName}]`

The following table lists the regular Akula MDC context keys:

<table>
<thead>
<tr>
<th>Log Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>Username of the current user.</td>
</tr>
<tr>
<td>AppName</td>
<td>App scope name, which corresponds to the name of the deployed AKZ file.</td>
</tr>
<tr>
<td>TransactionID</td>
<td>A unique value assigned to each client request. You can use this to sort or filter your log file by specific requests.</td>
</tr>
<tr>
<td>req.userAgent</td>
<td>Browser use agent string. For example, Chrome/25.0.1364.160.</td>
</tr>
<tr>
<td>req.requestURI</td>
<td>The request URL without the host. For example, /server/ToDo/data/Session.</td>
</tr>
<tr>
<td>req.requestURL</td>
<td>The entire request URL. For example, <a href="http://localhost:8080/server/ToDo/data/Session">http://localhost:8080/server/ToDo/data/Session</a>.</td>
</tr>
</tbody>
</table>
req.queryString | The query string portion of the URL. For example, "param=value".

The auditing REST API includes additional Akula MDC context keys. To use these keys, you must disable automatic auditing on the server. For more information, see [Auditing REST API](#).

The following table lists the auditing MDC context keys:

<table>
<thead>
<tr>
<th>Log Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_SCOPE</td>
<td>Name of the scope being audited.</td>
</tr>
<tr>
<td>AUDIT_CATEGORY</td>
<td>Name of the audit category. Currently, all auditing is in the MGMT (management) category.</td>
</tr>
<tr>
<td>AUDIT_COMPONENT</td>
<td>Name of the component type being audited, such as SCOPE and PUSH.</td>
</tr>
<tr>
<td>AUDIT_COMPONENT_ID</td>
<td>Specific identifier of a component. For example, a scope's ID is its name.</td>
</tr>
<tr>
<td>AUDIT_OPERATION</td>
<td>Operation a component is doing. For example, a valid operation for a PUSH component is CREATE.</td>
</tr>
<tr>
<td>AUDIT_PERMISSIONS</td>
<td>Permission required to perform an operation.</td>
</tr>
<tr>
<td>AUDIT_STATUS</td>
<td>Whether an operation succeeded. Valid values are SUCCESS and FAILURE.</td>
</tr>
<tr>
<td>AUDIT_DATE</td>
<td>Time stamp indicating when an audit event occurred.</td>
</tr>
<tr>
<td>AUDIT_PRINCIPAL</td>
<td>The user name of the person who triggered an event.</td>
</tr>
<tr>
<td>AUDIT_REALM</td>
<td>The realm of the user who initiated an event.</td>
</tr>
<tr>
<td>AUDIT_HOSTNAME</td>
<td>Name of the server from which an event was initiated.</td>
</tr>
<tr>
<td>AUDIT_IPADDRESS</td>
<td>IP address of the server from which an event was initiated.</td>
</tr>
<tr>
<td>AUDIT_PREVIOUS_VALUE</td>
<td>For an event that changed a value, this is the previous value.</td>
</tr>
<tr>
<td>AUDIT_NEW_VALUE</td>
<td>For an event that changed a value, this is the new value.</td>
</tr>
<tr>
<td>AUDIT_ERROR</td>
<td>For a failed event that produces an AKException, this is an error code. For a failed event that produces any other type of exception, this is the error message text.</td>
</tr>
<tr>
<td>AUDIT_REQUEST_TRANSACTION</td>
<td>The transaction ID of the request that initiated an event.</td>
</tr>
</tbody>
</table>

**Creating custom logging messages**

As you build and deploy your own app scopes, you will find it helpful to write and capture your own logging messages. The Akula logging mechanism relies on [SLF4J](https://slf4j.org) as an abstraction on Logback to let you create your own logging message.
To dispatch a logging message from your app scope (in a custom module, for example), create an instance of the SLF4J Logger class by using the LoggerFactory.

The following example creates a new instance of the Logger class:

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

private static Logger myLogger = LoggerFactory.getLogger(MyClass.class);
```

Then, use the Logger API to dispatch your logging messages, as shown below:

