DHIS 2 System Administration guide

Applicable to 2.33 version

DHIS 2 Documentation team
February 2020
Copyright © 2006-2020 DHIS 2 Documentation team

February 2020

<table>
<thead>
<tr>
<th>Revision History</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.33@fca034d</td>
<td>2020-02-27 11:02:03 +0100</td>
</tr>
</tbody>
</table>

**Warranty:** THIS DOCUMENT IS PROVIDED BY THE AUTHORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS MANUAL AND PRODUCTS MENTIONED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**License:** Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the source of this documentation, and is available here online: http://www.gnu.org/licenses/fdl.html
Table of Contents

1 About this guide
2 Installation
   2.1 Introduction
   2.2 Server specifications
   2.3 Software requirements
   2.4 Server setup
      2.4.1 Creating a user to run DHIS2
      2.4.2 Creating the configuration directory
      2.4.3 Setting server time zone and locale
      2.4.4 PostgreSQL installation
      2.4.5 PostgreSQL performance tuning
      2.4.6 System configuration
      2.4.7 Java installation
      2.4.8 Tomcat and DHIS2 installation
      2.4.9 Running DHIS2
   2.5 Base URL configuration
   2.6 File store configuration
   2.7 Google service account configuration
   2.8 LDAP configuration
   2.9 Encryption configuration
      2.9.1 Java Cryptography Extension
      2.9.2 Password configuration
      2.9.3 Considerations for encryption
   2.10 Read replica database configuration
   2.11 Web server cluster configuration
      2.11.1 Clustering overview
      2.11.2 Cluster instance configuration
      2.11.3 Cluster shared data store configuration
      2.11.4 Load balancing
   2.12 Analytics cache configuration
   2.13 Starting Tomcat at boot time
   2.14 Reverse proxy configuration
      2.14.1 Basic nginx setup
      2.14.2 Enabling SSL with nginx
      2.14.3 Enabling caching and SSL with nginx
      2.14.4 Rate limiting with nginx
      2.14.5 Additional resources on SSL
      2.14.6 Making resources available with nginx
      2.14.7 Basic reverse proxy setup with Apache
      2.14.8 SSL encryption with Apache
   2.15 DHIS2 configuration reference
   2.16 Application logging
      2.16.1 Log files
      2.16.2 Log configuration
   2.17 Working with the PostgreSQL database
   2.18 DHIS2 Live setup
1 About this guide

The DHIS2 documentation is a collective effort and has been developed by the development team and users. While the guide strives to be complete, there may be certain functionalities which have been omitted or which have yet to be documented. This section explains some of the conventions which are used throughout the document.

DHIS2 is a browser-based application. In many cases, screenshots have been included for enhanced clarity. Shortcuts to various functionalities are displayed such as Data element > Data element group. The “>” symbol indicates that you should click Data element and then click Data element group in the user interface.

Different styles of text have been used to highlight important parts of the text or particular types of text, such as source code. Each of the conventions used in the document are explained below.

- **Note**
  A note contains additional information which should be considered or a reference to more information which may be helpful.

- **Tip**
  A tip can be a useful piece of advice, such as how to perform a particular task more efficiently.

- **Important**
  Important information should not be ignored, and usually indicates something which is required by the application.

- **Caution**
  Information contained in these sections should be carefully considered, and if not heeded, could result in unexpected results in analysis, performance, or functionality.

- **Warning**
  Information contained in these sections, if not heeded, could result in permanent data loss or affect the overall usability of the system.

Program listings usually contain some type of computer code. They will be displayed with a shaded background and a different font.

Commands will be displayed in bold text, and represent a command which would need to be executed on the operating system or database.

Links to external web sites or cross references will be displayed in blue text, and underlined like this.
2 Installation

The installation chapter provides information on how to install DHIS2 in various contexts, including online central server, offline local network, standalone application and self-contained package called DHIS2 Live.

2.1 Introduction

DHIS2 runs on all platforms for which there exists a Java Runtime Environment version 8 or higher, which includes most popular operating systems such as Windows, Linux and Mac. DHIS2 runs on the PostgreSQL database system. DHIS2 is packaged as a standard Java Web Archive (WAR-file) and thus runs on any Servlet containers such as Tomcat and Jetty.

The DHIS2 team recommends Ubuntu 16.04 LTS operating system, PostgreSQL database system and Tomcat Servlet container as the preferred environment for server installations.

This chapter provides a guide for setting up the above technology stack. It should however be read as a guide for getting up and running and not as an exhaustive documentation for the mentioned environment. We refer to the official Ubuntu, PostgreSQL and Tomcat documentation for in-depth reading.

The dhis2-tools Ubuntu package automates many of the tasks described in the guide below and is recommended for most users, especially those who are not familiar with the command line or administration of servers. It is described in detail in a separate chapter in this guide.

2.2 Server specifications

DHIS2 is a database intensive application and requires that your server has an appropriate amount of RAM, number of CPU cores and a fast disk. These recommendations should be considered as rules-of-thumb and not exact measures. DHIS2 scales linearly on the amount of RAM and number of CPU cores so the more you can afford, the better the application will perform.

- **RAM**: At least 1 GB memory per 1 million captured data records per month or per 1000 concurrent users. At least 4 GB for a small instance, 12 GB for a medium instance.
- **CPU cores**: 4 CPU cores for a small instance, 8 CPU cores for a medium or large instance.
- **Disk**: Ideally use an SSD. Otherwise use a 7200 rpm disk. Minimum read speed is 150 Mb/s, 200 Mb/s is good, 350 Mb/s or better is ideal. In terms of disk space, at least 60 GB is recommended, but will depend entirely on the amount of data which is contained in the data value tables. Analytics tables require a significant amount of disk space. Plan ahead and ensure that your server can be upgraded with more disk space as it becomes needed.

2.3 Software requirements

Later DHIS2 versions require the following software versions to operate.

- Java JDK or JRE version 8 or later.
- Any operating system for which a Java JDK or JRE version 8 exists.
- PostgreSQL database version 9.6 or later.
- PostGIS database extension version 2.2 or later.
- Tomcat servlet container version 8.5.50 or later, or other Servlet API 3.1 compliant servlet containers.
2.4 Server setup

This section describes how to set up a server instance of DHIS2 on Ubuntu 16.04 64 bit with PostgreSQL as database system and Tomcat as Servlet container. This guide is not meant to be a step-by-step guide per se, but rather to serve as a reference to how DHIS2 can be deployed on a server. There are many possible deployment strategies, which will differ depending on the operating system and database you are using, and other factors. The term *invoke* refers to executing a given command in a terminal.

For a national server the recommended configuration is a quad-core 2 Ghz processor or higher and 12 Gb RAM or higher. Note that a 64 bit operating system is required for utilizing more than 4 Gb of RAM.

