System Administration Guide

DHIS core version master

DHIS2 Documentation Team

dhis2

Last update: 2021-12-02

Table of contents

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.

Introduction

DHIS2 runs on all platforms for which there exists a Java JDK, 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 18.04 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.

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 2 GB for a small instance, 12 GB for a medium instance, 64 GB or more for a large instance.
- *CPU cores:* 4 CPU cores for a small instance, 8 CPU cores or more for a medium or large instance.
- *Disk:* SSD is recommeded as storage device. 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 100 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 needed.

Software requirements

Later DHIS2 versions require the following software versions to operate.

- An operating system for which a Java JDK or JRE version 8 or 11 exists. Linux is recommended.
- Java JDK. OpenJDK is recommended.
    - For DHIS 2 version 2.35 version and later, JDK 11 is recommended and JDK 8 or later is required.
    - For DHIS 2 versions older than 2.35, JDK 8 is required.
- PostgreSQL database version 9.6 or later. A later PostgreSQL version such as version 13 is recommended.
- 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.

- Cluster setup only (optional): Redis data store version 4 or later.

Server setup

This section describes how to set up a server instance of DHIS2 on Ubuntu 18.04 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 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.

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.

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.

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
```

PostgreSQL installation

Install PostgreSQL by invoking:

```
sudo apt-get install postgresql-12 postgresql-12-postgis-3
```

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*.

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/12/main/postgresql.conf
```

and set the following properties:

```
max_connections = 200
```

Determines maximum number of connections which PostgreSQL will allow.

```
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.

```
work_mem = 24MB
```

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.

```
maintenance_work_mem = 1024MB
```

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.

```
temp_buffers = 16MB
```

Sets the maximum number of temporary buffers used by each database session. These are session-local buffers used only for access to temporary tables.

```
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.

```
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.

```
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.

```
track_activity_query_size = 8192
```

Specifies the number of bytes reserved to track the currently executing command for each active session. Useful to view the full query string for monitoring of currently running queries.

Restart PostgreSQL by invoking the following command:

```
sudo systemctl restart postgresql
```

Java installation

The recommended Java JDK for DHIS 2 is OpenJDK 11 (for version 2.35 and later). You can install it with the following command:

```
sudo apt-get install openjdk-11-jdk
```

If you prefer OpenJDK 8 (for versions older than 2.35) you can install it with this command:

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

Verify that your installation is correct by invoking:

```
java -version
```

DHIS2 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:

```
/home/dhis/config/dhis.conf
```

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

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

# 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

# ------------------------------------------------------------------------
# Server
# ------------------------------------------------------------------------

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

# Server base URL
# server.base.url = https://server.com
```

It is strongly recommended to enable the `server.https` setting and deploying DHIS 2 with an 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, e.g. like this where `DB\_PASSWD` is the name of the environment variable:

```
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 only the *dhis* user is allowed to read it:

```
chmod 600 dhis.conf
```

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:

```
cd /home/dhis/
sudo tomcat8-instance-create tomcat-dhis
sudo chown -R dhis:dhis tomcat-dhis/
```

This will create an instance in a directory called `tomcat-dhis`. Note that the `tomcat8-user` package allows for creating any number of DHIS2 instances if that is desired.

Next edit the file `tomcat-dhis/bin/setenv.sh` and add the lines below.

- `JAVA_HOME` sets the location of the JDK installation.
- `JAVA_OPTS` passes parameters to the JVM.
  - `-Xms` sets the initial allocation of memory to the Java heap memory space.
  - `-Xmx` sets the maximum allocation of memory to the Java heap memory space. This should reflect how much memory you would like to allocate to the DHIS 2 software application on your server.
- `DHIS2_HOME` sets the location of the `dhis.conf` configuration file for DHIS 2.

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-11-openjdk-amd64`. Note that you should adjust these values to your environment.

```
JAVA_HOME='/usr/lib/jvm/java-11-openjdk-amd64/'
JAVA_OPTS='-Xms4000m -Xmx7000m'
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 DHIS2 WAR files from the following location:

```
https://releases.dhis2.org/
```

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:

```
mv dhis.war tomcat-dhis/webapps/ROOT.war
```

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.

```
#!/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"
```

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
```

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:

```
# 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.

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 identify 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
```

OpenID Connect (OIDC) configuration

DHIS2 supports the OpenID Connect (OIDC) identity layer for single sign-in (SSO). OIDC is a standard authentication protocol that lets users sign in with an identity provider (IdP) such as for example Google. After users have successfully signed in to their IdP, they will be automatically signed in to DHIS2.

This section provides general information about using DHIS2 with an OIDC provider, as well as complete configuration examples.

