Chapter 8. Installation

The installation chapter provides information on how to install DHIS 2 in various contexts, including online central server, offline local network, standalone application and self-contained package called DHIS 2 Live.

DHIS 2 runs on all platforms for which there exists a Java Runtime Environment (version 6 or higher, 7 is recommended), which includes most popular operating systems such as Windows, Linux and Mac. DHIS 2 also runs on many relational database systems such as PostgreSQL, MySQL, H2 and Derby. DHIS 2 is packaged as a standard Java Web Archive (WAR-file) and thus runs on any Servlet containers such as Tomcat and Jetty.

The DHIS 2 team recommends Ubuntu 14.04 LTS operating system, PostgreSQL database system and Tomcat Servlet container as the preferred environment for server installations. The mentioned frameworks can be regarded as market leaders within their domain and is heavily field tested over many years.

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.

8.1. Server specifications

DHIS 2 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. DHIS 2 scales linearly on the amount of RAM and number of CPU cores so the more you can afford, the better the application will perform.

- RAM: At least 1 GB memory per 1 million captured data records per month or per 1000 concurrent users. At least 4 GB for a small instance, 12 GB for a medium instance.
- CPU cores: 4 CPU cores for a small instance, 8 CPU cores for a medium or large instance.
- Disk: Ideally use an SSD. Otherwise use a 7200 rpm disk. Minimum read speed is 150 Mb/s, 200 Mb/s is good, 350 Mb/s or better is ideal.

8.2. Server setup

This section describes how to set up a server instance of DHIS 2 on Ubuntu 14.04 64 bit with PostgreSQL as database system and Tomcat as Servlet container. This guide is not meant to be a step-by-step guide per se, but rather to serve as a reference to how DHIS2 can be deployed on a server. There are many possible deployment strategies, which will differ depending on the operating system and database you are using, and other factors. The term *invoke* refers to executing a given command in a terminal.

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

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

8.2.1. Creating a user to run DHIS2

You should create a dedicated user for running DHIS - it is not recommended to run as the root user. Create a new user called dhis by invoking:

```
useradd -d /home/dhis -m dhis -s /bin/bash
```

Then make the user able to perform operations temporarily as root user by invoking:

```
usermod -G sudo dhis
```

Then to set the password for your account invoke:

```
passwd dhis
```

Make sure you set a strong password with at least 15 random characters. You might want to disable remote login for the root account for improved security by invoking:

```
sudo passwd -l root
```

8.2.2. Operating system kernel tuning

These settings are completely optional and does not have to be changed for normal-sized servers. Open the kernel configuration file by invoking `sudo nano /etc/sysctl.conf` At the end of the file add the following lines and save.

```
kernel.shmmax = 4294967296
net.core.rmem_max = 8388608
net.core.wmem_max = 8388608
```

Load the settings like this:

```
sudo sysctl -p
```

8.2.3. Setting server time zone and locale

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

```
sudo dpkg-reconfigure tzdata
```

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

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

8.2.4. PostgreSQL installation

Install PostgreSQL 9.3 by invoking:

```
sudo apt-get install postgresql-9.3
```

Switch to the postgres user by invoking:

```
sudo su postgres
```

Create a non-privileged user called *dhis* by invoking:

```
createuser -SDRP dhis
```

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

```
createdb -O dhis dhis2
```

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

8.2.5. PostgreSQL performance tuning

Tuning PostgreSQL is necessary to achieve a high-performing system but is optional in terms of getting DHIS 2 to run. PostgreSQL is configured and tuned through the *postgresql.conf* file which can be edited like this:

```
sudo nano /etc/postgresql/9.3/main/postgresql.conf
```

and set the following properties:

```
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 = 20MB
```

Determines the amount of memory used for internal sort and hash operations. This setting is per connection, per query so a lot of memory may be consumed if raising this too high. Setting this value correctly is essential for DHIS 2 aggregation performance.

```
maintenance_work_mem = 512MB
```

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

```
effective_cache_size = 8000MB
```

An estimate of how much memory is available for disk caching by the operating system (not an allocation) and is 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_segments = 32
```

PostgreSQL writes new transactions to a log file called WAL segments which are 16MB in size. When a number of segments have been written a checkpoint occurs. Setting this number to a larger value will thus improve performance for write-heavy systems such as DHIS 2.