For this guide we assume that 8 Gb RAM is allocated for PostgreSQL and 8 GB RAM is allocated for Tomcat/JVM, and that a 64-bit operating system is used. If you are running a different configuration please adjust the suggested values accordingly! We recommend that the available memory is split roughly equally between the database and the JVM. Remember to leave some of the physical memory to the operating system for it to perform its tasks, for instance around 2 GB. The steps marked as *optional*, like the step for performance tuning, can be done at a later stage.

### 2.4.1 Creating a user to run DHIS2

You should create a dedicated user for running DHIS2.

#### Important

You should not run the DHIS2 server as a privileged user such as root.

Create a new user called dhis by invoking:

```
sudo useradd -d /home/dhis -m dhis -s /bin/false
```

Then to set the password for your account invoke:

```
sudo passwd dhis
```

Make sure you set a strong password with at least 15 random characters.

### 2.4.2 Creating the configuration directory

Start by creating a suitable directory for the DHIS2 configuration files. This directory will also be used for apps, files and log files. An example directory could be:

```
mkdir /home/dhis/
    config
chown dhis:dhis /home/dhis/
    config
```

DHIS2 will look for an environment variable called `DHIS2_HOME` to locate the DHIS2 configuration directory. This directory will be referred to as `DHIS2_HOME` in this installation guide. We will define the environment variable in a later step in the installation process.
2.4.3 Setting server time zone and locale

It may be necessary to reconfigure the time zone of the server to match the time zone of the location which the DHIS2 server will be covering. If you are using a virtual private server, the default time zone may not correspond to the time zone of your DHIS2 location. You can easily reconfigure the time zone by invoking the below and following the instructions.

```
sudo dpkg-reconfigure
tzdata
```

PostgreSQL is sensitive to locales so you might have to install your locale first. To check existing locales and install new ones (e.g. Norwegian):

```
locale
  a
sudo locale-gen
  nb_NO.UTF-8
```

2.4.4 PostgreSQL installation

Install PostgreSQL by invoking:

```
sudo apt-get install postgresql-10 postgresql-contrib-10 postgresql-10-postgis-2.4
```

Create a non-privileged user called dhis by invoking:

```
sudo -u postgres createuser -SDRP dhis
```

Enter a secure password at the prompt. Create a database by invoking:

```
sudo -u postgres createdb -O dhis dhis2
```

Return to your session by invoking exit You now have a PostgreSQL user called dhis and a database called dhis2.

The PostGIS extension is needed for several GIS/mapping features to work. DHIS 2 will attempt to install the PostGIS extension during startup. If the DHIS 2 database user does not have permission to create extensions you can create it from the console using the postgres user with the following commands:

```
sudo -u postgres psql -c "create extension postgis;" dhis2
```

Exit the console and return to your previous user with \q followed by exit.

2.4.5 PostgreSQL performance tuning

Tuning PostgreSQL is necessary to achieve a high-performing system but is optional in terms of getting DHIS2 to run. PostgreSQL is configured and tuned through the postgresql.conf file which can be edited like this:

```
sudo nano /etc/postgresql/10/main/postgresql.conf
```
and set the following properties:

```yaml
max_connections = 200
```

Determines maximum number of connections which PostgreSQL will allow.

```yaml
shared_buffers = 3200MB
```

Determines how much memory should be allocated exclusively for PostgreSQL caching. This setting controls the size of the kernel shared memory which should be reserved for PostgreSQL. Should be set to around 40% of total memory dedicated for PostgreSQL.

```yaml
work_mem = 20MB
```

Determines the amount of memory used for internal sort and hash operations. This setting is per connection, per query so a lot of memory may be consumed if raising this too high. Setting this value correctly is essential for DHIS2 aggregation performance.

```yaml
maintenance_work_mem = 512MB
```

Determines the amount of memory PostgreSQL can use for maintenance operations such as creating indexes, running vacuum, adding foreign keys. Increasing this value might improve performance of index creation during the analytics generation processes.

```yaml
effective_cache_size = 8000MB
```

An estimate of how much memory is available for disk caching by the operating system (not an allocation) and isdb.no used by PostgreSQL to determine whether a query plan will fit into memory or not. Setting it to a higher value than what is really available will result in poor performance. This value should be inclusive of the shared_buffers setting. PostgreSQL has two layers of caching: The first layer uses the kernel shared memory and is controlled by the shared_buffers setting. PostgreSQL delegates the second layer to the operating system disk cache and the size of available memory can be given with the effective_cache_size setting.

```yaml
checkpoint_completion_target = 0.8
```

Sets the memory used for buffering during the WAL write process. Increasing this value might improve throughput in write-heavy systems.

```yaml
synchronous_commit = off
```

Specifies whether transaction commits will wait for WAL records to be written to the disk before returning to the client or not. Setting this to off will improve performance considerably. It also implies that there is a slight delay between the transaction is reported successful to the client and it actually being safe, but the database state cannot be corrupted and this is a good alternative for performance-intensive and write-heavy systems like DHIS2.
**wal_writer_delay = 10000ms**

Specifies the delay between WAL write operations. Setting this to a high value will improve performance on write-heavy systems since potentially many write operations can be executed within a single flush to disk.

**random_page_cost = 1.1**

*SSD only.* Sets the query planner’s estimate of the cost of a non-sequentially-fetched disk page. A low value will cause the system to prefer index scans over sequential scans. A low value makes sense for databases running on SSDs or being heavily cached in memory. The default value is 4.0 which is reasonable for traditional disks.

**max_locks_per_transaction = 96**

Specifies the average number of object locks allocated for each transaction. This is set mainly to allow upgrade routines which touch a large number of tables to complete.

Restart PostgreSQL by invoking the following command:

```
sudo /etc/init.d/postgresql restart
```

### 2.4.6 System configuration

The database connection information is provided to DHIS2 through a configuration file called `dhis.conf`. Create this file and save it in the `DHIS2_HOME` directory. As an example this location could be:

```
sudo -u dhis nano /home/dhis/config/dhis.conf
```

A configuration file for PostgreSQL corresponding to the above setup has these properties:

```
# --------------------------------------------------------------------
# Database connection
# --------------------------------------------------------------------

# Hibernate SQL dialect
connection.dialect = org.hibernate.dialect.PostgreSQLDialect

# JDBC driver class
connection.driver_class = org.postgresql.Driver

# Database connection URL
connection.url = jdbc:postgresql:dhis2

# Database username
connection.username = dhis

# Database password
connection.password = xxxx
```

---

2 Installation 2.4.6 System configuration
# Server
# Enable secure settings if system is deployed on HTTPS, default 'off'
server.https = on

# Server base URL
server.base.url = http://server.com/

It is strongly recommended to enable the server.https setting and deploying DHIS 2 over the encrypted HTTPS protocol. This setting will enable e.g. secure cookies. HTTPS deployment is required when enabled.