The DHIS2 OIDC 'authorization code' authentication flow:

1. A user attempts to log in to DHIS2 and clicks the OIDC provider button on the login page.
2. DHIS2 redirects the browser to the IdP's login page.
3. If not already logged in, the user is prompted for credentials. When successfully authenticated, the IdP responds with a redirect back to the DHIS2 server. The redirect includes a unique authorization code generated for the user.
4. The DHIS2 server internally sends the user's authorization code back to the IdP server along with its own client id and client secret credentials.
5. The IdP returns an ID token back to the DHIS2 server. DHIS2 server performs validation of the token.
6. The DHIS2 server looks up the internal DHIS2 user with the mapping claims found in the ID token (defaults to email), authorizes the user and completes the login process.

Requirements for using OIDC with DHIS2:

IdP server account

You must have an admin account on an online identity provider (IdP) or on a standalone server that are supported by DHIS2.

The following IdPs are currently supported and tested:

- Google
- Azure AD
- WSO2
- Okta (See separate tutorial: here)

There is also a **generic provider** config which can support "any" OIDC compatible provider.

DHIS2 user account

You must explicitly create the users in the DHIS2 server before they can log in with the identity provider. Importing them from an external directory such as Active Directory is currently not supported. Provisioning and management of users with an external identity store is not supported by the OIDC standard.

IdP claims and mapping of users

To sign in to DHIS2 with OIDC, a given user must be provisioned in the IdP and then mapped to a pre created user account in DHIS2. OIDC uses a method that relies on claims to share user account attributes with other applications. Claims include user account attributes such as email, phone number, name, etc. DHIS2 relies on a IdP claim to map user accounts from the IdP to those in the DHIS2 server. By default, DHIS2 expects the IdP to pass the *email* claim. Depending on your IdP, you may need to configure DHIS2 to use a different IdP claim.

If you are using Google or Azure AD as an IdP, the default behavior is to use the *email* claim to map IdP identities to DHIS2 user accounts.

> **Note**
>
> In order for a DHIS2 user to be able to log in with an IdP, the user profile checkbox: *External authentication only OpenID or LDAP* must be checked and *OpenID* field must match the claim (mapping claim) returned by the IdP. Email is the default claim used by both Google and Azure AD.

Configure the Identity Provider for OIDC

This topic provides general information about configuring an identity provider (IdP) to use OIDC with DHIS2. This is one step in a multi-step process. Each IdP has slightly different ways to configure it. Check your IdP's own documentation for how to create and configure an OIDC application. Here we refer to the DHIS2 server as the OIDC "application".

Redirect URL

All IdPs will require a redirect URL to your DHIS2 server. You can construct it using the following pattern:

```
(protocol)://(your DHIS2 host)/oauth2/code/PROVIDER_KEY
```

Example when using Google IdP:

```
https://mydhis2-server.org/oauth2/code/google
```

External links to instructions for configuring your IdP:

- Google
- Azure AD tutorial

Example setup for Google

1. Register an account and sign in. For example, for Google, you can go to the Google developer console.
2. In the Google developer dashboard, click 'create a new project'.
3. Follow the instructions for creating an OAuth 2.0 client ID and client secret.
4. Set your "Authorized redirect URL" to: `https://mydhis2-server.org/oauth2/code/google`
5. Copy and keep the "client id" and "client secret" in a secure place.

> **Tip**

When testing on a local DHIS2 instance running for example on your laptop, you can use localhost as the redirect URL, like this: `https://localhost:8080/oauth2/code/google` > *Remember to also add the redirect URL in the Google developer console*

Google dhis.conf example:

```
# Enables OIDC login
oidc.oauth2.login.enabled = on

# Client id, given to you in the Google developer console
oidc.provider.google.client_id = my client id

# Client secret, given to you in the Google developer console
oidc.provider.google.client_secret = my client secret

# [Optional] Authorized redirect URI, the same as set in the Google developer console
# If your public hostname is different from what the server sees internally,
# you need to provide your full public url, like the example below.
oidc.provider.google.redirect_url = https://mydhis2-server.org/oauth2/code/goole

# [Optional] Where to redirect after logging out.
# If your public hostname is different from what the server sees internally,
# you need to provide your full public url, like the example below.
oidc.logout.redirect_url = https://mydhis2-server.org
```

Example setup for Azure AD

Make sure your Azure AD account in the Azure portal is configured with a redirect URL like: `(protocol)://(host)/oauth2/code/PROVIDER_KEY`. To register your DHIS2 server as an "application" in the Azure portal, follow these steps:

Note

PROVIDER_KEY is the "name" part of the configuration key, example: "oidc.provider.PROVIDER_KEY.tenant = My Azure SSO" If you have multiple Azure providers you want to configure, you can use this name form: (azure.0), (azure.1) etc. Redirect URL example: https://mydhis2-server.org/oauth2/code/azure.0

1. Search for and select *App registrations*.
2. Click *New registration*.
3. In the *Name* field, enter a descriptive name for your DHIS2 instance.
4. In the *Redirect URI* field, enter the redirect URL as specified above.
5. Click *Register*.