```java
myLogger.info("My message is: ", msg);
```

Using SSL with Akula

SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a client, such as a client browser or other type of client app. An encrypted link ensures that all data passed between the web server and client remains private.

When a client makes a secure request to the web server using SSL, the client specifies the protocol of the request as `https://`, instead of the unsecured `http://` protocol.

This document contains the following sections:

- SSL and the Akula Server
- SSL and Akula client apps
- Choosing a cipher suite
- Best practices for using SSL with Akula

SSL and the Akula Server

Because SSL is configured at the web server level, there is no configuration required on the Akula Server to support SSL. For example, you deploy the Akula Server on Tomcat, and also use Tomcat as your web server. Therefore, all SSL configuration is done in Tomcat. For more information on configuring SSL for Tomcat, see [http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html](http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html).

If you are not using Tomcat as your web server, then see the documentation on your web server for information and instructions on how to configure SSL.

Make sure that your web server uses an SSL protocol of “SSLv3+TLSv1”, “SSLv3”, or “TLSv1”. For strong encryption, do not use “SSLv2”.

SSL and Akula client apps

An Akula client app accesses the Akula Server by making requests to the REST endpoints exposed by the app.
scopes deployed on the Akula Server. An endpoint exposed by an Akula app scope consists of a URL and an HTTP access method, such as GET or POST. However, the protocol of the request, https:// or http://, is controlled by the web server through which you access the Akula Server.

The Akula Android, iOS, and JavaScript clients have built in support for SSL. That means you can make requests from an Akula client app to the Akula Server using SSL. You only have to make sure to specify the protocol of the request as https://. There is not other client configuration necessary to make SSL requests to the Akula Server.

The Akula clients only support SSL certificates from a Certificate Authority (CA) such as VeriSign or Thawte. Such certificates can be electronically verified by the CA. However, the Akula clients do not support self-signed certificates. Self-signed certificates are user generated and have not been registered with a CA and, therefore, cannot be verified.

Make sure that you choose a CA that is supported on all client devices, client device versions, client browsers, and client browser versions required by your client app. If a client does not support the CA used by the server, then the client will not be able to communicate with the server.

Choosing a cipher suite

Ciphers are the algorithms used to encrypt or hash data when communicating over SSL. A cipher suite is the set of ciphers that are used for all parts of communicating over SSL.

Characteristics of weak ciphers

Verivo recommends that you only use strong ciphers with SSL. To differentiate strong ciphers from weak, Verivo used the following criteria to identify weak ciphers:

- All anonymous versions
- All Export versions - Export versions of ciphers are often deliberately weak.
- All ciphers with null encryption
- Weak hash algorithms for the HMAC - However, ciphers using these weak algorithms might be needed to support older versions of Microsoft Windows.
- All ciphers using DES for symmetric encryption - The key size for DES is too small.
- All Kerberos ciphers - While Kerberos is secure, and can be used, most people do not run the server in a Kerberos environment.
- All ciphers using RC4 for symmetric encryption - RC4 is considered a weak cipher. However, ciphers using this algorithm might be needed to support older versions of Microsoft Windows.
- All non TLSv1 or SSLv3 ciphers - Any version prior to these are weak.
- All ciphers using 3DES for symmetric encryption - While it is considered secure, it is weaker and slower than RSA.

Recommended Cipher Suites

Verivo recommends the following asymmetric ciphers, in order from strongest to weakest:

- Elliptic Curve Diffie-Hellman Ephemeral with Elliptic Curve Digital Signature Algorithm (**)
- Elliptic Curve Diffie-Hellman Ephemeral with RSA (**)
- Elliptic Curve Diffie-Hellman with Elliptic Curve Digital Signature Algorithm (**)
- Elliptic Curve Diffie-Hellman with RSA (**)

© Copyright 2014, Verivo Software, Inc. All Rights Reserved.
Verivo recommends the following symmetric ciphers:

- AES with 128 bit keys - Some servers may have 256 bit AES support. If your system supports it, replace 128 with 256 in the cipher suites identifier.

Verivo recommends the following hashing algorithms, in order from strongest to weakest:

- SHA256
- SHA

Verivo recommends the following ciphers, in order from strongest to weakest:

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (**)
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (**)
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (**)
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (**)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (**)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (**)
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (**)
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (**)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (**)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (**)
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV (Required to support TLS renegotiation fix for vulnerability, as described here: http://www.oracle.com/technetwork/java/javase/documentation/tlsreadme2-176330.html)

** No CA currently uses Elliptic Curve encryption. However it is included because CAs will use them in the future.

Supporting Microsoft Windows XP and older Windows versions

Some older versions of Windows might require RC4 symmetric encryption or MD5 hashing, both of which are considered weak. If so, then you can use the following ciphers:

- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5

Best practices for using SSL with Akula

In a typical deployment environment, clients make all requests to the Akula Server using SSL. This is because all requests to the Akula Server, other than a login request, pass the user's session ID as part of the request to identify the user to the Akula Server. A login request does not pass a session ID but does pass user credentials, such as username and password, and therefore should also use SSL.

Also, many requests either pass data to the Akula Server, or receive data back from the server in the server response. To ensure that all data transactions are secure, make the requests by using SSL and the https:// protocol.
Using the Akula Command-line Management Utility

The Akula Command-line Management Utility lets you configure the Akula Server and app scopes running on the Akula Server. The Akula Command-line provides a simple way to write HTTP requests against the REST endpoints on the Akula Server used to manage app scopes. For a complete list of the REST endpoints, see REST API Reference.

Use the Akula Command-line to perform administration functions such as the following:

- **Configure roles and groups** – Create roles, associate permissions with a role, associate user groups with roles, and perform other operations on roles and groups.
- **Determine information about a realm** – List all groups in a realm and list all realms available to a role.
- **Obtain information about the Akula Server or an app scope** – Obtain configuration information about the Akula Server, determine which app scopes are enabled, and obtain information about an app scope.
- **Start and stop app scopes** – Start and stop all app scopes except the server app scope.
- **Licensing** – Install, update, and delete licenses.
- **App properties** – Set, update, and reset app properties.
- **Logging** – Set logging levels on a per-scope basis.

For all three supported platforms, Akula ships the Akula Command-line Management Utility as a Python application. For Microsoft Windows only, Akula ships the Akula Command-line Management Utility as the akula.exe file. For information on installing the Akula Command-line Management Utility, see Install the Akula Command-line Management Utility.

This document describes contains the following sections:

- Using the Akula Command-line
- Authorization required to run commands
- Example commands

### Using the Akula Command-line

To start the Akula Command-line:

1. Open a terminal window on your machine.
2. Change directory to the directory where the Akula Command-line is located.
   - For Microsoft Windows, the default location is the `server-sdk/tools/mgmt-util/bin` directory in the location where you installed Akula.
   - For all other platforms, the default location is the `server-sdk/tools/mgmt-util/src` directory.
3. **For Microsoft Windows only**, if you are using the akula.exe file, invoke the Akula Command-line and include the URL of the Akula Server on invocation, as the following example shows:

   ```bash
   akula.exe http://your.server.com/akula
   ```

   **For all platforms**, use Python to invoke the Akula Command-line from the command line. Include the URL of the Akula Server on invocation, as the following example shows:
3. "python akula.py http://your.server.com/akula"

where akula.py is the Python file for the Akula Command-line.

> Always connect to the URL of the Akula Server itself, not to a specific app scope running on the Akula Server. The Akula Command-line automatically uses the server app scope to perform its management functions.

After you start the Akula Command-line and successfully connect to the Akula Server, you see the Akula Command-line’s prompt:

> Enter commands to the Akula Command-line at the prompt. For example, enter help to see usage information.

4. If you deployed the default Akula Server configuration files, as described on Set up the Akula Server Environment, then log in to the Akula Server using the default Administrator account:

> login admin admin

Otherwise, you can log in with the credentials defined by your own security realm.

Authorization required to run commands

To use the Akula Command-line, you must first log in. After logging in, the server returns a list of permissions set for your account. You can only run the commands your account is authorized to use. If you run a command you are not authorized to use, the command line interface returns an HTTP 403 error - access forbidden.

For example, if you log in using an account that has only the "server.View_Roles" permission and then try to run the "roles server" command you will receive an HTTP 403 error. This is because your account can only view roles, but the "roles server" command returns permissions, groups, and realms in addition to roles. To run the command successfully, you would need permissions set to view the other three types of data.

Permissions are set either at the scope or server level. At scope level, a user can run commands for which permissions are set, but only on specified app scopes. The permissions set for a user can vary per app scope. At the server level, permissions for a user apply to all apps running on the server. However, if you use the command-line to send a request to a REST endpoint, the endpoint specifies the permissions required for access. For more information, see the REST API Reference.

Editing permissions requires the "server.Edit_Permissions" setting. The Akula Command-line Utility is shipped with a default administrator account that has that and all other permission settings enabled. To log in to that account, use login: admin and password: admin.

Example commands

This section contains examples of commands for the Akula Command-line.

Obtaining help
You can obtain help by entering the help command at the prompt:

> help

**Logging in**

The first command that you typically enter is the `login` command to log in to the Akula Server.

> login

Most other commands require the user to be authorized to perform administrative actions. Therefore, log in with the credentials of a server admin.

The `login` command has the following syntax:

```
login principal secret
```

as the following example shows:

> login smg@my.com myPWord

The principal is typically your username, and secret is your password.

If the credentials are correct, the Akula Command-line displays the following message:

```
Status: Akula Server Version: 2.5.0
Token: d8129a3f-icu8-1234-abcd-1becea2d866f
Timeout: 300000
User:
  Id: smg@my.com
  Realm: MyRealm
  Info:
    any_custom_realm_properties
Properties:
  any_app_properties
Permissions:
  any_user_permissions
```

**List the app scopes**

Use the `scopes` command to see a list of apps running on the Akula Server, as the following example shows:

> scopes

The output of this command is:
Scopes:

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name: server</th>
<th>Enabled: True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Name: App1</td>
<td>Enabled: True</td>
</tr>
<tr>
<td>Scope</td>
<td>Name: App2</td>
<td>Enabled: True</td>
</tr>
</tbody>
</table>

Where `server`, `App1`, and `App2` are the names of the apps. The `server` app is built into the Akula Server. `App1` and `App2` correspond to the names of the AKZ files that represent the apps. For example, `App1.akz` and `App2.akz`. For more information on packaging server apps, see Building and Deploying AKZ Files.

**Enable or disable an app scope**

You can use the Akula Command-line utility to enable or disable an app scope that has been deployed on the Akula Server.

To enable or disable an app scope, use the following command:

```plaintext
> updatescope scope_name [-enabled true|false]
```

To enable an app scope, set the `enabled` flag to `true`. To disable an app scope, set the `enabled` flag to `false`.

The following example enables the `MyApp` app scope:

```plaintext
> updatescope MyApp -enabled true
```

If successful, Akula returns an HTTP 200 OK.

If the Akula Server cannot enable or disable an app scope, the server returns one of the following errors:

- **SCOPE_NOT_FOUND (404)**
- **CANNOT_MODIFY_SERVER_SCOPE (403)**
- **NOT_PERMITTED (403)**

The following example shows the JSON response for a **SCOPE_NOT_FOUND** error:

```json
{
    "error": {
        "code": "SCOPE_NOT_FOUND",
        "description": "Scope not found",
        "information": {
            "name": "MyApp"
        }
    }
}
```
If you attempt to enable an app scope that is already enabled, or disable an app scope that is already disabled, Akula returns an HTTP 200 OK.

**List the roles, realms, and groups for a server app**

Many Akula Command-line commands require you to specify the scope name (corresponding to the app name), a role name, a group name, or a realm name. To determine the roles for a scope, use the command:

```plaintext
> roles App1
```

Where the argument to the roles command is a scope.

To determine the realms, and user groups in a realm, use:

```plaintext
> realms App1
```

To determine the groups defined in a realm for a scope, use the command:

```plaintext
> groups App1 MyRealm
```

**Updating roles**

Use many of the Akula Command-line commands to modify a role. For example, you can modify a role by:

- Adding, removing, or modifying the role
- Associating groups to a role or disassociating groups from a role
- Associating permissions to a role or disassociating permissions from a role

You can list the roles for an app, as shown below:

```plaintext
> roles App1
Roles:
```

Notice how the list of roles is blank.

Add a role to a scope by using the `addrole` command:

```plaintext
> addrole App1 newRole
```

Where `newRole` is the name of the role.

You can now see the new role:
Use the `addgroups` command to associate a user group defined in a realm with the role. The name of the group is the unique ID of the group. For example, if you are using Active Directory as your realm, it is a string in the form:

```
CN=Group Policy Creator Owners,CN=Users,DC=vvod,DC=com
```

The following example adds a group to the role you created. The realm name in this example is MyRealm:

```
> addgroups App1 newRole MyRealm "CN=Domain Admins,CN=Users,DC=vvod,DC=com"
```

You can now see the group in the role:

```
> roles App1
Roles:
  Role:
    Name: newRole
    Permissions:
    Realms:
      Realm:
        Name: MyRealm
        Groups:
        Id: CN=Domain Admins,CN=Users,DC=vvod,DC=com
```

Use the `addpermission` command to associate a permission with the role, as the following example shows:

```
> addpermission App1 newRole ToDo.read
```

You can now see the permission on the role:

```
> roles App1
Roles:
  Role:
    Name: newRole
    Permissions:
      Id: ToDo.read
    Realms:
      Realm:
        Name: MyRealm
        Groups:
        Id: CN=Domain Admins,CN=Users,DC=vvod,DC=com
```
To delete a role, use the `deleterole` command, as shown below:

> deleterole App1 newRole

### Using a command file

You can write a text file that contains a set of commands to the Akula Command-line, and then run the commands by using the `-s` option on the command line. The following example runs all the commands in the text file `MyCommands.txt` using the Microsoft Windows executable version of the Akula Command-line:

```bash
akula.exe -s MyCommands.txt http://myServer.myCom:8080/akula
```

For all platforms, use Python version of the Akula Command-line:

```bash
python akula.py -s MyCommands.txt http://myServer.myCom:8080/akula
```

### Upgrading the Akula Server

This document describes how to manually upgrade an Akula Server installation by using files downloaded from the Verivo DevCenter Downloads page (you must log in to access this page).

- These instructions apply to upgrading an Akula Server from version 2.0 to version 2.5. To upgrade from a previous release, consult the relevant upgrade instructions such as: 1.0 to 1.5 or 1.5 to 2.0.

- Within the 2.5 release package download, the Akula server is simply named "server.war". Rename this file during deployment to "akula.war".

### Upgrade overview

The Akula Server requires a database to store the configuration and run time information used by the server. The Akula Server includes an internal Apache Derby database that you can use to get up and running quickly in a development environment. While the Derby database is adequate for development work, it is not intended to function in a deployment environment. For deployment, Verivo recommends that you configure the Akula Server to use an external enterprise-class database, such as Microsoft SQL Server, MySQL, or Oracle.
The primary task for upgrading the Akula Server is to upgrade the database. On start up, the akula.war file automatically upgrades the database that it uses for configuration and run-time information. Therefore, most of the steps in the procedure below describe how to edit configuration files in the AKULA_HOME directory to ensure that the akula.war file has proper access to the database to perform the upgrade.

Akula Server upgrade procedure

1. Back up the existing database used to store the configuration and run time information used by the Akula Server. This is the database specified in the AKULA_HOME/global/context.properties file.

2. **(If you are using an external database only)** The db_jndi_name property in the AKULA_HOME/global/cont ext.properties file specifies the JNDI database resource for the Akula Server database. You therefore require a corresponding JNDI database resource definition on your JEE server or servlet container. For example, from the context.properties shown above, you require a resource named java:comp/env/jdbc/akula-db.

   If you are running the Akula Server on Tomcat, you would add the following resource definition to the Tomcat context.xml file:

   ```xml
   <Resource
     name="jdbc/akula-db"
     auth="Container"
     type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000"
     username="javauser" password="javaadmin"
     driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/akuladb"
   />
   
   The akula.war file automatically upgrades its database when you first start it. Therefore, ensure that the username specified in the JNDI database resource definition has permissions to the database so that the akula.war file can modify the database tables and table schemas.

3. Edit the Akula global properties file to add definitions to support features included in this version of Akula. The location and name of the file is: AKULA_HOME/global/properties.xml. Add the following change:

   - This version of Akula includes an auditing mode that allows built-in Akula database auditing when it is enabled. To enable the auditing setting, add the following definition for the akula_auditing_enabled property:

     ```xml
     <akula-property key="akula_auditing_enabled"
                     description="Enable built-in Akula database auditing"
                     display="Auditing Enabled"
                     isClientProperty="false"
                     overridable="false"
                     type="boolean">true</akula-property>
     
     For more information on using the akula_auditing_enabled key to enable or disable auditing, see "Auditing Mode".
     ```
Installing and Using the Akula Management Console

The Akula Management Console is a web-based user interface that provides a graphical overview of individual app scopes on an Akula Server, enables you to perform tasks that previously required using the command line interface, and offers an additional way to configure and send push notifications and actions queries.

