Copyright © 2006-2020 DHIS 2 Documentation team

April 2020

<table>
<thead>
<tr>
<th>Revision History</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.34@6d48396</td>
<td>2020-04-30 10:28:59 +0200</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](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 File store configuration
   2.6 Google service account configuration
   2.7 LDAP configuration
   2.8 Encryption configuration
   2.9 Read replica database configuration
   2.10 Web server cluster configuration
      2.10.1 Clustering overview
      2.10.2 DHIS 2 instance cluster configuration
      2.10.3 Redis shared data store cluster configuration
      2.10.4 Files folder configuration
      2.10.5 Load balancer configuration
   2.11 Analytics cache configuration
   2.12 Monitoring
   2.13 Reverse proxy configuration
      2.13.1 Basic nginx setup
      2.13.2 Enabling SSL with nginx
      2.13.3 Enabling caching with nginx
      2.13.4 Rate limiting with nginx
      2.13.5 Making resources available with nginx
   2.14 DHIS2 configuration reference
   2.15 Application logging
      2.15.1 Log files
      2.15.2 Log configuration
   2.16 Working with the PostgreSQL database
3 Monitoring
   3.1 Introduction
   3.2 Setup
      3.2.1 Installing Prometheus + Grafana on Ubuntu and Debian
      3.2.2 Configuring Prometheus as a service
      3.2.3 Create a Prometheus service
      3.2.4 Set-up Nginx reverse proxy
      3.2.5 Enable reverse proxy authentication
      3.2.6 Installing Grafana on Ubuntu and Debian
      3.2.7 Installing Prometheus + Grafana using Docker
      3.2.8 Configure Prometheus to pull metrics from one or more DHIS2 instances
      3.2.9 Configure the DHIS2 exporter
4 Audit

4.1 Introduction
4.2 Single Audit table
4.3 Audit Scope
4.4 Audit Type
4.5 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
locale-gen
```

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:

\[
\text{max_connections} = 200
\]

Determines maximum number of connections which PostgreSQL will allow.

\[
\text{shared_buffers} = 3200\text{MB}
\]

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.

\[
\text{work_mem} = 20\text{MB}
\]

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.

\[
\text{maintenance_work_mem} = 512\text{MB}
\]

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.

\[
\text{effective_cache_size} = 8000\text{MB}
\]

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.

\[
\text{checkpoint_completion_target} = 0.8
\]

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

\[
\text{synchronous_commit} = \text{off}
\]

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:

```ini
# 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
```

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 this setting is enabled.

The `server.base.url` setting refers to the URL at which the system is accessed by end users over 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:

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

Note that this file contains the password for your DHIS2 database in clear text 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. OpenJDK is licensed under the GPL license and can be run free of charge. You can install it with the following command:

```
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:

```bash
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.

```xml
<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):

```bash
wget https://releases.dhis2.org/2.33/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 your preferred patch, if necessary):

```bash
wget https://releases.dhis2.org/2.33/2.33.1/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"
/home/dhis/tomcat-dhis/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 File store configuration

DHIS2 is capable of capturing and storing files. By default, files will be stored on the local 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. AWS S3 is the only supported provider currently. To enable cloud-based storage you must define the following additional properties in your dhis.conf file:

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

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

# The following configuration is applicable to cloud storage only (AWS S3)
# Datacenter location. Optional but recommended for performance reasons.
filestore.location = eu-west-1

# Username / Access key on AWS S3
filestore.identity = xxxx

# Password / Secret key on AWS S3 (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 needs to be defined, as well as the provider property is 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 but more providers are likely to be added in the future, such as Google Cloud Store and Azure Blob Storage. Let us know if you have a use case for additional providers.
2.6 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 documentation](#).
- Visit the [Google cloud console](#) 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
```

2.7 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.

### 2.8 Encryption configuration

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

**Important**

A word of caution: It is not possible to recover encrypted data if the encryption password is lost or changed. If the password is lost, so is the encrypted data. 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.

### 2.9 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.