```
checkpoint_completion_target = 0.8
```

Determines the percentage of segment completion before a checkpoint occurs. Setting this to a high value will thus spread the writes out and lower the average write overhead.

```
wal_buffers = 16MB
```

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 DHIS 2.

```
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.

Restart PostgreSQL by invoking `sudo /etc/init.d/postgresql restart`

8.2.6. Database configuration

The database connection information is provided to DHIS 2 through a configuration file called *hibernate.properties*. Create this file and save it in a convenient location. A configuration file for PostgreSQL corresponding to the above setup has these properties:

```
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc:postgresql:dhis2
hibernate.connection.username = dhis
hibernate.connection.password = xxxx
hibernate.hbm2ddl.auto = update
encryption.password = xxxx
```

The *encryption.password* property is the password used when encrypting and decrypting data in the database. It applies for version 2.16 and later. 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. If the database is copied to another server the encryption password must be identical. Remember to set a strong password of at least 8 characters. A system-provided password will be used if not set in the configuration file, this can however not be considered secure.

A common mistake is to have a white-space after the last property value so make sure there is no white-space at the end of any line. Also remember that this file contains the clear text password for your DHIS 2 database so it needs to be protected from unauthorized access. To do this invoke the following command which ensures that only the dhis user which owns the file is allowed to read it:

```
chmod 0600 hibernate.properties
```

8.2.7. Install Java

Install Java by invoking the following command:

```
sudo apt-get install openjdk-7-jdk
```

Check that your installation is okay by invoking:

```
java -version
```

8.2.8. Install Tomcat and DHIS2

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

```
sudo apt-get install tomcat7-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:

```
tomcat7-instance-create tomcat-dhis
```

This will create an instance in a directory called *tomcat-dhis*. Note that the tomcat7-user package allows for creating any number of dhis instances if that is desired.

Next edit the file *tomcat-dhis/bin/setenv.sh* and add the lines below. The first line will set the location of your Java Runtime Environment, the second will dedicate memory to Tomcat and the third will set the location for where DHIS 2 will search for the *hibernate.properties* configuration file. Please check that the path the Java binaries are correct as they might vary from system to system, e.g. on AMD systems you might see */java-7-openjdk-amd64* Note that you should adjust this to your environment:

```
export JAVA_HOME='/usr/lib/jvm/java-7-openjdk'
export JAVA_OPTS='-Xmx7500m -Xms4000m -XX:MaxPermSize=500m -XX:PermSize=300m'
export DHIS2_HOME='/home/dhis/config'
```

The Tomcat configiration 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. If UTF-8 encoding of request data is needed, make sure that the *URIEncoding* attribute is set to *UTF-8*.

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443"
  URIEncoding="UTF-8" />
```

The next step is to download the DHIS 2 WAR file and place it into the webapps directory of Tomcat. You can download the DHIS version 2.15 WAR release like this (replace 2.15 with your preferred version if necessary):

```
wget https://www.dhis2.org/download/releases/2.15/dhis.war
```

Move the WAR file into the Tomcat webapps directory. We want to call the WAR file ROOT.war in order to make it available at localhost directly without a context path:

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

8.2.9. Running DHIS2

Make the Tomcat startup script executable by invoking:

```
chmod +x tomcat-dhis/bin/*.sh
```

DHIS 2 can now be started by invoking:

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

DHIS 2 can be stopped by invoking:

```
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 DHIS instance at the following URL:

```
http://localhost:8080
```

8.3. 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 DHIS on the same server. It also makes it possible to change the internal server setup without affecting clients.
- The DHIS 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.

8.3.1. Basic setup for nginx

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

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

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

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

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

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

nginx configuration is built around a hierarchy of blocks representing http, server and location, where each block inherit settings from parent blocks. The following snippet will configure nginx to proxy pass (redirect) requests from port 80 (which is the port nginx will listen on by default) to our Tomcat instance. Include the following configuration in nginx.conf:

```
http {
  gzip on; # Enables compression

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

    # Serve static files

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

    # Proxy pass to servlet container

    location / {
      proxy_pass        http://localhost:8080/;
      proxy_redirect    off;
      proxy_set_header  Host               $host;
      proxy_set_header  X-Real-IP          $remote_addr;
      proxy_set_header  X-Forwarded-For    $proxy_add_x_forwarded_for;
      proxy_set_header  X-Forwarded-Proto  http;
    }
  }
}
```

You can now access your DHIS 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" ... >
```

8.3.2. Enabling SSL on nginx

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

```
sudo apt-get install openssl
```

To configure nginx to use SSL you will need a proper SSL certificate from an SSL provider. The cost of a certificate varies a lot depending on encryption strength. An affordable certificate from Rapid SSL Online should serve most

purposes. To generate the CSR (certificate signing request) you can invoke the command below. When you are prompted for the *Common Name*, enter the fully qualified domain name for the site you are securing.

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

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

Below is an nginx server block where the certificate files are named server.crt and server.key. Since SSL connections usually occur on port 443 (HTTPS) we pass requests on that port (443) on to the DHIS 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 DHIS 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

  # HTTP server - rewrite to force use of SSL

  server {
    listen     80;
    rewrite    ^ https://<server-url>$request_uri? permanent;
  }

  # HTTPS server

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

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

    ssl_session_cache    shared:SSL:20m;
    ssl_session_timeout  10m;

    ssl_protocols              TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers                RC4:HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    # Serve static files

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

    # Proxy pass to servlet container

    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;
    }
  }
```

```
}
```

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

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

8.3.3. Enabling caching and SSL on 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 {
  # ...
  root              /home/dhis/tomcat/webapps/ROOT; # Update path!
  proxy_cache_path  /var/cache/nginx  keys_zone=dhis:250m  inactive=1d;
  gzip              on;

  # HTTP server - rewrite to force use of HTTPS

  server {
    listen     80;
    rewrite    ^ https://<server-ip>$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_timeout  30m;

    ssl_protocols              SSLv2 SSLv3 TLSv1;
    ssl_ciphers                HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    # Serve static files

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

    # Proxy pass to servlet container and potentially cache response

    location / {
      proxy_pass        http://localhost:8080/;
      proxy_redirect    off;
      proxy_set_header  Host               $host;
      proxy_set_header  X-Real-IP          $remote_addr;
      proxy_set_header  X-Forwarded-For    $proxy_add_x_forwarded_for;
      proxy_set_header  X-Forwarded-Proto  https;
      proxy_cache       dhis;
```

```
    }
  }
}
```

Important

Be aware that a server side cache shortcuts the DHIS 2 security features in the sense that requests which hit the server side cache will be served directly from the cache outside the control of DHIS 2 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 cahce is not recommended.

8.3.4. Starting tomcat on boot-time

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

```
#!/bin/sh
#Tomcat init script

HOME=/home/dhis/tomcat/bin

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

Move the script to the init script directory and make them executable by invoking:

```
sudo mv tomcat /etc/init.d
sudo chmod +x /etc/init.d/tomcat
```

Next make sure the tomcat init script will be invoked during system startup and shutdown:

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

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

8.3.5. Making resources available with nginx

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

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

```
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;
  }
}
```

8.3.6. App setup with nginx

DHIS 2 supports installation of apps. To avoid having to re-install your apps every time you update and replace the DHIS 2 WAR file it is beneficial to deploy apps on the server file system outside the DHIS 2 webapp directory. To install apps directly in nginx you can follow these steps. First create a directory which will be used as the app installation folder.