The server.base.url setting refers to the URL which the system is accessed by end users on the network.

Note that the configuration file supports environment variables. This means that you can set certain properties as environment variables and have them resolved by DHIS 2, e.g. like this where DB_PASSWD is the name of the environment variable:

```
connection.password = ${DB_PASSWD}
```

A common mistake is to have a white-space after the last property value so make sure there is no white-space at the end of any line. Also remember that this file contains the clear text password for your DHIS2 database so it needs to be protected from unauthorized access. To do this invoke the following command which ensures that only the dhis user which owns the file is allowed to read it:

```
chmod 0600
dhis.conf
```

### 2.4.7 Java installation

The recommended Java JDK for DHIS 2 is OpenJDK 8. You can issue the following command to install OpenJDK 8:

```
sudo apt-get install openjdk-8-jdk
```

Verify that your installation is okay by invoking:

```
java -version
```

### 2.4.8 Tomcat and DHIS2 installation

To install the Tomcat servlet container we will utilize the Tomcat user package by invoking:

```
sudo apt-get install tomcat8-user
```

This package lets us easily create a new Tomcat instance. The instance will be created in the current directory. An appropriate location is the home directory of the dhis user:
This will create an instance in a directory called `tomcat-dhis`. Note that the `tomcat7-user` package allows for creating any number of dhis instances if that is desired.

Next edit the file `tomcat-dhis/bin/setenv.sh` and add the lines below. The first line will set the location of your Java Runtime Environment, the second will dedicate memory to Tomcat and the third will set the location for where DHIS2 will search for the `dhis.conf` configuration file. Please check that the path the Java binaries are correct as they might vary from system to system, e.g. on AMD systems you might see `/java-8-openjdk-amd64` Note that you should adjust this to your environment:

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64/"
export JAVA_OPTS="-Xmx7500m -Xms4000m"
export DHIS2_HOME="/home/dhis/config"
```

The Tomcat configuration file is located in `tomcat-dhis/conf/server.xml`. The element which defines the connection to DHIS is the `Connector` element with port 8080. You can change the port number in the `Connector` element to a desired port if necessary. The `relaxedQueryChars` attribute is necessary to allow certain characters in URLs used by the DHIS2 front-end.

```
<Connector
  port="8080"
  protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443"
  relaxedQueryChars="[]" />
```

The next step is to download the DHIS2 WAR file and place it into the `webapps` directory of Tomcat. You can download the DHIS2 version 2.31 WAR release like this (replace 2.31 with your preferred version if necessary):

```
wget https://releases.dhis2.org/2.31/
dhis.war
```

Alternatively, for patch releases, the folder structure is based on the patch release ID in a subfolder under the main release. E.g. you can download the DHIS2 version 2.31.1 WAR release like this (replace 2.31 with your preferred version, and 2.31.1 with you preferred patch, if necessary):

```
wget https://releases.dhis2.org/2.31/2.31.1/dhis.war
```

Move the WAR file into the Tomcat `webapps` directory. We want to call the WAR file `ROOT.war` in order to make it available at localhost directly without a context path:
DHIS2 should never be run as a privileged user. After you have modified the setenv.sh file, modify the startup script to check and verify that the script has not been invoked as root.

```bash
#!/bin/
sh
set -e

if [ "$" (id -u) eq "0" ]; then
  echo "This script must NOT be run as root" 1>&2
  exit 1
fi

export CATALINA_BASE="/home/dhis/tomcat-dhis"
/usr/share/tomcat8/bin/startup.sh
echo "Tomcat started"
```

2.4.9 Running DHIS2

DHIS2 can now be started by invoking:

```
sudo -u dhis tomcat-dhis/bin/startup.sh
```

**Important**

The DHIS2 server should never be run as root or other privileged user.

DHIS2 can be stopped by invoking:

```
sudo -u dhis tomcat-dhis/bin/shutdown.sh
```

To monitor the behavior of Tomcat the log is the primary source of information. The log can be viewed with the following command:

```
tail -f tomcat-dhis/logs/catalina.out
```

Assuming that the WAR file is called ROOT.war, you can now access your DHIS2 instance at the following URL:

```
http://localhost:8080
```
2.5 Base URL configuration

To set the base URL of the DHIS2 instance, you can specify the following property in the dhis.conf configuration file. This URL should point to the location where end users can reach DHIS2 over the network.

```
server.base.url = https://play.dhis2.org/dev
```

2.6 File store configuration

DHIS2 is capable of capturing and storing files. By default files will be stored on the file system of the server which runs DHIS2 in a files directory under the DHIS2_HOME external directory location.

You can also configure DHIS2 to store files on cloud-based storage providers. Currently, AWS S3 is the only supported provider. To enable cloud-based storage you must define the following additional properties in your dhis.conf file:

```
# File store provider. Currently 'filesystem' and 'aws-s3' are supported.
filestore.provider = filesystem

# Directory in external directory on local file system and bucket on AWS S3.
filestore.container = files

# The following configuration is applicable only on non-filesystem providers (AWS S3)
# Datacenter location. Not required but recommended for performance reasons.
filestore.location = eu-west-1

# Public identity / username
filestore.identity = xxxx

# Secret password (sensitive)
filestore.secret = xxxx
```

This configuration is an example reflecting the defaults and should be changed to fit your needs. In other words, you can omit it entirely if you plan to use the default values. If you want to use an external provider the last block of properties need to be defined, as well as the provider property being set to a supported provider (currently only AWS S3).

**Note**

If you've configured cloud storage in dhis.conf, all files you upload or the files the system generates will use cloud storage.

For a production system the initial setup of the file store should be carefully considered as moving files across storage providers while keeping the integrity of the database references could be complex. Keep in mind that the contents of the file store might contain both sensitive and integral information and protecting access to the folder as well as making sure a backup plan is in place is recommended on a production implementation.

**Note**

AWS S3 is the only supported provider (starting from version 2.27) but more providers are likely to be added, such as Google Cloud Store and
2.7 Google service account configuration

DHIS2 can connect to various Google service APIs. For instance, the DHIS2 GIS component can utilize the Google Earth Engine API to load map layers. In order to provide API access tokens you must set up a Google service account and create a private key:

- Create a Google service account. Please consult the [Google identity platform](https://developers.google.com/identity) documentation.
- Visit the [Google cloud console](https://console.cloud.google.com/) and go to API Manager > Credentials > Create credentials > Service account key. Select your service account and JSON as key type and click Create.
- Rename the JSON key to `dhis-google-auth.json`.

After downloading the key file, put the `dhis-google-auth.json` file in the DHIS2_HOME directory (the same location as the `dhis.conf` file). As an example this location could be:

```
/home/dhis/config/dhis-google-auth.json
```

2.8 LDAP configuration

DHIS2 is capable of using an LDAP server for authentication of users. For LDAP authentication it is required to have a matching user in the DHIS2 database per LDAP entry. The DHIS2 user will be used to represent authorities / user roles.

To set up LDAP authentication you need to configure the LDAP server URL, a manager user and an LDAP search base and search filter. This configuration should be done in the main DHIS 2 configuration file (`dhis.conf`). LDAP users, or entries, are identified by distinguished names (DN from now on). An example configuration looks like this:

```
# LDAP server URL
ldap.url = ldaps://domain.org:636

# LDAP manager entry distinguished name
ldap.manager.dn = cn=johndoe,dc=domain,dc=org

# LDAP manager entry password
ldap.manager.password = xxxx

# LDAP base search
ldap.search.base = dc=domain,dc=org

# LDAP search filter
ldap.search.filter = (cn={0})
```

The LDAP configuration properties are explained below:

- `ldap.url`: The URL of the LDAP server for which to authenticate against. Using SSL/encryption is strongly recommended in order to make authentication secure. As example URL is `ldaps://domain.org:636`, where ldaps refers to the protocol, `domain.org` refers to the domain name or IP address, and 636 refers to the port (636 is default for LDAPS).
**ldap.manager.dn:** An LDAP manager user is required for binding to the LDAP server for the user authentication process. This property refers to the DN of that entry. I.e. this is not the user which will be authenticated when logging into DHIS2, rather the user which binds to the LDAP server in order to do the authentication.

- **ldap.manager.password:** The password for the LDAP manager user.
- **ldap.search.base:** The search base, or the distinguished name of the search base object, which defines the location in the directory from which the LDAP search begins.
- **ldap.search.filter:** The filter for matching DNs of entries in the LDAP directory. The \{0\} variable will be substituted by the DHIS2 username, or alternatively, the LDAP identifier defined for the user with the supplied username.

DHIS2 will use the supplied username / password and try to authenticate against an LDAP server entry, then look up user roles / authorities from a corresponding DHIS2 user. This implies that a user must have a matching entry in the LDAP directory as well as a DHIS2 user in order to log in.

During authentication, DHIS2 will try to bind to the LDAP server using the configured LDAP server URL and the manager DN and password. Once the binding is done, it will search for an entry in the directory using the configured LDAP search base and search filter.

The \{0\} variable in the configured filter will be substituted before applying the filter. By default, it will be substituted by the supplied username. You can also set a custom LDAP identifier on the relevant DHIS2 user account. This can be done through the DHIS2 user module user interface in the add or edit screen by setting the “LDAP identifier” property. When set, the LDAP identifier will be substituted for the \{0\} variable in the filter. This feature is useful when the LDAP common name is not suitable or cannot for some reason be used as a DHIS2 username.

### 2.9 Encryption configuration

DHIS2 allows for encryption of data. This however requires some extra setup.

#### 2.9.1 Java Cryptography Extension

DHIS2 uses an encryption algorithm classified as strong and therefore requires the *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files* to be installed. These files can be installed through these steps:

1. Download the JCE Unlimited Strength Jurisdiction Policy Files for your java version of Java from the Oracle Web site. Scroll down to the "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files" section. It is important that the version of the files match the version of Java on your server.


2. Extract the downloaded ZIP archive. It contains two JAR files: *local_policy.jar* and *US_export_policy.jar*.

3. Locate the JDK directory of your Java installation. From there, navigate into the *jre/security* directory. On Ubuntu it is often found at `/usr/lib/jvm/java-8-oracle/jre/lib/security`.

4. (Optional) Back up your existing *local_policy.jar* and *US_export_policy.jar* in case you want to revert to them later.

5. Copy the *local_policy.jar* and *US_export_policy.jar* files into the security folder. You should now have the following files which completes the installation. Remember to restart your servlet container for it to take effect.
2.9.2 Password configuration

To provide security to the encryption algorithm you will have to set a password in the *dhis.conf* configuration file through the `encryption.password` property:

```
encryption.password = xxxx
```

The `encryption.password` property is the password used when encrypting and decrypting data in the database. Note that the password must not be changed once it has been set and data has been encrypted as the data can then no longer be decrypted.

The password must be at least 24 **characters long**. A mix of numbers and lower- and uppercase letters are recommended. The encryption password must be kept secret.

2.9.3 Considerations for encryption

*A word of caution*: It is not possible to recover encrypted data if the encryption password is lost or changed. Conversely, the encryption provides no security if the password is compromised. Hence, great consideration should be given to storing the password in a safe place.

2.10 Read replica database configuration

DHIS 2 allows for utilizing read only replicas of the master database (the main DHIS 2 database). The purpose of read replicas is to enhance the performance of database read queries and scale out the capacity beyond the constraints of a single database. Read-heavy operations such as analytics and event queries will benefit from this.

The configuration requires that you have created one or more replicated instances of the master DHIS 2 database. PostgreSQL achieves this through a concept referred to as **streaming replication**. Configuring read replicas for PostgreSQL is not covered in this guide.

Read replicas can be defined in the *dhis.conf* configuration file. You can specify up to 5 read replicas per DHIS 2 instance. Each read replica is denoted with a number between 1 and 5. The JDBC connection URL must be defined per replica. The username and password can be specified; if not, the username and password for the master database will be used instead.

The configuration for read replicas in *dhis.conf* looks like the below. Each replica is specified with the configuration key `readN` prefix, where N refers to the replica number.

```
# Read replica 1 configuration
# Database connection URL, username and password
read1.connection.url = jdbc:postgresql://127.0.0.11/dbread1
read1.connection.username = dhis
read1.connection.password = xxxx