```plaintext
# 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

# Read replica 3 configuration
# Database connection URL, fallback to master for username and password
read3.connection.url = jdbc:postgresql://127.0.0.13/dbread3
```

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.

### 2.10 Web server cluster configuration

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

#### 2.10.1 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.

There are a few aspects to configure in order to run DHIS 2 in a cluster.

- Each DHIS 2 instance must specify the other DHIS 2 instance members of the cluster in dhis.conf.

- A Redis data store must be installed and connection information must be provided for each DHIS 2 application instance in dhis.conf.

- DHIS 2 instances and servers must share the same files folder used for apps and file uploads, either through the AWS S3 cloud filestorage option or a shared network drive.
A load balancer such as nginx must be configured to distribute Web requests across the cluster instances.

### 2.10.2 DHIS 2 instance cluster configuration

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.

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 or if you need to explicitly specify the ports to match a firewall configuration. When running cluster instances on separate servers 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`:

```bash
# 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):

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

2.10.3 Redis shared data store cluster 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 console 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:

```plaintext
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 become unavailable. 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.

```plaintext
# Redis Configuration
redis.enabled = true
redis.host = 193.158.100.111
redis.port = 6379
redis.password = <your password>
redis.use.ssl = false
```
2 Installation

2.10.4 Files folder configuration

DHIS 2 will store several types of files outside the application itself, such as apps, files saved in data entry and user avatars. When deployed in a cluster, the location of these files must be shared across all instances. On the local filesystem, the location is:

```
{DHIS2_HOME}/files
```

Here, DHIS2_HOME refers to the location of the DHIS 2 configuration file as specified by the DHIS 2 environment variable, and files is the file folder immediately below.

There are two ways to achieve a shared location:

- Use the AWS S3 cloud filestorage option. Files will be stored in an S3 bucket which is automatically shared by all DHIS 2 instances in the cluster. See the File store configuration section for guidance.
- Set up a shared folder which is shared among all DHIS 2 instances and servers in the cluster. On Linux this can be achieved with NFS (Network File System) which is a distributed file system protocol. Note that only the files subfolder under DHIS2_HOME should be shared, not the parent folder.

2.10.5 Load balancer configuration

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 guide.

### 2.11 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
```

### 2.12 Monitoring

DHIS 2 can export Prometheus compatible metrics for monitoring DHIS2 instances. The DHIS2 monitoring infrastructure is designed to expose metrics related to the application runtime and other application-related information.

Infrastructure related metrics (such as host metrics, Tomcat or Postgres) are not directly exposed by the application monitoring engine and they have to be collected separately. The metrics currently exposed by the application are:

- DHIS 2 API (response time, number of calls, etc.)
- JVM (Heap size, Garbage collection, etc.)
- Hibernate (Queries, cache, etc)
- C3P0 Database pool
- Application uptime
- CPU

Monitoring can be enabled in dhis.conf with the following properties (default is off for all properties):

```bash
monitoring.api.enabled = on
monitoring.jvm.enabled = on
monitoring.dbpool.enabled = on
monitoring.hibernate.enabled = off
monitoring.uptime.enabled = on
monitoring.cpu.enabled = on
```

The recommended approach for collecting and visualizing these metrics is through Prometheus and Grafana. For more information, see the monitoring infrastructure page and the Prometheus and Grafana install chapter.
2.13 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.

2.13.1 Basic nginx setup

We recommend using nginx as a 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 inherits 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;
    client_max_body_size 10M;

    # Proxy pass to servlet container
```
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:

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

### 2.13.2 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 a 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](https://www.rapidssl.com) 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 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;

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

  # HTTPS server
  server {
    listen 443 ssl;
    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:HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    # 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 https;
      proxy_buffer_size 128k;
      proxy_buffers 8 128k;
      proxy_busy_buffers_size 256k;
      proxy_cookie_path ~^*(.*) "$1; SameSite=Lax";
    }
  }
}

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 URL headers using HTTPS you also need to add two other parameters to the Connector in the Tomcat server.xml file:

```xml
<Connector scheme="https" proxyPort="443"/>
```
2.13.3 Enabling caching 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.

```nginx
http {
  
  proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=dhis:250m inactive=1d;

  server {

    # 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;
    }
  }
}
```

**Important**

Be aware that a server side cache shortcuts the DHIS2 security features in the sense that requests which hit the server side cache will be served directly from the cache outside the control of DHIS2 and the servlet container. This implies that request URLs can be guessed and reports retrieved from the cache by unauthorized users. Hence, if you capture sensitive information, setting up a server side cache is not recommended.

2.13.4 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):

```nginx
http {

  limit_req_zone $binary_remote_addr zone=limit_analytics:10m rate=5r/s;
}
```
The various elements of the configuration can be described as:

- **limit_req_zone $binary_remote_addr**: Rate limiting is done per request IP.
- **zone=limit_analytics:20**: 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.

### 2.13.5 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.
2.14 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.