The Akula Management Console is compatible with Akula Server version 2.0 and higher.

This section describes how to install and configure the Akula Server Management Console, and contains the following topics:

- Installing the Akula Management Console
- Using the Akula Management Console

Installing the Akula Management Console

To install the Akula Management Console, deploy the AkulaManagementConsole.war file to your Java application server or servlet container in the same location you deployed the Akula Server WAR file.

The Akula Management Console is compatible with Akula Server version 2.0 only.

This topic contains the following sections:

- Deploy the Akula Management Console on Tomcat
- Deploy the Akula Management Console on JBoss
- Deploy the Akula Management Console on Oracle WebLogic

Deploy the Akula Management Console on Tomcat

Tomcat lets you deploy a WAR file in several different ways, including:

- Startup deployment, where you have to manually start and stop the Tomcat server.
- Hot deployment, where you can deploy the WAR file without restarting Tomcat.
- Tomcat Web Application Manager is a browser-based utility used to manage Tomcat.

The following sections describe all three methods.

For more information on Tomcat, see http://tomcat.apache.org/download-70.cgi.

If this is the first time you have installed and used Tomcat, you might find it easiest to use startup deployment.

You must have the necessary system permissions to install and start Tomcat on your machine.

Using startup deployment on Tomcat
Startup, or folder, deployment requires that you restart Tomcat after you deploy a WAR file. Use the following procedure to install the AkulaManagementConsole.war file on the Apache Tomcat application server.

**To use startup deployment:**

1. Stop the Tomcat server by using the command `shutdown` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `shutdown.bat`
   - On Mac/Unix, run `./shutdown.sh`
2. Browse to directory where you unzipped the Akula Server ZIP file or where you saved the **AkulaManagementConsole.war** file.
3. Deploy the `server-sdk/bin/AkulaManagementConsole.war` file by copying it to the webapps directory of your Tomcat installation. If the `unpackWARs` option is set to `true`, the default, in the Tomcat conf/server.xml file, Tomcat automatically expands the WAR file when it starts. Otherwise, Tomcat runs the app directly from the WAR file.
4. Start the Tomcat server by using the command `startup` from the bin directory of your Tomcat installation directory based on your operating system:
   - On Microsoft Windows, run `startup.bat`
   - On Mac/Unix, run `./startup.sh`

**Using hot deployment on Tomcat**

With hot deployment, you can redeploy a WAR file or edit an existing application without restarting Tomcat.

To use hot deployment to deploy the AkulaManagementConsole.war file, make sure that the `autoDeploy` option is set to `true`, the default, in the Tomcat conf/server.xml file to enable hot deployment. Then, copy the WAR file to the Tomcat webapps directory.

If the `unpackWARs` option is also set to `true`, the default, in the Tomcat conf/server.xml file, Tomcat automatically expands the akula.war file.

**To use hot deployment:**

1. Browse to the `server-sdk/bin` directory where you unzipped the Akula Server ZIP file, or to where you saved the **AkulaManagementConsole.war** file.
2. Deploy the **AkulaManagementConsole.war** file by copying it to the webapps directory of your Tomcat installation. If the `unpackWARs` option is set to `true` in the Tomcat conf/server.xml file, you do not have to expand the WAR file. Tomcat will automatically expand it when it starts.

**Using the Tomcat Web Application Manager**

The Tomcat Web Application Manager is a console that lets you stop, deploy, undeploy, and reload an application.

To use the Tomcat Web Application Manager, you must create a user in the Tomcat conf/tomcat-users.xml file that has the proper role, as the following example shows:

```xml
<role rolename="manager-gui"/>
<user username="tcAdmin" password="tcAdminPW" roles="manager-gui"/>
```

After creating the user, restart Tomcat and then you can access the Tomcat Web Application Manager at the following URL:
http://localhost:8080/manager/html

Port 8080 is the default port, but it can be different, depending on your configuration.

If the Akula Management Console has been deployed correctly, it will appear in the list of available apps along with the default applications, as the following example shows:

After deploying the Akula Management Console, see Using the Akula Management Console.

Deploy the Akula Management Console on JBoss

This section describes how to deploy the AkulaManagementConsole.war file on JBoss EAP Version 6.1. This procedure assumes that you are running JBoss in standalone mode, which is typical for a development environment. If you are running JBoss in domain mode, see the JBoss documentation for deployment information.

This procedure describes starting and stopping JBoss from the command line. You can also start and stop JBoss, and deploy WAR files by using the JBoss Management Console.

Use the following procedure to deploy the AkulaManagementConsole.war file on JBoss in standalone mode:

1. Ensure that you have already set the JBOSS_HOME environment variable to the home directory of JBoss.
2. Change directories to the JBoss bin directory.
3. Run ./standalone.sh to start the JBoss server in standalone mode. This command uses the standalone.xml file for running the JBoss server instance.

To access JBoss on a remote machine, specify the -b parameter to standalone.sh script with the IP address of the machine:

./standalone.sh -b 172.17.3.201

4. Copy the AkulaManagementConsole.war file from the location where you installed Akula to JBOSS_HOME/standalone/deployments directory.

   If the autoDeploy attribute is true, the default value, JBoss automatically deploys the WAR file and create
server.war.deployed file. If the autoDeploy attribute is false, you have to restart JBoss.