# Read replica 2 configuration
# Database connection URL, username and password
read2.connection.url = jdbc:postgresql://127.0.0.12/dbread2
read2.connection.username = dhis
read2.connection.password = xxxx
```
Note that you must restart your servlet container for the changes to take effect. DHIS 2 will automatically distribute the load across the read replicas. The ordering of replicas has no significance.

2.11 Web server cluster configuration

This section describes how to set up the DHIS 2 application to run in a cluster.

2.11.1 Clustering overview

Clustering is a common technique for improving system scalability and availability. Clustering refers to setting up multiple web servers such as Tomcat instances and have them serve a single application. Clustering allows for *scaling out* an application in the sense that new servers can be added to improve performance. It also allows for *high availability* as the system can tolerate instances going down without making the system inaccessible to users.

When setting up multiple Tomcat instances there is a need for making the instances aware of each other. This awareness will enable DHIS 2 to keep the local data (Hibernate) caches in sync and in a consistent state. When an update is done on one instance, the caches on the other instances must be notified so that they can be invalidated and avoid becoming stale.

There are two aspects to configure in *dhis.conf* in order to run DHIS 2 in a cluster.

- Each instance must specify the other DHIS 2 application members of the cluster.
- An instance of the Redis data store must be installed and connection information must be configured for each DHIS 2 application instance.

2.11.2 Cluster instance configuration

A DHIS 2 cluster setup is based on manual configuration of each instance. For each DHIS 2 instance one must specify the public *hostname* as well as the hostnames of the other instances participating in the cluster.

The hostname of the server is specified using the *cluster.hostname* configuration property. Additional servers which participate in the cluster are specified using the *cluster.members* configuration property. The property expects a list of comma separated values where each value is of the format *host:port*.

The hostname must be visible to the participating servers on the network for the clustering to work. You might have to allow incoming and outgoing connections on the configured port numbers in the firewall.

The port number of the server is specified using the *cluster.cache.port* configuration property. The remote object port used for registry receive calls is specified using *cluster.cache.remote.object.port*. Specifying the port numbers is typically only useful when you have multiple cluster instances on the same server / virtual machine or if you need to explicitly specify the ports to be used so as to have them configured in firewall. When running cluster instances on separate servers / virtual machines it is often appropriate to use the default port number and omit the ports configuration properties. If omitted, 4001 will be assigned as the listener port and a random free port will be assigned as the remote object port.
An example setup for a cluster of two web servers is described below. For server A available at hostname 193.157.199.131 the following can be specified in `dhis.conf`:

```properties
# Cluster configuration for server A
# Hostname for this web server
cluster.hostname = 193.157.199.131

# Ports for cache listener, can be omitted
cluster.cache.port = 4001
cluster.cache.remote.object.port = 5001

# List of Host:port participating in the cluster
cluster.members = 193.157.199.132:4001
```

For server B available at hostname 193.157.199.132 the following can be specified in `dhis.conf` (notice how port configuration is omitted):

```properties
# Cluster configuration for server B
# Hostname for this web server
cluster.hostname = 193.157.199.132

# List of servers participating in cluster
cluster.members = 193.157.199.131:4001
```

You must restart each Tomcat instance to make the changes take effect. The two instances have now been made aware of each other and DHIS 2 will ensure that their caches are kept in sync.

### 2.11.3 Cluster shared data store configuration

In a cluster setup, a Redis instance is required and will handle shared user sessions, application cache and cluster node leadership.

For optimum performance, Redis Keyspace events for generic commands and expired events need to be enabled in the Redis Server. If you are using a cloud platform-managed Redis server (like AWS ElastiCache for Redis or Azure Cache for Redis) you will have to enable keyspace event notifications using the respective cloud interfaces. If you are setting up a standalone Redis server, enabling keyspace event notifications can be done in the `redis.conf` file by adding or uncommenting the following line:

```properties
notify-keyspace-events Egx
```

DHIS2 will connect to Redis if the `redis.enabled` configuration property in `dhis.conf` is set to `true` along with the following properties:

- **`redis.host`**: Specifies where the redis server is running. Defaults to `localhost`. Mandatory.
- **`redis.port`**: Specifies the port in which the redis server is listening. Defaults to 6379. Optional.
- **`redis.password`**: Specifies the authentication password. If a password is not required it can be left blank.
redis.use.ssl: Specifies whether the Redis server has SSL enabled. Defaults to false.
* Optional. Defaults to false.

When Redis is enabled, DHIS2 will automatically assign one of the running instances as the leader of the cluster. The leader instance will be used to execute jobs or scheduled tasks that should be run exclusively by one instance. Optionally you can configure the `leader.time.to.live.minutes` property in `dhis.conf` to set up how frequently the leader election needs to occur. It also gives an indication of how long it would take for another instance to take over as the leader after the previous leader has shutdown/crashed. The default value is 2 minutes. Note that assigning a leader in the cluster is only done if Redis is enabled. An example snippet of the `dhis.conf` configuration file with Redis enabled and leader election time configured is shown below.

```
# Redis Configuration
redis.enabled = true
redis.host = 193.158.100.111
redis.port = 6379
redis.password = <your password>
redis.use.ssl = false
# Optional, defaults to 2 minutes
leader.time.to.live.minutes=4
```

2.11.4 Load balancing

With a cluster of Tomcat instances set up, a common approach for routing incoming web requests to the backend instances participating in the cluster is using a load balancer. A load balancer will make sure that load is distributed evenly across the cluster instances. It will also detect whether an instance becomes unavailable, and if so, stop routine requests to that instance and instead use other available instances.

Load balancing can be achieved in multiple ways. A simple approach is using `nginx`, in which case you will define an `upstream` element which enumerates the location of the backend instances and later use that element in the `proxy` location block.

```
http {
  # Upstream element with sticky sessions
  upstream dhis_cluster {
    ip_hash;
    server 193.157.199.131:8080;
    server 193.157.199.132:8080;
  }

  # Proxy pass to backend servers in cluster
  server {
    listen 80;

    location / {
      proxy_pass http://dhis_cluster/;
    }
  }
```
DHIS 2 keeps server-side state for user sessions to a limited degree. Using “sticky sessions” is a simple approach to avoid replicating the server session state by routing requests from the same client to the same server. The `ip_hash` directive in the upstream element ensures this.

Note that several instructions have been omitted for brevity in the above example. Consult the reverse proxy section for a detailed configuration guide.

### 2.12 Analytics cache configuration

DHIS 2 supports a server-side cache for analytics API responses, used by all of the analytics web apps. This cache sits within the DHIS 2 application and hence is protected by the DHIS 2 authentication and security layer. You can configure the expiration of cached entries in seconds. To enable the cache you can define the `analytics.cache.expiration` property in `dhis.conf`. The example below enabled the cache and sets expiration to one hour.