```bash
# 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
ldap.search.filter = (cn=*)

# Node [Optional]

# Node identifier, optional, useful in clusters
2.15 Application logging

This section covers application logging in DHIS 2.

2.15.1 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 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

2.15.2 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:

-Dlog4j.configuration=file:/home/dhis/config/log4j.properties

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:

```plaintext
# 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.

### 2.16 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:

```bash
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:

```bash
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 invoke:

```bash
psql -d dhis2 -U dhis -f dhis2.sql
```
3 Monitoring

3.1 Introduction

DHIS2 can export Prometheus compatible metrics for monitoring DHIS2 nodes.

This section describes the steps required to install Prometheus and Grafana using a standard installation procedure (apt-get) and Docker and configure Grafana to show DHIS2 metrics.

For a list of the metrics exposed by a DHIS2 instance, please refer to the monitoring guide on GitHub.

3.2 Setup

The next sections describe how to set up Prometheus and Grafana and how to set up Prometheus to pull data from one or more DHIS2 instances.

3.2.1 Installing Prometheus + Grafana on Ubuntu and Debian

- Download Prometheus from the official download page.
- Make sure to filter for your operating system and your CPU architecture (Linux and amd64).
- Make sure to select the latest stable version, and not the “rc” one, as it is not considered stable enough for now.
- Download the archive, either by clicking on the link or using wget.

```
wget https://github.com/prometheus/prometheus/releases/download/v2.15.2/prometheus-2.15.2.linux-amd64.tar.gz
```

- Untar the zip

```
tar xvzf prometheus-2.15.2.linux-amd64.tar.gz
```

The archive contains many important files, but here is the main ones you need to know.

- `prometheus.yml`: the configuration file for Prometheus. This is the file that you are going to modify in order to tweak your Prometheus server, for example to change the scraping interval or to configure custom alerts;
- `prometheus`: the binary for your Prometheus server. This is the command that you are going to execute to launch a Prometheus instance on your Linux box;
- `promtool`: this is a command that you can run to verify your Prometheus configuration.

3.2.2 Configuring Prometheus as a service

- Create a Prometheus user with a Prometheus group.

```
useradd -rs /bin/false prometheus
```

- Move the Prometheus binaries to the local bin directory
• Create a folder in the /etc folder for Prometheus and move the console files, console libraries and the Prometheus configuration file to this newly created folder.

```bash
mkdir /etc/prometheus
cp -R consoles/ console_libraries/ prometheus.yml /etc/prometheus
```

Create a data folder at the root directory, with a prometheus folder inside.

```bash
mkdir -p data/prometheus
chown -R prometheus:prometheus /data/prometheus /etc/prometheus/*
```

### 3.2.3 Create a Prometheus service

To create a Prometheus systemd service, head over to the /lib/systemd/system folder and create a new systemd file named prometheus.service.

```bash
cd /lib/systemd/system
touch prometheus.service
```

• Edit the newly created file, and paste the following content inside:

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
User=prometheus
Group=prometheus
ExecStart=/usr/local/bin/prometheus \
    --config.file=/etc/prometheus/prometheus.yml \
    --storage.tsdb.path="/data/prometheus" \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
    --web.listen-address=0.0.0.0:9090 \
    --web.enable-admin-api

Restart=always

[Install]
WantedBy=multi-user.target
```

• Save the file and enable the Prometheus service at startup
3 Monitoring

3.2.4 Set-up Nginx reverse proxy

Prometheus does not natively support authentication or TLS encryption. If Prometheus has to be exposed outside the boundaries of the local network, it is important to enable authentication and TLS encryption. The following steps show how to use Nginx as a reverse proxy.

- Test that the service is running

```bash
systemctl status prometheus
```

...  
Active: active (running)

- It should be now possible to access the Prometheus UI by accessing http://localhost:9090.

### 3.2.4 Set-up Nginx reverse proxy

Prometheus does not natively support authentication or TLS encryption. If Prometheus has to be exposed outside the boundaries of the local network, it is important to enable authentication and TLS encryption. The following steps show how to use Nginx as a reverse proxy.

- Install Nginx, if not already installed

```bash
apt update
apt-get install nginx
```

By default, Nginx will start listening for HTTP requests in the default `http` port, which is 80.

If there is already an Nginx instance running on the machine and you are unsure on which port it is listening on, run the following command:

```bash
> lsof | grep LISTEN | grep nginx
```

| nginx | 15792 | root | 8u | IPv4 | 1140223421 | 0t0 | TCP *:http (LISTEN) |

The last column shows the port used by Nginx (`http -> 80`).