```
sudo mkdir /usr/share/nginx/apps
```

Make sure that the directory is owned by the user which runs Tomcat in order to allow the Tomcat process to save apps to this directory when they are uploaded. If the user running Tomcat is *dhis* you can use the below command.

```
sudo chown dhis:dhis /usr/share/nginx/apps
```

You must add an *apps* location in the nginx.conf configuration file like below. Note that we omit the apps directory itself in the root directive.

```
server {
  ...
  location /apps/ {
    root      /usr/share/nginx;
    expires   max;
  }
  ...
}
```

Finally navigate to the app configuration screen in your DHIS 2 instance by going to App management > Settings and adjust the settings accordingly.

The *app installation folder* should point to the base directory path on the server file system:

```
/usr/share/nginx/apps
```

The *app base URL* should point to the URL where DHIS 2 can find the apps. Replace www.domain.com with your real domain name and according to how you have deployed DHIS 2.

```
http://www.domain.com/apps
```

You should now be ready to upload apps from the App management screen. Remember to reload the nginx configuration.

8.3.7. Basic reverse proxy setup with Apache

The Apache HTTP server is the most common

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

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

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

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

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

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

Now, we need to make the adjustments to the Apache HTTP server which will answer requests on port 80 and pass them to the Tomcat server through an AJP connector. Edit the file `/etc/apache2/mods-enabled/proxy.conf` so that it looks like the example below. Be sure that the port defined in the configuration file matches the one from Tomcat.

```
<IfModule mod_proxy.c>

ProxyRequests Off
ProxyPass /dhis  ajp://localhost:8009/dhis
ProxyPassReverse /dhis  ajp://localhost:8009/dhis

<Location "/dhis">
  Order allow,deny
  Allow from all
</Location>
</IfModule>
```

You now can restart Tomcat and the Apache HTTPD server and your DHIS 2 instance should be available on http://*myserver*/dhis where *myserver* is the hostname of your server.

8.3.8. Basic load-balancing with Apache and Tomcat

Load balancing may be employed to more evenly distribute system load across multiple Tomcat instances in situations where user load is too high to be handled by a single server instance. In this example, we will create a simple load-balanced architecture using "sticky sessions" to distribute users across two instances of Tomcat.

First, we need at least two instances of Tomcat running DHIS2, which are connected to the same database. There are various architectures, such as running the application servers (Tomcat) on separate (virtual) machines connected to a single database server, or perhaps running multiple Tomcat instances and a database on a single-high capacity machine in situations with I/O is not an issue, but when CPU usage of a single Tomcat instance limits overall system performance. In this scenario, we will configure connect two Tomcat instances running on the same machine to a single

database through a load-balanced reverse proxy. Apache will take care of the details of determining which Tomcat instance a particular client is interfaced to with the

The first step is to configure our Tomcat instances. The previous sections have detailed how this should be done. Importantly, both Tomcat instances should be configured to use the same database server. Some modifications need to be made to the server.xml file of each Tomcat instance, which will be used to uniquely identify each instance. Two copies of Tomcat should be extracted to a directory of your choice. Modify the server.xml file so that the following lines are unique for each instance.

```
<Server port="8005" shutdown="SHUTDOWN">
...
<Connector port="8009" protocol="AJP/1.3" redirectPort="8444" />
...

<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
```

The important parameters here are the server port, the AJP connector port, and the *jvmRoute* identifier. The *jvmRoute* identifier will be appended to the JSESSIONID so that Apache will know which Tomcat instance a particular session should be routed to. The parameters must be unique for each Tomcat instance. After configuring Tomcat, setup DHIS2 according to the normal procedures detailed in other sections.

Next, we will configure the Apache HTTP server to perform load balancing. Incoming client requests will be assigned to one of the instances with a sticky session. Alter the `/etc/apache2/apache2.conf` file (or other appropriate file depending on your exact configuration) to define a proxy load balancer and a proxy and reverse proxy path. Note that the port numbers and *route* parameters must match the Tomcat port and *jvmRoute* parameters which were defined earlier in the Tomcat configuration.