Azure AD dhis.conf example:

```
# Enables OIDC login
oidc.oauth2.login.enabled = on

# First provider (azure.0):

# Alias, or name that will show on the login button in the DHIS2 login screen.
oidc.provider.azure.0.tenant = organization name
```

```
# Client id, given to you in the Azure portal
oidc.provider.azure.0.client_id = my client id

# Client secret, given to you in the Azure portal
oidc.provider.azure.0.client_secret = my client secret

# [Optional] Authorized redirect URI, the as set in Azure portal
# If your public hostname is different from what the server sees internally,
# you need to provide your full public url, like the example below.
oidc.provider.azure.0.redirect_url = https://mydhis2-server.org/oauth2/code/azure.0

# [Optional] Where to redirect after logging out.
# If your public hostname is different from what the server sees internally,
# you need to provide your full public URL, like the example below.
oidc.logout.redirect_url = https://mydhis2-server.org

# [Optional], defaults to 'email'
oidc.provider.azure.0.mapping_claim = email

# [Optional], defaults to 'on'
oidc.provider.azure.0.support_logout = on


# Second provider (azure.1):

oidc.provider.azure.1.tenant = other organization name
...
```

Generic providers

The generic provider can be used to configure "any" standard OIDC provider which are compatible with "Spring Security".

In the example below we will configure the Norwegian governmental *HelseID* OIDC provider using the provider key `helseid`.

The defined provider will appear as a button on the login page with the provider key as the default name, or the value of the `display_alias` if defined. The provider key is arbitrary and can be any alphanumeric string, except for the reserved names used by the specific providers (`google`, `azure.0`,`azure.1...`, `wso2`).

> **Note** The generic provider uses the following hardcoded configuration defaults: **(These are not possible to change)**
>
> - Client Authentication, `ClientAuthenticationMethod.BASIC`: rfc
> - Authenticated Requests, `AuthenticationMethod.HEADER`: rfc

Generic (helseid) dhis.conf example:

```
# Enables OIDC login
oidc.oauth2.login.enabled = on

# Required variables:
oidc.provider.helseid.client_id = CLIENT_ID
oidc.provider.helseid.client_secret = CLIENT_SECRET
oidc.provider.helseid.mapping_claim = helseid://claims/identity/email
oidc.provider.helseid.authorization_uri = https://helseid.no/connect/authorize
oidc.provider.helseid.token_uri = https://helseid.no/connect/token
```

```
oidc.provider.helseid.user_info_uri = https://helseid.no/connect/userinfo
oidc.provider.helseid.jwk_uri = https://helseid.no/.well-known/openid-configuration/jwks
oidc.provider.helseid.end_session_endpoint = https://helseid.no/connect/endsession
oidc.provider.helseid.scopes = helseid://scopes/identity/email

# [Optional] Authorized redirect URI, the as set in Azure portal
# If your public hostname is different from what the server sees internally,
# you need to provide your full public url, like the example below.
oidc.provider.helseid.redirect_url = https://mydhis2-server.org/oauth2/code/helseid

# [Optional], defaults to 'on'
oidc.provider.helseid.enable_logout = on

# [Optional] Where to redirect after logging out.
# If your public hostname is different from what the server sees internally,
# you need to provide your full public URL, like the example below.
oidc.logout.redirect_url = https://mydhis2-server.org

# [Optional] PKCE support, see: https://oauth.net/2/pkce/), default is 'false'
oidc.provider.helseid.enable_pkce = on

# [Optional] Extra variables appended to the request.
# Must be key/value pairs like: "KEY1 VALUE1,KEY2 VALUE2,..."
oidc.provider.helseid.extra_request_parameters = acr_values lvl4,other_key value2

# [Optional] This is the alias/name displayed on the login button in the DHIS2 login page
oidc.provider.helseid.display_alias = HelseID

# [Optional] Link to an url for a logo. (Can use absolute or relative URLs)
oidc.provider.helseid.logo_image = ../security/btn_helseid.svg
# [Optional] CSS padding for the logo image
oidc.provider.helseid.logo_image_padding = 0px 1px
```

JWT bearer token authentication

Authentication with *JWT bearer tokens* can be enabled for clients which API-based when OIDC is configured. The DHIS2 Android client is such a type of client and have to use JWT authentication if OIDC login is enabled.

> **Note**
>
> DHIS2 currently only supports the OAuth2 authorization code grant flow for authentication with JWT, (also known as "three-legged OAuth") DHIS2 currently only supports using Google as an OIDC provider when using JWT tokens

Requirements

- Configure your Google OIDC provider as described above
- Disable the config parameter `oauth2.authorization.server.enabled` by setting it to 'off'
- Enable the config parameter `oidc.jwt.token.authentication.enabled` by setting it to 'on'
- Generate an Android OAuth2 client_id as described here

JWT authentication example

The following `dhis.conf` section shows an example of how to enable JWT authentication for an API-based client.

```
# Enables OIDC login
oidc.oauth2.login.enabled = on

# Minimum required config variables:
oidc.provider.google.client_id = my_client_id
oidc.provider.google.client_secret = my_client_secret

# Enable JWT support
oauth2.authorization.server.enabled = off
oidc.jwt.token.authentication.enabled = on

# Define client 1 using JWT tokens
oidc.provider.google.ext_client.0.client_id = JWT_CLIENT_ID

# Define client 2 using JWT tokens
oidc.provider.google.ext_client.1.client_id = JWT_CLIENT_ID
```

> **Note**
>
> See link for a separate tutorial for setting up Okta as a generic OIDC provider. link

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.