By default, Nginx configuration is located in `/etc/nginx/nginx.conf`

Make sure that `nginx.conf` contains the `Virtual Host Config` section

```bash
#
# Virtual Host Configs
#
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*.conf;
```

- Create a new file in `/etc/nginx/conf.d` called `prometheus.conf`

```bash
touch /etc/nginx/conf.d/prometheus.conf
```

- Edit the newly created file, and paste the following content inside:
server {
    listen 1234;

    location / {
        proxy_pass           http://localhost:9090/;
    }
}

• Restart Nginx and browse to http://localhost:1234

systemctl restart nginx

# in case of start-up errors
journalctl -f -u nginx.service

• Configure Prometheus for reverse proxying, by editing /lib/systemd/system/prometheus.service and add the following argument to the list of arguments passed to the Prometheus executable

--web.external-url=https://localhost:1234

• Restart the service

systemctl daemon-reload
systemctl restart prometheus

# in case of errors
journalctl -f -u prometheus.service

3.2.5 Enable reverse proxy authentication

This section shows how to configure basic authentication via the reverse proxy. If you need a different authentication mechanism (SSO, etc.) please check the relevant documentation.

• Make sure that htpasswd is installed on the system

apt-get install apache2-utils

• Create an authentication file

cd /etc/prometheus
htpasswd -c .credentials admin

Choose a strong password, and make sure that the pass file was correctly created.

• Edit the previously created Nginx configuration file (/etc/nginx/conf.d/prometheus.conf), and add the authentication information.
server {
  listen 1234;
  
  location / {
    auth_basic           "Prometheus";
    auth_basic_user_file /etc/prometheus/.credentials;
    proxy_pass           http://localhost:9090/;
  }
}

• Restart Nginx

systemctl restart nginx

# in case of errors
journalctl -f -u nginx.service

• http://localhost:1234 should now prompt for username and password.

3.2.6 Installing Grafana on Ubuntu and Debian

• Add a gpg key and install the OSS Grafana package from APT repo

  apt-get install -y apt-transport-https
  wget -q -O - "https://packages.grafana.com/gpg.key" | sudo apt-key add -
  add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
  apt-get update
  apt-get install grafana

• If the system is using systemd, a new grafana-service is automatically created. Check the systemd file to gain some insight on the Grafana installation

  cat /usr/lib/systemd/system/grafana-server.service

This file is quite important because it offers information about the newly installed Grafana instance.

The file shows:

The **Grafana server binary** is located at `/usr/sbin/grafana-server`. The file that defines all the **environment variables** is located at `/etc/default/grafana-server` The **configuration file** is given via the `CONF_FILE` environment variable. The **PID of the file** is also determined by the **PID_FILE_DIR** environment variable. **Logging, data, plugins** and **provisioning** paths are given by environment variables.

• Start the server
systemctl start grafana-server


The default login for Grafana is admin and the default password is also admin. You will be prompted to change the password on first access.

- Configure Prometheus as a Grafana datasource

Access to the datasources panel by clicking on Configuration > Data sources via the left menu.

Click on Add a datasource

Select a Prometheus data source on the next window.

Configure the datasource according to the Prometheus setup (use authentication, TSL, etc.)

### 3.2.7 Installing Prometheus + Grafana using Docker

This section describes how to start-up a Prometheus stack containing Prometheus and Grafana.

The configuration is based on this project: https://github.com/vegasbrianc/prometheus

- Clone this Github project: https://github.com/vegasbrianc/prometheus

- Start the Prometheus stack using:

  ```
docker stack deploy -c docker-stack.yml prom
  ```

The above command, may result in the following error:

*This node is not a swarm manager. Use “docker swarm init” or “docker swarm join” to connect this node to swarm and try again*

If that happens, you need to start Swarm. You can use the following command line:

```
docker swarm init --advertise-addr <YOUR_IP>
```

Once this command runs successfully, you should be able to run the previous command without problems.

The stack contains also a Node exporter for Docker monitoring. If you are not interested in Docker monitoring, you can comment out the relevant sections in the docker-stack.yml file:

- node-exporter

- cadvisor

- To stop the Prometheus stack:

  ```
docker stack rm prom
  ```
The Prometheus configuration (prometheus.yml) file is located in the prometheus folder.


### 3.2.8 Configure Prometheus to pull metrics from one or more DHIS2 instances

Prior to using Prometheus, it needs basic configuring. Thus, we need to create a configuration file named prometheus.yml

**Note**
The configuration file of Prometheus is written in YAML which strictly forbids to use tabs. If your file is incorrectly formatted, Prometheus will not start. Be careful when you edit it.