If you see any errors during deployment, check the JBoss logs/server.log and boot.log files.

After deploying the Akula Management Console, see Using the Akula Management Console.

Deploy the Akula Management Console on Oracle WebLogic

This section describes how to deploy the AkulaManagementConsole.war file on Oracle WebLogic 12.1. Deploy the WAR file in the same domain where you installed the akula.war file, as described in Deploy the Akula WAR file on your Java Server.

To deploy the AkulaManagementConsole.war file on WebLogic:

1. Stop the WebLogic server.
2. Navigate to the directory where you unzipped the Akula Server ZIP file or to where you saved the AkulaManagementConsole.war file.
3. Copy the WAR file to the directory where you deployed the akula.war file. The example directory used in this documentation is $MW_HOME/user_projects/domains/akula.
4. Start WebLogic.

After deploying the Akula Management Console, see Using the Akula Management Console.

Using the Akula Management Console

The Akula Management Console enables you to manage individual app scopes on an Akula Server. For each app scope, you can create and send push notifications and client actions, view push and action history, override default app properties, create roles using existing groups and permissions, and create log in and sync client rules.

The initial App Scope page displays whether each app scope is enabled or disabled. If an app scope is disabled you will receive errors if you try to view or edit its properties. To change the status of an app scope use the Overview page for the app scope within the web management console, the App Scope Management REST API, or the Akula CLI command updatescpe.
To view tool tips in the Akula Management Console, hover the mouse over any Help icon: 🎫.

The Akula Management Console consists of the following menu options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>Displays a visual representation of push notifications and action requests and used to enable/disable the app scope (top right corner of the page). Hover the mouse over a graph bar or segment to view the corresponding numerical data. The information is current for when you opened the page. The page does not update automatically to reflect any data changes that occur while you are viewing it. To update the page, click the reload button in your browser.</td>
</tr>
</tbody>
</table>
| General | There are two pages under the General menu option:  
- App Properties - Lists app properties, their default values, and current override settings. From this page you can override the default value for a property when specified conditions are met. Select a property to view its description and settings. To add or edit an override condition, click the edit icon next to the value in the Scope Override section. Then, set the condition under which the override occurs by clicking the Add Override button.  
- Logging - Set the log levels for the app scope.  
For general information about app properties, see Using App Properties. For information specifically about CORS properties, see Allow Cross Origin Requests in a Browser App. |
| Security | There are two pages under the Security menu option:  
- Roles - Lists existing roles and enables you to add new ones. You can associate each role with one or more groups and permissions. Groups and permission types must be defined for a scope before you can associate them with a role using the console. For more information about roles and how to create new groups and permission types, see Authorizing Users.  
- Credentials - Lists existing scope credentials that all users of the app scope share. Scope credentials can only be set from the Akula Command-line, or by making requests to the REST API as an administrator. For more information, see Using Credentials in a Custom Module. |
### Rules

Enables you to view and edit client rules. The templates you can use to define new rules are listed on the left side of the page. Select a template to view or edit existing rules that use it, or to define a new one. The arguments required for a rule are described in the Arguments table for each rule. The description for each rule is generated automatically using the argument values you enter. For more information, see [Applying Client Rules](#).

### Push

There are three pages under the Push menu option:

- **Create** - Provides an interface for creating and sending push notifications.
- **History** - Displays a history of push notifications that you can filter by start and end dates.
- **Configuration** - Enables you to configure APNS and GCM.

For more information, see [Overview of Push](#).

### Action

There are two pages under the Action menu option:

- **Create** - Provides an interface for creating and sending client actions. You can select a built-in action or specify a custom one.
- **History** - Displays a history of actions that you can filter by start and end dates. Each history entry shows the number of devices in a given state for that action. The states are color-coded: ignored or canceled in grey, failed in red, completed in green, and created or received in orange.

For more information, see [Overview of Client Actions](#).

---

### Viewing and Overriding App Properties

The App Properties page in the Management Console displays the default values of app properties and enables you to override them for all users of an app, and to override them for individual users and user roles.

The properties consist of the following:

- The values that enable cross-origin requests (CORS)
- Client rules consisting of those for sync and for blocking clients from logging in after failed attempts
- Security manager settings
- Whether the server is in development or production mode
- Any custom app properties you define

The order of precedence for properties is as follows (highest to lowest): An override value for a user > an override value for a role > an app scope override value > an app scope default value > the server-level value

Configuration settings for the server are defined in the AKULA_HOME/global/properties.xml file. Configuration settings for apps are defined in a properties.xml file you include in the app's AKZ file. Any custom properties you define must reside in the same files as the Akula properties.
Properties must be defined in a configuration file before you can manage them in the console, you cannot use the console to add new ones. If you edit the configuration files to set values you must restart the server. However, you can use the Management Console to enter override values while the server is running. Any overrides you enter using the Management Console are stored internally and do not affect corresponding properties.xml files.

For detailed information about app properties, see Using App Properties.

This document contains the following sections:

- Prerequisites
- Override an app property for a scope
- Override an app property for a user or user role
- Related Akula CLI commands

Prerequisites

Before you override an app property:

- Add a user role if you want to override a property by role.
- Ensure you have a realm defined for a user if you want to override a condition by user name.
- If applicable, define any custom properties you want to use.