Encryption configuration

DHIS2 allows for encryption of data. Enabling it 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 (key) 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 is recommended. The encryption password must be kept secret.

> **Important**
>
> 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.

Note that encryption support depends on the *Java Cryptography Extension* (JCE) policy files to be available. These are included in all versions of OpenJDK and Oracle JDK 8 Update 144 or later.

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

# 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.

Web server cluster configuration

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

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 have enabled Hibernate cache invalidation.
- 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.

DHIS 2 instance cache invalidation

DHIS2 can invalidate the application cache by listening for replication events emitted by the PostgreSQL database. This allows for adding new DHIS2 instances without having to configure a unique ID for each "instance" in `dhis.conf`.

Cache invalidation is based on the open-source project Debezium, which works by listening to the replication stream from a PostgreSQL database to detect updates made by other instances.

Prerequisites

- PostgreSQL version 10 or later.
- Logical replication enabled in PostgreSQL.
- PostgreSQL user must have replication authority.

PostgreSQL configuration

The following properties must be specified in the PostgreSQL configuration file:

```
wal_level = logical

# This number has to be the same or bigger as the total number of
# DHIS2 instances you intend to run at the same time.
max_replication_slots = 10
max_wal_senders = 10
```

DHIS2 configuration

The following properties must be specified in the DHIS 2 `dhis.conf` configuration file:

```
debezium.enabled = on
debezium.db.hostname = DHIS2_DATABASE_HOSTNAME
debezium.db.port = DHIS2_DATABASE_PORT_NUMBER
debezium.db.name = DHIS2_DATABASE_NAME
debezium.connection.username = DHIS2_DATABASE_USER
debezium.connection.password = DHIS2_DATABASE_PASSWORD

# [Optional] If you want the server to shutdown if the replication connection is lost (default
is off)
debezium.shutdown_on.connector_stop = on
```

Enable replication access on the database user

Execute the following statement with a PostgreSQL superuser to give replication access to your database user:

```
alter role {db-user} with replication;
```

Troubleshooting

Running out of available replication slots

Every time a DHIS2 instance is started, a new replication slot is created in the database. On every normal shutdown of the instance, the slot is automatically removed. However, if the server has not shut down normally, like for example on a power outage, the replication slot will remain in the

database until it is manually removed. Each replication slot name includes the date it was created and a random string, such that no replication slot will ever have the same name. The number of available replication slots is fixed and is determined by the Postgres config variables: 'max_wal_senders' and 'max_replication_slots'.

To manually remove old stale replication slots, you can use the following SQL statement:

```
select * from pg_replication_slots;
```

To remove a stale replication slot, you can use the following SQL statement (replace the slot ID):

```
select pg_drop_replication_slot('dhis2_1624530654__a890ba555e634f50983d4d6ad0fd63f1');
```

Debezium engine losing connection to the database

If the replication connection fails and can not recover, the node will eventually be out of sync. To prevent this from happening, you can enable the server to shut down if it detects it has lost connection. This feature is disabled by default, since it will shut down the instance without warning. If however you have several instances in a typical load balanced setup and can tolerate that some instances are down, and you have adequate monitoring and alerting set up, you might consider enabling this. It can be enabled by setting this dhis.conf variable:

```
debezium.shutdown_on.connector_stop = on
```

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:

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

DHIS2 will connect to Redis if the *redis.enabled* configuration property in *dhis.conf* is set to *on* 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.

```
# Redis Configuration

redis.enabled = on

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
```

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.

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.

ActiveMQ Artemis configuration

By default DHIS2 will start an embedded instance of ActiveMQ Artemis when booting up. For most use-cases, you do not need to do anything. If you have an existing ActiveMQ Artemis service you want to use instead of the embedded instance you can change the default configuration in your `dhis.conf` file with the configuration properties in the following table.

Property	Value (default first)	Description
amqp.mode	EMBEDDED | NATIVE	The default `EMBEDDED` mode starts up an internal AMQP service when the DHIS2 instance is starting up. If you want to connect to an external AMQP service, set the mode to `NATIVE`.
amqp.host	127.0.0.1	Host to bind to.
amqp.port	15672	If mode is `EMBEDDED`, the embedded server will bind to this port. If mode is `NATIVE`, the client will use this port to connect.
amqp.username	guest	Username to connect to if using `NATIVE` mode.
amqp.password	guest	Password to connect to if using `NATIVE` mode.
amqp.embedded.persistence	off | on	If mode is `EMBEDDED`, this property controls persistence of the internal queue.

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):

```
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.

System configuration

This section covers various system configuration properties.