```
<Proxy balancer://dhiscluster>
Order Allow,Deny
Allow from all
</Proxy>

<Proxy balancer://dhiscluster>
BalancerMember ajp://127.0.0.1:8009/dhis route=dhis1
BalancerMember ajp://127.0.0.1:9009/dhis route=dhis2

ProxySet lbmethod=byrequests
ProxySet stickysession=JSESSIONID
</Proxy>

ProxyVia Off
ProxyPass /dhis/ balancer://dhiscluster/ stickysession=JSESSIONID nofailover=on

ProxyPassReverse /dhis/ balancer://dhiscluster/ stickysession=JSESSIONID|jsessionid
```

Finally, start both Tomcat instances, and then restart Apache HTTP.

This example demonstrates how to implement a simple load balanced system with sticky sessions using Apache HTTP server.

8.3.9. Basic SSL encryption with Apache

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

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

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

```
openssl req -new -key server.key -out server.csr
```

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

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

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

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

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

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

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

```
<VirtualHost *:443>
        ServerAdmin wemaster@mydomain.org
       SSLEngine On
       SSLCertificateFile /etc/apache2/ssl/server.crt
       SSLCertificateKeyFile /etc/apache2/ssl/server.key
...
```

Be sure that the *:80 section of this file is changed to port *:443, which is the default SSL port. Also, be sure to change the ServerAdmin to the webmaster's email. Lastly, we need to be sure that the hostname is setup properly in /etc/hosts. Just under the "localhost" line, be sure to add the server's IP address and domain name.

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

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

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

8.4. DHIS 2 Live setup

The DHIS 2 Live package is extremely convenient to install and run. It is intended for demonstrations, for users who want to explore the system and for small, offline installations typically at districts or facilities. It only requires a Java Runtime Environment and runs on all browsers except Internet Explorer 7 and lower.

To install start by downloading DHIS 2 Live from *http://dhis2.org* and extract the archive to any location. On Windows click the executable archive. On Linux invoke the startup.sh script. After the startup process is done your default web browser will automtically be pointed to *http://localhost:8082* where the application is accessible. A system tray menu is accessible on most operating systems where you can start and stop the server and start new browser sesssions. Please note that if you have the server running there is no need to start it again, simply open the application from the tray menu.

DHIS 2 Live is running on an embedded Jetty servlet container and an embedded H2 database. However it can easily be configured to run on other database systems such as PostgreSQL. Please read the section above about server installations for an explanation of the database configuration. The *hibernate.properties* configuration file is located in the *conf* folder. Remember to restart the Live package for your changes to take effect. The server port is 8082 by default. This can be changed by modifying the value in the *jetty.port* configuration file located in the *conf* directory.

8.5. Backup

Doing automated database backups for information systems in production is an absolute must, and might have uncomfortable consequences if ignored. Backups have two main purposes: The primary is data recovery in case data is lost, the secondary purpose is archiving of data for a historical period of time.

Backup should be central in a disaster recovery plan. Even though such a plan should cover additional subjects, the database is the key component to consider since this is where all data used in the DHIS 2 application is stored. Most other parts of the IT infrastructure surrounding the application can be restored based on standard components.

There are of course many ways to set up backup; however the following describes a setup where the database is copied into a dump file and saved on the file system. This can be considered a *full* backup. The backup is done with a *cron job*, which is a time-based scheduler in Unix/Linux operating systems.
You can download both files from http://dhis2.com/download/pg_backup.zip

The cron job is set up with two files. The first is a *script* which performs the actual task of backup up the database. It uses a PostgreSQL program called *pg_dump* for creating the database copy. The second is a crontab file which runs the backup script every day at 23:00. Note that this script backs up the database file to the local disk. It is strongly recommended to store a copy of the backup at a location outside the server where the application is hosted. This can be achieved with the *scp* tool. Make sure that you have set the system date correctly on your server.

8.6. Working with the PostgreSQL database

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

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

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

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

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

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