Override an app property for a scope

To override an app logging property for a scope:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click General and then App Properties. A list of default app properties along with any custom properties appears.
3. Scroll through the list of properties and click the name of the property you want to manage.
4. In the Scope Override section of the property's settings, click the pencil icon.
5. Enter a value in the Scope Override dialog box, and click OK.

If you want to remove an override later, click the Trash icon.

Override an app property for a user or user role

To override a property for a user or role:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click General and then App Properties. A list of default app properties along with any custom properties appears.
3. Scroll through the list of properties and click the name of the property you want to manage.
4. In the Override Conditions section of the property's settings, click the Add Override button.
5. In the dialog box, select Role or User. Either available roles or available user realms appear.
6. Depending on your selection, choose the role whose users' value you want to override; or select the realm of the individual user whose value you want to override. To override the value for a user you must also enter the user's login name.
7. Enter an override value for the property and click the Add button.

If you want to remove an override later, click the Trash icon.
Related Akula CLI commands

- properties
- addrole
- realms

**Viewing and Overriding Logging Levels**

The Logging page displays loggers that are configured for an app and their current logging level. You can use this page to change the log level of an existing logger, but not to create a new one. New loggers are created by adding a logback.xml configuration file to an AKZ file before you deploy it.

XML configuration files determine how Akula writes log files at both the server level and app level. Default configuration settings for the server are provided in the AKULA_HOME/global/logback.xml file. You can edit that file to change server log settings or use it as a guide for creating app-level logback.xml files. Changing log settings by editing the configuration file requires restarting the server. However, you can use the Logging page to override app logging settings while the server is running. Any overrides you enter using the Management Console are stored internally and do not affect corresponding logback.xml files.

This document contains the following sections:

- Prerequisite
- View or override a logging level
- Related Akula CLI commands

**Prerequisite**

Configure Logging

**View or override a logging level**

To view or override an app's logging level:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click General and then Logging. A list of configured loggers appears along with their current logging level.
3. To change a level, click the current setting and select a new one from the list.

**Related Akula CLI commands**

Logging

**Manage Roles, Groups, and Permissions**

The Roles page displays the roles defined for an app scope. Click a role to display its associated groups and permissions. You can then add or remove groups and set permissions. Groups are defined in a security manager realm outside of the Akula server, such as on an Active Directory Server. A realm is a collection of users and groups that the Akula server can authorize as valid. You specify the names and connection information for the realms you
want to use in the security-template.xml file in the AKULA_HOME/global directory. The permissions are defined for an app scope in its permissions.xml configuration file. The name of each permission is arbitrary. The action enabled by a permission is defined by pairing the permission name with an action URL in the apiendpoint.xml configuration file.

The default security manager uses a built-in Apache Derby database to store authentication information. Two groups are included in the default database - Users and Administrators.

Configuration files for an app scope must be included in the app's AKZ file before you deploy it. Any changes you make to the files after you deploy an AKZ file will be overwritten whenever the server restarts, unless you remove the AKZ file from the deploy directory before restarting. Realms, groups, and permissions must be defined in configuration files before you can manage them in the console, you cannot use the console to add new ones. You can edit the configuration files to change default values, but then you must restart the server. However, you can use the Management Console to enter override values while the server is running. Any overrides you enter using the Management Console are stored internally and do not affect corresponding configuration files.

For detailed information about roles, see Authorizing Users.

This document contains the following sections:

- Add a role
- Configure permissions for a role
- Related Akula CLI commands

Add a role

To add a role:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Security and then Roles. A list of existing roles appears, along with groups and permissions.
3. Above the Roles list, click the Add button (+).
4. In the Add Role window, enter a role name and click Add.

Configure permissions for a role

To configure permissions for a role:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Security and then Roles. A list of existing roles appears, along with groups and permissions.
3. Click the role you want to configure. Its current permissions settings appear.
4. Select the checkbox next to each permission you want to give to users in that role; clear the remaining checkboxes.
5. Click Save.

Related Akula CLI commands

addrole, deleterole, role, roles
addpermission, deletepermission, permissions
addgroup, deletegroup, group, groups
reamls

Setting Credentials

The Credentials page lists the credentials defined for an app scope. Credentials are defined in the credentials.xml file in an app's AKZ file within the AKZ-INF/config/scope directory. Credentials enable you to define, but not set, authentication credentials for an app scope to access back-end data. You can also define user credentials, however they are not accessible in the Web Management Console. Using the console, you can set the principal and secret of a scope credential during runtime. You cannot use the console to define new types of credentials. The principal and secret are stored internally by the Akula server.

For more detailed information, see Using Credentials in a Custom Module.

This document contains the following sections:

- Set scope credentials
- Related Command-line Management Utility commands

Set scope credentials

To view or set credentials:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Security and then Credentials. A list of predefined credentials appears along with their status - configured or not set.
3. To set a credential, scroll through the list of credentials and click the name of the one you want to manage.
4. In the Values section, click the pencil icon.
5. Enter new values in the dialog box, and click SAVE.

If you want to remove credentials later, click the Trash icon.

Related Command-line Management Utility commands

credentials, deletecredential, updatecredential

Setting Rules to Block Client Sync and Login

In an enterprise production environment, you often apply policies, or rules, to client apps and the users of those apps. These rules can control the action of the client app in response to events in the app, events caused by a user action, or events caused by the device itself. The Akula Server ships with a set of predefined rules that you can configure by using the Akula Command-line Management Utility or by using the Akula REST API.