```
system.read_only_mode = on | off
```

Sets the system in read-only mode. This is useful when you run DHIS 2 on a read-only replica database, to avoid DHIS 2 performing database write operations. Can be `on` or `off`. Default is `off`.

```
system.session.timeout = (seconds)
```

Sets the user session timeout in seconds. Default is 3600 seconds (1 hour).

```
system.sql_view_table_protection = on | off
```

Enables or disables the sensitive database table protection for SQL views. This will prohibit database tables with sensitive data to be queried through SQL views. Disabling is not recommended. Can be `on` or `off`. Default is `on`.

```
system.program_rule.server_execution = on | off
```

Enables or disables execution of server-side program rules. This refers to program rules which have actions for assigning values, sending messages or scheduling messages to be sent. Can be `on` or `off`. Default is `on`.

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.

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

    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" />
```

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 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:

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

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.

```
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.

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):

```
http {
  ..
  limit_req_zone $binary_remote_addr zone=limit_analytics:10m rate=5r/s;
```

```
  server {
    ..

    location ~ ^/api/(\d+/)?analytics(.*)$ {
      limit_req    zone=limit_analytics burst=20;
      proxy_pass   http://localhost:8080/api/$1analytics$2$is_args$args;
      ..
    }
  }
}
```

The various elements of the configuration can be described as:

- *limit_req_zone $binary_remote_addr*: Rate limiting is done per request IP.
- _zone=limit_analytics:20m_: A rate limit zone for the analytics API which can hold up to 10 MB of request IP addresses.
- *rate=20r/s*: Each IP is granted 5 requests per second.
- *location ~ ^/api/(\d+/)?analytics(.*)$*: 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.

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.

```
http {
  ..

  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;
    }
  }
}
```

Block specific Android App versions with nginx

In some scenarios the system administrator might want to block certain Android clients based on its DHIS2 App version. For example, if the users on the field have not updated their Android App version to a specific one and the system administrator wants to block their access to force an update; or completely the opposite scenario when the system administrator wants to block new versions of the App as they have not been yet tested. This can be easily implemented by using specific *User-Agent* rules in the `nginx` configuration file.

```
http {

  server {
    listen       80;
    server_name  api.somedomain.com;

    # Block the latest Android App as it has not been tested
    if ( $http_user_agent ~ 'com\.dhis2/1\.2\.1/2\.2\.1/' ) {
        return 403;
    }

    # Block Android 4.4 (API is 19) as all users should have received new tablets
    if ( $http_user_agent ~ 'com\.dhis2/.*/.*/Android_19' ) {
        return 403;
    }
  }
}
```

> **Note** For the implementation of the method described above note the following:
>
> - Before version 1.1.0 the *User-Agent* string was not being sent.
> - From version 1.1.0 to 1.3.2 the *User-Agent* followed the pattern Dhis2/AppVersion/AppVersion/Android_XX
> - From version 2.0.0 and above the *User-Agent* follows the pattern com.dhis2/SdkVersion/AppVersion/Android_XX
> - Android_XX refers to the Android API level i.e. the Android version as listed here.
> - nginx uses PCRE for Regular Expression matching .

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

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

# --------------------------------------------------------------------
# Database connection for PostgreSQL [Optional]
# --------------------------------------------------------------------

# Minimum number of Connections a pool will maintain at any given time (default: 5).
connection.pool.min_size=5

# Initial size of connection pool (default : 5)
#Number of Connections a pool will try to acquire upon startup. Should be between minPoolSize
and maxPoolSize
connection.pool.initial_size=5

#Determines how many connections at a time will try to acquire when the pool is exhausted.
connection.pool.acquire_incr=5

#Seconds a Connection can remain pooled but unused before being discarded. Zero means idle
connections never expire. (default: 7200)
connection.pool.max_idle_time=7200

#Number of seconds that Connections in excess of minPoolSize should be permitted to remain idle
in the pool before being culled (default: 0)
connection.pool.max_idle_time_excess_con=0

#If this is a number greater than 0, dhis2 will test all idle, pooled but unchecked-out
connections, every this number of seconds. (default: 0)
connection.pool.idle.con.test.period=0

#If on, an operation will be performed at every connection checkout to verify that the
connection is valid. (default: false)
connection.pool.test.on.checkout=false

#If on, an operation will be performed asynchronously at every connection checkin to verify
that the connection is valid. (default: on)
```

```
connection.pool.test.on.checkin=on

#Defines the query that will be executed for all connection tests. Ideally this config is not
needed as postgresql driver already provides an efficient test query. The config is exposed
simply for evaluation, do not use it unless there is a reason to.
connection.pool.preferred.test.query=select 1

#Configure the number of helper threads used by dhis2 for jdbc operations. (default: 3)
connection.pool.num.helper.threads=3

# ----------------------------------------------------------------------
# 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

# Disable server-side program rule execution, can be 'on', 'off'
system.program_rule.server_execution = 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={0})

# --------------------------------------------------------------------
# Node [Optional]
# --------------------------------------------------------------------

# Node identifier, optional, useful in clusters
node.id = 'node-1'

# --------------------------------------------------------------------
# Monitoring [Optional]
# --------------------------------------------------------------------

# DHIS2 API monitoring
monitoring.api.enabled = on

# JVM monitoring
monitoring.jvm.enabled = on

# Database connection pool monitoring
monitoring.dbpool.enabled = on

# Hibernate monitoring, do not use in production
monitoring.hibernate.enabled = off

# Uptime monitoring
monitoring.uptime.enabled = on

# CPU monitoring
monitoring.cpu.enabled = on

# --------------------------------------------------------------------
# Analytics [Optional]
# --------------------------------------------------------------------

# Analytics server-side cache expiration in seconds
analytics.cache.expiration = 3600

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

# System monitoring URL
system.monitoring.url =

# System monitoring username
system.monitoring.username =
```

```
# System monitoring password (sensitive)
system.monitoring.password = xxxx

# ------------------------------------------------------------------------
# App Hub [Optional]
# ------------------------------------------------------------------------

# Base URL to the DHIS2 App Hub service
apphub.base.url = https://apps.dhis2.org"
# Base API URL to the DHIS2 App Hub service, used for app updates
apphub.api.url = https://apps.dhis2.org/api
```