```bash
analytics.cache.expiration = 3600
```

### 2.13 Starting Tomcat at boot time

In certain situations a server might reboot unexpectedly. It is hence preferable to have Tomcat start automatically when the server starts. To achieve that the first step is to create init scripts. Create a new file called `tomcat` and paste the below content into it (adjust the `HOME` variable to your environment):

```bash
#!/bin/
sh
#Tomcat init script
HOME=/home/dhis/tomcat/bin

case $1 in
  start) sh ${HOME}/startup.sh ;;
  stop) sh ${HOME}/shutdown.sh ;;
  restart
    sh ${HOME}/shutdown.sh
    sleep 5
    sh ${HOME}/startup.sh ;;
  esac
exit 0
```

Move the script to the init script directory and make them executable by invoking:
Next make sure the tomcat init script will be invoked during system startup and shutdown:

```
sudo /usr/sbin/update-rc.d -f tomcat defaults 81
```

Tomcat will now be started at system startup and stopped at system shutdown. If you later need to revert this you can replace `defaults` with `remove` and invoke the above commands again.

### 2.14 Reverse proxy configuration

A reverse proxy is a proxy server that acts on behalf of a server. Using a reverse proxy in combination with a servlet container is optional but has many advantages:

- Requests can be mapped and passed on to multiple servlet containers. This improves flexibility and makes it easier to run multiple instances of DHIS2 on the same server. It also makes it possible to change the internal server setup without affecting clients.
- The DHIS2 application can be run as a non-root user on a port different than 80 which reduces the consequences of session hijacking.
- The reverse proxy can act as a single SSL server and be configured to inspect requests for malicious content, log requests and responses and provide non-sensitive error messages which will improve security.

#### 2.14.1 Basic nginx setup

We recommend using `nginx` as reverse proxy due to its low memory footprint and ease of use. To install invoke the following:

```
sudo apt-get install nginx
```

NGINX can now be started, reloaded and stopped with the following commands:

```
sudo /etc/init.d/nginx start
sudo /etc/init.d/nginx reload
sudo /etc/init.d/nginx stop
```

Now that we have installed nginx we will now continue to configure regular proxying of requests to our Tomcat instance, which we assume runs at `http://localhost:8080`. To configure nginx you can open the configuration file by invoking:

```
sudo nano /etc/nginx/nginx.conf
```

NGINX configuration is built around a hierarchy of blocks representing http, server and location, where each block inherit settings from parent blocks. The following snippet will configure nginx to proxy pass (redirect) requests from port 80 (which is the port nginx will listen on by default) to our Tomcat instance. Include the following configuration in nginx.conf:
http {
  gzip on; # Enables compression, incl Web API content-types
  gzip_types
    "application/json;charset=utf-8" application/json
    "application/javascript;charset=utf-8" application/javascript text/javascript
    "application/xml;charset=utf-8" application/xml text/xml
    "text/css;charset=utf-8" text/css
    "text/plain;charset=utf-8" text/plain;

  server {
    listen 80;
    root /home/dhis/tomcat/webapps/ROOT; # Update path!
    client_max_body_size 10M;

    # Serve static files
    location ~ (\.js|\.css|\.gif|\.woff|\.ttf|\.eot|\.ico|/dhis-web-commons/|/images/|icons/).*\.png$ {
      add_header  Cache-Control public;
      expires 14d;
    }

    # Proxy pass to servlet container
    location / {
      proxy_pass http://localhost:8080/;
      proxy_redirect off;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto http;
      proxy_buffer_size 128k;
      proxy_buffers 8 128k;
      proxy_busy_buffers_size 256k;
      proxy_cookie_path ~*^/(.*) "$1; SameSite=Lax";
    }
  }
}

You can now access your DHIS2 instance at http://localhost. Since the reverse proxy has been set up we can improve security by making Tomcat only listen for local connections. In /conf/server.xml you can add an address attribute with the value localhost to the Connector element for HTTP 1.1 like this:

```
<Connector
  address="localhost"
  protocol="HTTP/1.1" />
```

### 2.14.2 Enabling SSL with nginx

In order to improve security it is recommended to configure the server running DHIS2 to communicate with clients over an encrypted connection and to identify itself to clients using a trusted certificate. This can be achieved through SSL which is an cryptographic communication protocol running on top of TCP/IP. First, install the required openssl library:

```
sudo apt-get install openssl
```
To configure nginx to use SSL you will need a proper SSL certificate from an SSL provider. The cost of a certificate varies a lot depending on encryption strength. An affordable certificate from Rapid SSL Online should serve most purposes. To generate the CSR (certificate signing request) you can invoke the command below. When you are prompted for the Common Name, enter the fully qualified domain name for the site you are securing.

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

When you have received your certificate files (.pem or .crt) you will need to place it together with the generated server.key file in a location which is reachable by nginx. A good location for this can be the same directory as where your nginx.conf file is located.

Below is an nginx server block where the certificate files are named server.crt and server.key. Since SSL connections usually occur on port 443 (HTTPS) we pass requests on that port (443) on to the DHIS2 instance running on http://localhost:8080 The first server block will rewrite all requests connecting to port 80 and force the use of HTTPS/SSL. This is also necessary because DHIS2 is using a lot of redirects internally which must be passed on to use HTTPS. Remember to replace <server-ip> with the IP of your server. These blocks should replace the one from the previous section.

```
http {
  gzip on; # Enables compression, incl Web API content-types
  gzip_types
    "application/json;charset=utf-8" application/json
    "application/javascript;charset=utf-8" application/javascript
    "application/xml;charset=utf-8" application/xml text/xml
    "text/css;charset=utf-8" text/css
    "text/plain;charset=utf-8" text/plain;

  # HTTP server - rewrite to force use of SSL
  server {
    listen     80;
    rewrite    ^ https://<server-url>$request_uri? permanent;
  }

  # HTTPS server
  server {
    listen               443 ssl;
    root  /home/dhis/tomcat/webapps/ROOT; # Update path!
    client_max_body_size 10M;

    ssl                  on;
    ssl_certificate      server.crt;
    ssl_certificate_key  server.key;

    ssl_session_cache    shared:SSL:20m;
    ssl_session_timeout  10m;
    ssl_protocols              TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers                RC4:H:MD5:TLSv1.2a:TLSv1.2:
    ssl_prefer_server_ciphers on;

    # Serve static files
    location ~ (/.js|\.css|\.woff|\.ttf|\.eot|\.ico|(/dhis-web-commons/|/images/|icons/).*)\.(png)$ { 
```

Note the last “https” header value which is required to inform the servlet container that the request is coming over HTTPS. In order for tomcat to properly produce Location URLs using https you also need to add two other parameters to the Connector in tomcat's server.xml file:

```xml
<Connector
    scheme="https"
    proxyPort="443" />
```

### 2.14.3 Enabling caching and SSL with nginx

Requests for reports, charts, maps and other analysis-related resources will often take some time to respond and might utilize a lot of server resources. In order to improve response times, reduce the load on the server and hide potential server downtime we can introduce a cache proxy in our server setup. The cached content will be stored in directory `/var/cache/nginx`, and up to 250 MB of storage will be allocated. Nginx will create this directory automatically.