Prometheus’ configuration file is divided into three parts: global, rule_files, and scrape_configs.

In the global part we can find the general configuration of Prometheus: scrape_interval defines how often Prometheus scrapes targets, evaluation_interval controls how often the software will evaluate rules. Rules are used to create new time series and for the generation of alerts.

The rule_files block contains information of the location of any rules we want the Prometheus server to load.

The last block of the configuration file is named scrape_configs and contains the information which resources Prometheus monitors.

A simple DHIS2 Prometheus monitoring file looks like this example:

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
- job_name: 'dhis2'
  metrics_path: '/dhis/api/metrics'
  basic_auth:
    username: admin
    password: district

static_configs:
- targets:
  - localhost:80
```

The global scrape_interval is set to 15 seconds which is enough for most use cases.
In the `scrape_configs` part we have defined the DHIS2 exporter. The `basic_auth` blocks contains the credentials required to access the metrics API: consider creating an ad-hoc user only for accessing the metrics endpoint.

Prometheus may or may not run on the same server as DHIS2: in the above configuration, it is assumed that Prometheus monitors only one DHIS2 instance, running on the same server as Prometheus, so we use `localhost`.

**3.2.9 Configure the DHIS2 exporter**

The monitoring subsystem is disabled by default in DHIS2.

Each metrics cluster has to be explicitly enabled in order for the metrics to be exported. To configure DHIS2 to export one or more metrics, check this [document](#).
4.1 Introduction

DHIS2 supports a new audit service which is based on Apache ActiveMQ Artemis. Artemis is used as an asynchronous messaging system by DHIS2.

After an entity is saved to database, an audit message will be sent to the Artemis message consumer service. The message will then be processed in a different thread.

Audit logs can be retrieved from the DHIS2 database. Currently there is no UI or API endpoint available for retrieving audit entries.

4.2 Single Audit Table

All audit entries will be saved into one single table named audit.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>auditid</td>
<td>integer</td>
</tr>
<tr>
<td>audittype</td>
<td>text</td>
</tr>
<tr>
<td>auditscope</td>
<td>text</td>
</tr>
<tr>
<td>klass</td>
<td>text</td>
</tr>
<tr>
<td>attributes</td>
<td>jsonb</td>
</tr>
<tr>
<td>data</td>
<td>bytea</td>
</tr>
<tr>
<td>createdat</td>
<td>timestamp</td>
</tr>
<tr>
<td>createdby</td>
<td>text</td>
</tr>
<tr>
<td>uid</td>
<td>text</td>
</tr>
<tr>
<td>code</td>
<td>text</td>
</tr>
</tbody>
</table>

The new Audit service makes use of two new concepts: Audit Scopes and Audit Type.

4.3 Audit Scope

An Audit Scope is a logical area of the application which can be audited. Currently there are three Audit Scopes:

- Tracker
- Metadata
- Aggregate

- For the Tracker Audit Scope, the audited objects are: Tracked Entity Instance, Tracked Entity Attribute Value, Enrollment, Event
For the Metadata Scope, all “metadata” objects are audited.

- For the Aggregate Scope, the Aggregate Data Value objects are audited.

### 4.4 Audit Type

An Audit Type is an action that triggers an audit operation. Currently we support the following types:

- **READ**
- **CREATE**
- **UPDATE**
- **DELETE**

As an example, when a new Tracked Entity Instance gets created, and if configured like so, the CREATE action is used to insert a new Audit entry in the audit db table.

**Note**: the READ Audit Type will generate a lot of data in database and may have an impact on the performance.

### 4.5 Setup

The audit system is automatically configured to audit for the following scopes and types:

- CREATE, UPDATE, DELETE
- METADATA, TRACKER, AGGREGATE

**No action is required to activate the audit.** The audit can still be configured using the “audit matrix”. The audit matrix is driven by 3 properties in dhis.conf:

- `audit.metadata`
- `audit.tracker`
- `audit.aggregate`

Each property accepts a semicolon delimited list of valid Audit Types:

- **CREATE**
- **UPDATE**
- **DELETE**
- **READ**

For instance, in order to only audit Tracker related object creation and deletion, the following property should be added to dhis.conf:

```ini
audit.tracker = CREATE;DELETE
```
In order to completely disable auditing, this is the configuration to use:

```yaml
audit.metadata =
audit.tracker =
audit.aggregate =
```