Working with client rules does not require you to write any client-side code. Instead, you create client rules from predefined rule templates located on the Akula Server. On a successful log in to an app scope from a client app, the rules are automatically sent down to the client from the Akula Server.

For more information, see Client Rules Overview.

Set a rule to block logging in to block sync
The procedure for setting a client rule is the same for blocking log in as it is for blocking sync.

When you create a client rule to block sync, you can specify to block sync for all entities, or for a specific set of entities, where the name of the entity corresponds to the class name that defines the syncable object.

To set rules to block sync:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Rules and then Client Rules. A list of client rule templates appears.
3. Click the rule you want to use as the basis for a new one. Its default settings and any rules currently based on it appear.
4. Click the Add Rule button at the top-right of the screen. A dialog box for the rule appears.
5. Using the arguments described on the rule’s template page, complete the dialog box. The rule key can be any string that is unique across all client rules.

Later, to edit or remove a rule, use the pencil or trash bin icons respectively.

Related Akula CLI commands

createrulefromtemplate, deleterule, updaterulefromtemplate

Configuring, Sending, and Viewing Push Notifications

Use the Push menu options to configure APNS and GCM for push, create a push notification, and to view a history of push notifications. To configure APNS requires a p.12 certificate. Each app scope requires its own certificate.

Configuring GCM requires an API key.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Badge</td>
<td>The number displayed as the badge of an app icon. If this property is absent, the badge is not changed. To remove the badge, set the value of this property to 0.</td>
</tr>
</tbody>
</table>

Creating and Viewing Actions

The Create Action page enables you to use built-in Akula actions, or your own custom actions, to perform actions on client devices. In order to create an action, a client device must be already known by its device token to the Akula server. Then, you can group users by role, realm, platform type, or individually by name to receive client actions.

The Action History page displays a textual and visual history of actions that were created and their current status.

This document contains the following sections:
Prerequisites

Before you can create an action:

- You must have devices registered with device tokens on the Akula server.
- Optionally, you can define roles and define users and realms to associate with registered devices.
- Optionally, you can create a custom action you want to send. The names of custom actions do not appear in the Web Management Console so you must know their name to use them.

To create an action

To create an action:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Action and then Create. The Create Action page appears.
3. On the Recipients of the Action section, select one of the following tabs and follow the procedure in the Steps column:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>By Role</td>
<td>A list of known roles. If you select this tab, use the Preview button to verify that there are any devices are registered for that role.</td>
<td>If you select this tab, use the Preview button to verify that there are any devices are registered for that role.</td>
</tr>
<tr>
<td>By User</td>
<td>A list of of individual recipients to receive an action.</td>
<td>To add users, click the Add User button. In the Add User window, you can select any realm configured for Akula. Security realms are defined in the security-template.xml file (described here: Defining a Security Manager).</td>
</tr>
<tr>
<td>By Platform</td>
<td>Enables you to send an action to all registered devices by platform. Android devices, iOS devices, or both.</td>
<td>Select Any Platform, iOS, Android, Cordova iOS, or Cordova Android. Then, click Preview to see a list of known devices for the selection you chose.</td>
</tr>
</tbody>
</table>
3. By Device
   Enables you to add action recipients based on both platform and role.
   To add recipients, click Add Devices. In the Select Recipients window, specify which platforms and which role to search for recipient devices. Click Search. A list of registered devices fitting the criteria appears. Select the checkbox next to each device you want to send an action to. Click Add Devices to close the window.

4. Advanced
   Use this tab to write JSON objects to enter the request parameters for creating an action.

5. For Action Type, select whether you want to create an Akula built-in action or a custom action.
6. For Command field, either select the name of a built-in action or enter the name of the custom action you want to use.
7. If using a custom action, if you want to include custom data when you create the action, enter it in the Custom Data field.
8. For the Time To Live fields, enter the length of time you want the action to remain on the server and attempt to complete sending it all intended recipients.
9. Click the Create Client Action button.

View action history

To view action history:

1. In the Management Console, select the scope you want to manage from the App Scopes drop-down list at the top-left of the screen.
2. From the side menu, click Action and then History. A list of actions appears along with a details page for the first action in the list.
3. To search for actions between a start date and end date, enter the dates in the two fields provided at the top of the list of action events and press the Filter button. Later, to remove the dates, you can delete them directly by highlighting their contents and using the Delete key on your keyboard.
4. Scroll through the list of actions and click the name of each action you want to view. The color codes under each action entry are described in the next section.
5. Alternatively, you can display results by their status. To do so, in the Action Command Results section at the bottom of the right-side panel, select a type from the View results by status list. The number next to each status type indicates the number of actions currently in that state.
6. When more than 10 actions are in your history, the records will be split into pages, with the first page showing the most recent 10 records. Buttons for "Older" and "Newer" will appear when applicable to allow navigation through the history.

Action History Color Codes

The following table describes the color codes on the Action History page. The number inside each block indicates the number of devices corresponding currently in that state.

<table>
<thead>
<tr>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Action request is expired, cancelled, or ignored</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>Action was sent but returned an error</td>
</tr>
<tr>
<td>0</td>
<td>Action was successful</td>
</tr>
<tr>
<td>0</td>
<td>Action is in queue.</td>
</tr>
</tbody>
</table>

**Related CLI commands**

actioncommandlogs, actionlogs, deleteaction, sendaction