```conf
http {
    root              /home/dhis/tomcat/webapps/ROOT; # Update path!
    proxy_cache_path  /var/cache/nginx  levels=1:2  keys_zone=dhis:250m  inactive=1d;
    gzip on; # Enables compression, incl Web API content-types
    gzip_types
        "application/json;charset=utf-8" application/json
        "application/javascript;charset=utf-8" application/javascript
        "application/xml;charset=utf-8" application/xml
text/css_CHARSET=utf-8" text/css
        "text/plain;charset=utf-8" text/plain;

    # HTTP server - rewrite to force use of HTTPS
    server {
        listen 80;
        rewrite ^ https://www.domain.com/$request_uri? permanent;
    }

    # HTTPS server
    server {
```
2 Installation

2.14.4 Rate limiting with nginx

Certain web API calls in DHIS 2, like the analytics APIs, are compute intensive. As a result it is favorable to rate limit these APIs in order to allow all users of the system to utilize a fair share of the server resources. Rate limiting can be achieved with nginx. There are numerous approaches to achieving rate limiting and this is intended to document the nginx-based approach.

The below nginx configuration will rate limit the analytics web API, and has the following elements at the http and location block level (the configuration is shortened for brevity):

```
listen 443 ssl;
client_max_body_size 10M;
ssl on;
ssl_certificate server.crt;
ssl_certificate_key server.key;
ssl_session_timeout 30m;
ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers HIGH:aNULL:!MD5;
ssl_prefer_server_ciphers on;
# Serve static files
location ~ (\.js|\.css|\.gif|\.woff|\.ttf|\.eot|\.ico|/dhis-web-commons|/images|/icons)/.*\.png$ {
  add_header Cache-Control public;
  expires 14d;
}
# Proxy pass to servlet container and potentially cache response
location / {
  proxy_pass http://localhost:8080/;
  proxy_redirect off;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto https;
  proxy_buffer_size 128k;
  proxy_buffers 8 128k;
  proxy_busy_buffers_size 256k;
  proxy_cookie_path ~*^/(.*) "/$1; SameSite=Lax";
  proxy_cache dhis;
}
```
The various elements of the configuration can be described as:

- **limit_req_zone $binary_remote_addr zone=limit_analytics:10m rate=5r/s;** Rate limiting is done per request IP.
- **zone=limit_analytics:20m;** A rate limit zone for the analytics API which can hold up to 10 MB of request IP addresses.
- **rate=20r/s;** Each IP is granted 5 requests per second.
- **location ~ ^/api/(/)?analytics(.*)$ {** Requests for the analytics API endpoint are rate limited.
- **burst=20:** Bursts of up to 20 requests will be queued and serviced at a later point; additional requests will lead to a 503.

For a full explanation please consult the [nginx documentation](#).

### 2.14.5 Additional resources on SSL

The configuration demonstrated above should be regarded as the absolute minimum in order to establish a secure server. However, encryption methods are constantly being updated, so implementers who are administering their own server, should ensure that the server is regularly updated with recent security patches (particularly the HTTP server and SSL libraries).

There are numerous additional tutorials and information available on the web, including a helpful [step-by-step guide](#) for using the free [Let's Encrypt SSL certificate system](#). It may also be useful to regularly test your SSL security with [this website](#).

### 2.14.6 Making resources available with nginx

In some scenarios it is desirable to make certain resources publicly available on the Web without requiring authentication. One example is when you want to make data analysis related resources in the Web API available in a Web portal. The following example will allow access to charts, maps, reports, report table and document resources through basic authentication by injecting an Authorization HTTP header into the request. It will remove the Cookie header from the request and the Set-Cookie header from the response in order to avoid changing the currently logged in user. It is recommended to create a user for this purpose given only the minimum authorities required. The Authorization value can be constructed by Base64-encoding the username appended with a colon and the password and prefix it “Basic”, more precisely “Basic base64_encode(username:password)”. It will check the HTTP method used for requests and return 405 Method Not Allowed if anything but GET is detected.

It can be favorable to set up a separate domain for such public users when using this approach. This is because we don't want to change the credentials for already logged in users when they access the public resources. For instance, when your server is deployed at somedomain.com, you can set a dedicated subdomain at api.somedomain.com, and point URLs from your portal to this subdomain.
server {
  listen 80;
  server_name api.somedomain.com;

  location ~ ^/(api/(charts|chartValues|reports|reportTables|documents|maps|organisationUnits)|
  dhis-web-commons/javascripts|images|dhis-web-commons-ajax-json|dhis-web-mapping|dhis-web-
  visualizer) {
    if ($request_method != GET) {
      return 405;
    }

    proxy_pass http://localhost:8080;
    proxy_redirect off;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto http;
    proxy_set_header Authorization "Basic YWRtaW46ZGlzdHJpY3Q=";
    proxy_set_header Cookie "";
    proxy_hide_header Set-Cookie;
  }
}

2.14.7 Basic reverse proxy setup with Apache

The Apache HTTP server is a popular HTTP server. Depending on your exact nature of
deployment, you may need to use Apache as a reverse proxy for your DHIS2 server. In this
section, we will describe how to implement a simple reverse proxy setup with Apache.

Important

Using nginx is the preferred option as reverse proxy with DHIS2 and
you should not attempt to install both nginx and Apache on the same
server. If you have installed nginx please ignore this section.

First we need to install a few necessary programs modules for Apache and enable the modules.

```
sudo apt-get install apache2 libapache2-mod-proxy-html libapache2-mod-jk
a2enmod proxy proxy_ajp proxy_connect
```

Let’s define an AJP connector which Apache HTTP server will use to connect to Tomcat with. The
Tomcat server.xml file should be located in the /conf/ director of your Tomcat installation. Be
sure this line is uncommented. You can set the port to anything you like which is unused.

```
<Connector
  port="8009"
  protocol="AJP/1.3"
  redirectPort="8443" />
```

Now, we need to make the adjustments to the Apache HTTP server which will answer requests on
port 80 and pass them to the Tomcat server through an AJP connector. Edit the file /etc/
apache2/mods-enabled/proxy.conf so that it looks like the example below. Be sure that the
port defined in the configuration file matches the one from Tomcat.
You now can restart Tomcat and the Apache HTTPD server and your DHIS2 instance should be available on http://myserver/dhis where myserver is the hostname of your server.

2.14.8 SSL encryption with Apache

Using Apache and the reverse proxy setup described in the previous section, we can easily implement encrypted transfer of data between clients and the server over HTTPS. This section will describe how to use self-signed certificates, although the same procedure could be used if you have fully-signed certificates as well.

First (as root), generate the necessary private key files and CSR (Certificate Signing Request):

```
mkdir /etc/apache2/ssl
cd /etc/apache2/ssl
openssl genrsa -des3 -out server.key 1024
openssl req -new -key server.key -out server.csr
```

We need to remove the password from the key, otherwise Apache will not be able to use it:

```
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
```

Next, generate a self-signed certificate which will be valid for one year.

```
openssl x509 -req -days 365 -in server.csr -signkey \\server.key -out server.crt
```

Now, lets configure Apache by enabling the SSL modules and creating a default site.

```
a2enmod ssl
a2ensite default-ssl
```

Now, we need to edit the default-ssl (located at /etc/apache2/sites-enabled/default-ssl) file in order to enable the SSL transfer functionality of Apache.