Changelog

DHIS2 writes entries to changelogs when certain entities were changed in the system. The entities fall within two categories: *Aggregate* and *tracker*. The *aggregate* category includes changes to aggregate data values. The *tracker* category includes changes to program instances, program temporary ownership items, tracked entity attribute values and tracked entity data values.

The changelog for both categories are enabled by default. You can control whether to enable or disable the changelog by category through the `dhis.conf` configuration file using the properties described below. Property options are `on` (default) and `off`.

The benefit of the changelog is the ability to see changes which have been performed to the data. The benefits of disabling the changelog is a minor performance improvement by avoiding the cost of writing changelog items to the database, and less database storage used. It is recommended to enable changelog, and great care should be taken if disabling it.

```
# Aggregate changelog, can be 'on', 'off'
changelog.aggregate = on

# Tracker changelog, can be 'on', 'off'
changelog.tracker = on
```

Application logging

This section covers application logging in DHIS 2.

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
```

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:

```
# 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.

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 invoke:

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

Upgrading

Upgrading vs. Updating

When we talk about upgrading DHIS2, we generally simply mean "moving to a newer version". However, there is an important distinction between *upgrading* and *updating*.

Upgrading : Moving to a newer base version of DHIS2 (for example, from 2.34 to 2.36). Upgrading typically requires planning, testing, training (for new features or interfaces), which may take significant time and effort.

Updating : Moving to a newer patch of the current DHIS2 version (for example, from 2.35.1 to 2.35.4). Updating mainly provides bug fixes without changing the functionality of the software. It is lower risk, and we advise everyone to keep their version up to date.

Before you begin

> **Caution**
>
> It is important to note that once you upgrade you will not be able to use the upgraded database with an older version of DHIS2. That is to say **it is not possible to downgrade**.
>
> If you wish to revert to an older version, you must do so with a copy of the database that was created from that older version, or a previous version. Therefore, it is almost always a good idea to make a copy of your database before you uprgrade.

Performing the upgrade

Regardless of whether you are upgrading or updating, the technical process is more-or-less identical. We will just refer to it as upgrading.

1 Safeguard your data

Depending on what sort of DHIS2 instance you have, and what you use it for, the first step is to make sure that you can recover any important data if anything goes wrong with the upgrade.

This means performing standard system admin tasks, such as:

1. Backing up your database
2. Testing in a development environment
3. Scheduling down time (to avoid data being entered during the upgrade)
4. etc.

2 Upgrade the software

From v2.29 or below

If you are starting from v2.29 or below, you must first upgrade to v2.30 version-by-version, manually, following the upgrade notes you find under the specific version numbers on our releases site. When you are at v2.30 you can go to the next section.

From v2.30 or above

If you are starting from at least v2.30:

1. **Read all of the upgrade notes from your current version up to the target version on our releases site.** Make sure your environment meets all of the requirements
2. Make a copy of your database if you didn't do so in step 1
3. Drop any materialized SQL views from your database
4. Stop the server
5. Replace the war file with the target version (There is no need to upgrade to intermediate versions; in fact, it is not recommended)
6. Start the server

You should now be ready to enjoy the new fixes and features.

Monitoring

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.

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.

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.

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

```
cd prometheus-2.15.2.linux-amd64/
cp prometheus promtool /usr/local/bin
chown prometheus:prometheus /usr/local/bin/prometheus
```

- 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.

```
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.

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

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`.

```
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

```
systemctl enable prometheus
systemctl start prometheus
```

- Test that the service is running

```
systemctl status prometheus

...
Active: active (running)
```

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

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

```
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:

```
> 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

```
##
# Virtual Host Configs
##

include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

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

```
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
```

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.

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
```

- Access Grafana web console: http://localhost:3000

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.)

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.