```
<VirtualHost *:443>
  ServerAdmin wemaster@mydomain.org
  SSLEngine On
  SSLCertificateFile /etc/apache2/ssl/server.crt
</VirtualHost>
```
Be sure that the *:80 section of this file is changed to port *:443, which is the default SSL port. Also, be sure to change the ServerAdmin to the webmaster’s email. Lastly, we need to be sure that the hostname is setup properly in /etc/hosts. Just under the “localhost” line, be sure to add the server’s IP address and domain name.

```
127.0.0.1 localhost
XXX.XX.XXX.XXX foo.mydomain.org
```

Now, just restart Apache and you should be able to view https://foo.mydomain.org/dhis.

```
/etc/init.d/apache2 restart
```

### 2.15 DHIS2 configuration reference

The following describes the full set of configuration options for the `dhis.conf` configuration file. The configuration file should be placed in a directory which is pointed to by a `DHIS2_HOME` environment variable.

Note: You should not attempt to use this configuration file directly, rather use it as a reference for the available configuration options. Many of the properties are optional.

```
# Database connection for PostgreSQL [Mandatory]
# -----------------------------------------------
# Hibernate SQL dialect
connection.dialect = org.hibernate.dialect.PostgreSQLDialect

# JDBC driver class
connection.driver_class = org.postgresql.Driver

# Database connection URL
connection.url = jdbc:postgresql:dhis2

# Database username
connection.username = dhis

# Database password (sensitive)
connection.password = xxxx

# Database schema behavior, can be 'validate', 'update', 'create', 'create-drop'
connection.schema = update

# Max size of connection pool (default: 40)
# connection.pool.max_size = 40
```

# Server [Mandatory]
# -----------------------------------------------
# Base URL to the DHIS 2 instance
server.base.url = https://play.dhis2.org/dev

# Enable secure settings if system is deployed on HTTPS, can be 'off', 'on'
server.https = off

# System [Optional]
# -----------------------------------------------------------------------------

# System mode for database read operations only, can be 'off', 'on'
system.read_only_mode = off

# Session timeout in seconds, default is 3600
system.session.timeout = 3600

# SQL view protected tables, can be 'on', 'off'
system.sql_view_table_protection = on

# Encryption [Optional]
# -----------------------------------------------------------------------------

# Encryption password (sensitive)
encryption.password = xxxx

# File store [Optional]
# -----------------------------------------------------------------------------

# File store provider, currently 'filesystem' and 'aws-s3' are supported
filestore.provider = filesystem

# Directory / bucket name, folder below DHIS2_HOME on file system, 'bucket' on AWS S3
filestore.container = files

# Datacenter location (not required)
filestore.location = eu-west-1

# Public identity / username
filestore.identity = dhis2-id

# Secret key / password (sensitive)
filestore.secret = xxxx

# LDAP [Optional]
# -----------------------------------------------------------------------------

# LDAP server URL
ldap.url = ldaps://300.20.300.20:636

# LDAP manager user distinguished name
ldap.manager.dn = cn=JohnDoe,ou=Country,ou=Admin,dc=hisp,dc=org

# LDAP manager user password (sensitive)
ldap.manager.password = xxxx

# LDAP entry distinguished name search base
ldap.search.base = dc=hisp,dc=org

# LDAP entry distinguished name filter
2.16 Application logging

This section covers application logging in DHIS 2.

2.16.1 Log files

The DHIS2 application log output is directed to multiple files and locations. First, log output is sent to standard output. The Tomcat servlet container usually outputs standard output to a file under “logs”:

<tomcat-dir>/logs/catalina.out

Second, log output is written to a “logs” directory under the DHIS2 home directory as defined by the the DHIS2_HOME environment variables. There is a main log file for all output, and separate log files for various background processes. The main file includes the background process logs as well. The log files are capped at 50 Mb and log content is continuously appended.

<DHIS2_HOME>/logs/dhis.log
<DHIS2_HOME>/logs/dhis-analytics-table.log
<DHIS2_HOME>/logs/dhis-data-exchange.log
<DHIS2_HOME>/logs/dhis-data-sync.log

2.16.2 Log configuration

In order to override the default log configuration you can specify a Java system property with the name log4j.configuration and a value pointing to the Log4j configuration file on the classpath. If you want to point to a file on the file system (i.e. outside Tomcat) you can use the file prefix e.g. like this:
Java system properties can be set e.g. through the `JAVA_OPTS` environment variable or in the Tomcat startup script.

A second approach to overriding the log configuration is to specify logging properties in the `dhis.conf` configuration file. The supported properties are:

```ini
# Max size for log files, default is '100MB'
logging.file.max_size = 250MB

# Max number of rolling log archive files, default is 0
logging.file.max_archives = 2
```

DHIS2 will eventually phase out logging to standard out / catalina.out and as a result it is recommended to rely on the logs under DHIS2_HOME.

### 2.17 Working with the PostgreSQL database

Common operations when managing a DHIS2 instance are dumping and restoring databases. To make a dump (copy) of your database, assuming the setup from the installation section, you can invoke the following:

```
pg_dump dhis2 -U dhis -f dhis2.sql
```

The first argument (dhis2) refers to the name of the database. The second argument (dhis) refers to the database user. The last argument (dhis2.sql) is the file name of the copy. If you want to compress the file copy immediately you can do:

```
pg_dump dhis2 -U dhis | gzip > dhis2.sql.gz
```

To restore this copy on another system, you first need to create an empty database as described in the installation section. You also need to gunzip the copy if you created a compressed version. You can the invoke:

```
psql -d dhis2 -U dhis -f dhis2.sql
```

### 2.18 DHIS2 Live setup

The DHIS2 Live package is extremely convenient to install and run. It is intended for demonstrations, for users who want to explore the system and for small, offline installations typically at districts or facilities. It only requires a Java Runtime Environment and runs on all browsers except Internet Explorer 7 and lower.

To install start by downloading DHIS2 Live from http://dhis2.org and extract the archive to any location. On Windows click the executable archive. On Linux invoke the startup.sh script. After the startup process is done your default web browser will automatically be pointed to http://localhost:8082 where the application is accessible. A system tray menu is accessible on most operating systems where you can start and stop the server and start new browser sessions. Please note that if you have the server running there is no need to start it again, simply open the application from the tray menu.
DHIS2 Live is running on an embedded Jetty servlet container and an embedded H2 database. However it can easily be configured to run on other database systems such as PostgreSQL. Please read the section above about server installations for an explanation of the database configuration. The *dhis.conf* configuration file is located in the *conf* folder. Remember to restart the Live package for your changes to take effect. The server port is 8082 by default. This can be changed by modifying the value in the *jetty.port* configuration file located in the *conf* directory.