- Access Grafana web console at: http://localhost:3000 with username: `admin` and password: `foobar`

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 `scape_configs` and contains the information which resources Prometheus monitors.

A simple DHIS2 Prometheus monitoring file looks like this example:

```
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`.

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.

Audit

Introduction

DHIS2 supports a new audit service 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 created and 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.

Single Audit table

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

Column	Type	Description	
auditid	integer	Primary key.	
audittype	text	READ, CREATE, UPDATE, DELETE, SEARCH	
auditscope	text	METADATA, AGGREGATE, TRACKER	
klass	text	Audit Entity Java class name.	
attributes	jsonb	A JSON string with attributes of the audited object. Example: `{"valueType":"TEXT", "categoryCombo": "SWQW313FQY", "domainType":"TRACKER"}`.	
data	bytea	Compressed JSON string of the audit entity in byte array format (not humanly readable).	
createdat	timestamp without time zone	Time of creation.	
createdby	text	Username of the user performing the audited operation.	
uid	text	The UID of the audited object.	
code	text	The code of the audited object.	

The audit service makes use of two new concepts: *Audit Scope* and *Audit Type*.

Audit Scope

An audit scope is a logical area of the application which can be audited. Currently there are three audit scopes.

Scope	Key	Audited objects
Tracker	tracker	Tracked Entity Instance, Tracked Entity Attribute Value, Enrollment, Event.
Metadata	metadata	All metadata objects (e.g. Data Element, Organisation Unit).
Aggregate	aggregate	Aggregate Data Value.

Audit Type

An audit type is an action that triggers an audit operation. Currently we support the following four types.

Name	Key	Description
Read	READ	Object was read.
Create	CREATE	Object was created.
Update	UPDATE	Object was updated.
Delete	DELETE	Object was deleted.
Disabled	DISABLED	Disable audit.

> **Caution**
>
> The READ audit type may generate a lot of data in the database and may have an impact on the performance.

Setup

The audit system is enabled by default for the following scopes and types.

Scopes:

- `CREATE`
- `UPDATE`
- `DELETE`

Types:

- `METADATA`
- `TRACKER`
- `AGGREGATE`

This means that **no action is required** to enable the default audit system. The default setting is equivalent to the following `dhis.conf` configuration.

```
audit.metadata = CREATE;UPDATE;DELETE
audit.tracker = CREATE;UPDATE;DELETE
audit.aggregate = CREATE;UPDATE;DELETE
```

The audit can be configured using the *audit matrix*. The audit matrix represents the valid combinations of scopes and types, and is defined with the following properties in the `dhis.conf` configuration file. Each property accepts a semicolon (`;`) delimited list of audit types.

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

Examples

This section demonstrates how to configure the audit system in `dhis.conf`.

To enable audit of create and update of metadata and tracker only:

```
audit.metadata = CREATE;UPDATE
audit.tracker = CREATE;UPDATE
audit.aggregate = DISABLED
```

To only audit tracker related objects create and delete:

```
audit.metadata = DISABLED
audit.tracker = CREATE;DELETE
audit.aggregate = DISABLED
```

To completely disable audit for all scopes:

```
audit.metadata = DISABLED
audit.tracker = DISABLED
audit.aggregate = DISABLED
```

Using Gateways for SMS reporting

DHIS2 supports accepting data via SMS, however, the SMS needs to be composed in a cryptic way to protect the information. The DHIS2 Android App acts as a transparent layer to send the information via SMS where the user does not have to worry about writing the SMS. To send SMSs with the Android App the SMS gateway need to be properly configured. This section explains the different options available and how to achieve that.

Sending SMS

It is important to clarify firstly, that this section mainly concerns the set up of **receiving SMS** (from mobile devices to the DHIS2 server), which is necessary when considering using the App to send (sync) information recorded in the app to the DHIS2 server via SMS. In the App this can be set-up under the *Settings* > *SMS Settings*

Sending SMS, i.e. from the DHIS2 server to mobile devices, is relatively simple to set up. If all that is required is the sending of notifications to users phones from DHIS2 when certain events occur (messaging, thresholds e.t.c.) only sending SMS is required.

This can all be configured in the SMS Service Configuration page within the Mobile Configuration section.

There is out of the box support for common providers such as *Bulk SMS* and *Clickatell*, and both providers support sending of SMS to numbers in most countries.

Note also, it is possible to use a different SMS Gateway for sending and receiving SMS. So even if you set up a solution for receiving SMS below, it is still possible to use one of the aforementioned solutions above for sending SMS.

Using an Android device as SMS Gateway

The simplest solution by far is to use a dedicated Android device as your SMS Gateway. Any phone or tablet running Android OS (4.4, Kitkat or above) should be fine. It will require a constant internet connection, in order to forward messages to your DHIS2 server and it will also need a SIM card to receive the incoming SMS.

You'll need to download and install the DHIS2 Android SMS Gateway app on the mobile device. See a list of releases where you can download the latest APK file to install. There are instructions on the app page itself, but essentially you'll just need to start the app and enter the details of your DHIS2 server (URL, username and password).

Once this is set up and running, you then enter the phone number of this gateway device in the configuration page of any other mobile device using the DHIS2 Capture App. Then, when SMS are sent from these reporting devices, they will be received on the gateway device and automatically forwarded to the DHIS2 server where they will be processed.

Using this gateway device is perfect when testing the SMS functionality. It would be fine when piloting projects that require SMS reporting. As long as the device is plugged into a power supply and has a constant internet connection it works well for small scale projects.

However, when considering moving a project to production it would be necessary to investigate one of the more permanent and reliable solutions for gateways below.

Sending SMS using an Android Device Gateway

This option is currently not supported nor documented.

Dedicated SMS Gateways

This section discusses the use of more permanent and dedicated SMS gateways and the options available. Each of these options below will involve a provider (or yourself) having an SMPP connection to a phone carrier in country and using this connection to receive incoming SMS and forward them on to your DHIS2 server over the internet using HTTP.

These solutions can either use a **long number** or **short code**. A long number is a standard mobile phone number of the type that most private people use, i.e. +61 400123123. A short code is simply a short number, such as 311. Short codes typically cost more to set up and maintain.

Ensuring incoming SMS to DHIS2 server are formatted correctly

When sending incoming SMS to a DHIS2 server via the API you use the following URL: *https:///api/sms/inbound*

In DHIS2 version 2.34 and below, this endpoint requires the format of inbound SMS to be in a very specific format, i.e. the message itself must be a parameter called text, the phone number of the sender must be a parameter called originator.

When using all of the below SMS gateway options, when you configure them to forward incoming SMS on to another web service, they will each have their own format, which will be different to the one expected by the DHIS2 API. For this reason then, it's necessary to reformat them before sending them on to the DHIS2 server.

One option is to run your own very simple web service, which simply receives the incoming SMS from the gateway provider, reformats it to the one required for DHIS2 and forwards it on to your DHIS2 API. Such a service would need to be written by a software developer.

In DHIS2 version 2.35, it is planned to support these cases with a templating system for incoming SMS, so you can specify the format of the messages which will be sent from your provider. That way, you can configure the DHIS2 server to accept incoming SMS from any other SMS gateway provider and they can directly send incoming SMS to the DHIS2 API, without the need for such a formatting web service.

Using RapidPro

RapidPro is a service run by UNICEF in over 50 countries around the world. It is a collection of software which works with in-country phone carriers to enable organisations to design SMS solutions for their projects, such as SMS reporting or awareness campaigns.

The RapidPro service will involve an SMPP connection to one or more phone carriers in-country, usually via a shortcode, potentially dedicated to Health work for NGOs. It's then possible to add a webhook so that incoming SMS are forwarded to another web service, such as the formatting web service described above. If the shortcode is used for other purposes as well, it may be necessary to add the phone numbers of your reporting devices to a separate group, so that only the incoming SMS from those devices is forwarded to the webhook.

RapidPro is currently set up and running in roughly half of the countries which are currently using or piloting DHIS2. Before considering one of the solutions below, which can be costly in terms of both finance and time, it is worth getting in contact with Unicef to see if RapidPro is available and if it can be used for health reporting in your country.

Using commercial SMS gateway providers

Of the commercial SMS gateway providers mentioned in the Sending SMS section above, they will usually have capability to *send* SMS in most countries but can only support *receiving* SMS in a limited amount of countries. The majority of countries they support receiving SMS in are not those using

DHIS2. Of the countries that are using DHIS2, most are already covered by having a RapidPro service running in-country.

However, it is worth researching what commercial options are available for your country. In some countries there will be small national companies that provide SMS services, they'll have existing SMPP connections with the phone providers you can use.

Using phone carriers directly

If none of the above solutions are available it would be necessary to approach the phone carriers in your country directly. The first question to ask them would be whether they are aware of any companies which are operating SMPP connections with them which you may be able to approach.

If not, as a final option, you would need to consider setting up and maintaining your own SMPP connection with the phone provider. However, not all phone providers might offer such a service.

You would need to run your own server running software such as Kannel, which connects (usually via a VPN) to an SMPP service running in the phone providers network. With this in place, any incoming SMS for the configured long number or shortcode are sent from the phone carrier to your Kannel server and you can then forward on these messages as above.

Receiving concatenated or multipart SMS

When syncing data via SMS with the DHIS2 Android App, it uses a compressed format to use as little space (characters of text) as possible. Despite this, it will quite often be the case that a message will extend over the 160 character limit of one standard SMS. On most modern mobile devices these messages will still be sent as one concatenated or multipart SMS, and received as one message. When sending between two mobile devices, when an Android device is used as the gateway, this should be handled without issue.

When selecting an SMS gateway then, it is important to confirm that the phone carrier used supports concatenated SMS. Most of them will support this, but it is important to confirm as the SMS functionality will not work if SMS are split. This relies on something called a UDH (User Data Header). When discussing with providers then, ensure you ask if it is supported.