|  |
| --- |
| Kund/Projekt |
| OpeneMap2 |
| Dokumenttyp |
| Systemdokumentation |
| Titel |
| Systemdokumentation OpeneMap2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Dok ID |  | Datum | 2018-01-17 |
| Upprättad av | Iman Tahriri | Ämne | Systemdokumentation OpeneMap2 |
| Dokument-  ansvarig | Lina Riddarström | Version  D | Filnamn |

Table of Contents

[1. Change log 3](#_Toc505615080)

[2. Installing Apache Web Server on Ubuntu 16.04 3](#_Toc505615081)

[2.1 Introduction 3](#_Toc505615082)

[2.2 Prerequisites 3](#_Toc505615083)

[2.2.1 Connecting to the server remotely 3](#_Toc505615084)

[2.2.2 Transferring files from a Windows machine to a remote Linux machine 4](#_Toc505615085)

[2.3 Installing Apache 6](#_Toc505615086)

[2.4 Manage the Apache Process 7](#_Toc505615087)

[2.5 Getting Familiar with Important Apache Files and Directories 8](#_Toc505615088)

[2.5.1 Content 8](#_Toc505615089)

[2.5.2 Server Logs 8](#_Toc505615090)

[2.6 Folder Structure 8](#_Toc505615091)

[3. Installing Node.js on Ubuntu 9](#_Toc505615092)

[3.1 Introduction 9](#_Toc505615093)

[3.2 Prerequisites 9](#_Toc505615094)

[3.3 Installing Node.js 9](#_Toc505615095)

[4. Installing Origo Server 10](#_Toc505615096)

[5. Building OrigoMap (Client) and Hosting the Static Files 12](#_Toc505615097)

[6. Building OrigoAdmin and Hosting the Static Files 12](#_Toc505615098)

[7. Hosting Origo Node.js Server in Apache 13](#_Toc505615099)

[7.1 Introduction 13](#_Toc505615100)

[7.2 Running the Server as a Service 13](#_Toc505615101)

[7.2.1 Installing PM2 13](#_Toc505615102)

[7.2.2 Starting Application 13](#_Toc505615103)

[7.2.3 Other PM2 Usage 14](#_Toc505615104)

[7.3 Configuring Apache to Serve as a Reverse proxy 15](#_Toc505615105)

[8. Conclusion 16](#_Toc505615106)

# Change log

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| A | 180115 | Iman Tahriri | First version |
| B | 180115 | Iman Tahriri | Updated table of contents |
| C | 180116 | Iman Tahriri | Updated table of contents |
| D | 180205 | Iman Tahriri | Updated table of contents & section 4 |

# Installing Apache Web Server on Ubuntu 16.04

## Introduction

The Apache HTTP server is the most widely-used web server in the world. It provides many powerful features including dynamically loadable modules, robust media support, and extensive integration with other popular software.

In this guide, we'll discuss how to install an Apache web server on your Ubuntu 16.04 server.

## Prerequisites

Before you begin this guide, you should have a regular, non-root user with sudo privileges configured on your server. Additionally, you will need to configure a basic firewall to block non-essential ports. When you have an account available, log in as your non-root user to begin.

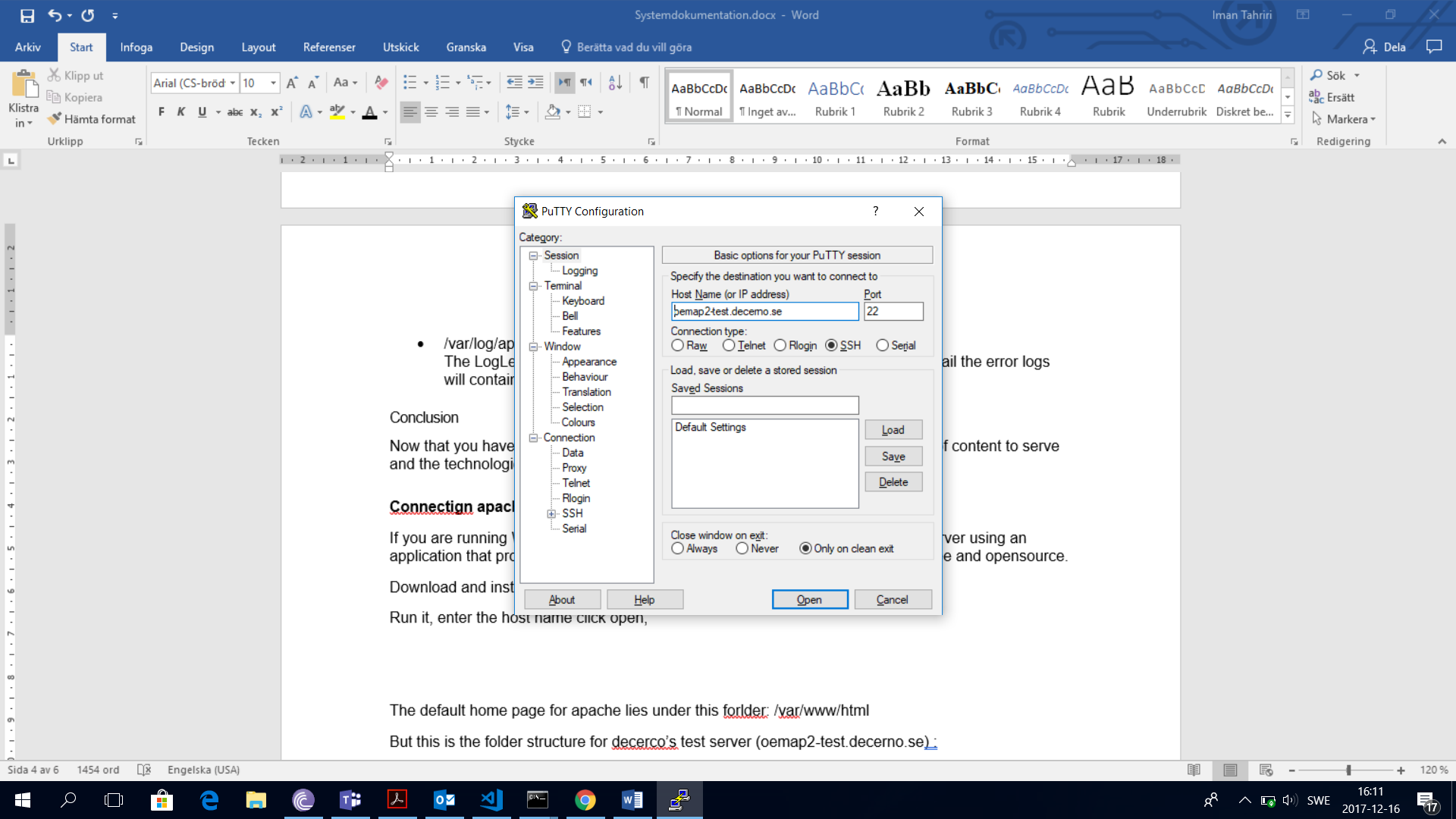
### Connecting to the server remotely

Note: if you are NOT connecting to a remote machine in order to configure your server, you can skip ahead sections 2.2.1 and 2.2.2.

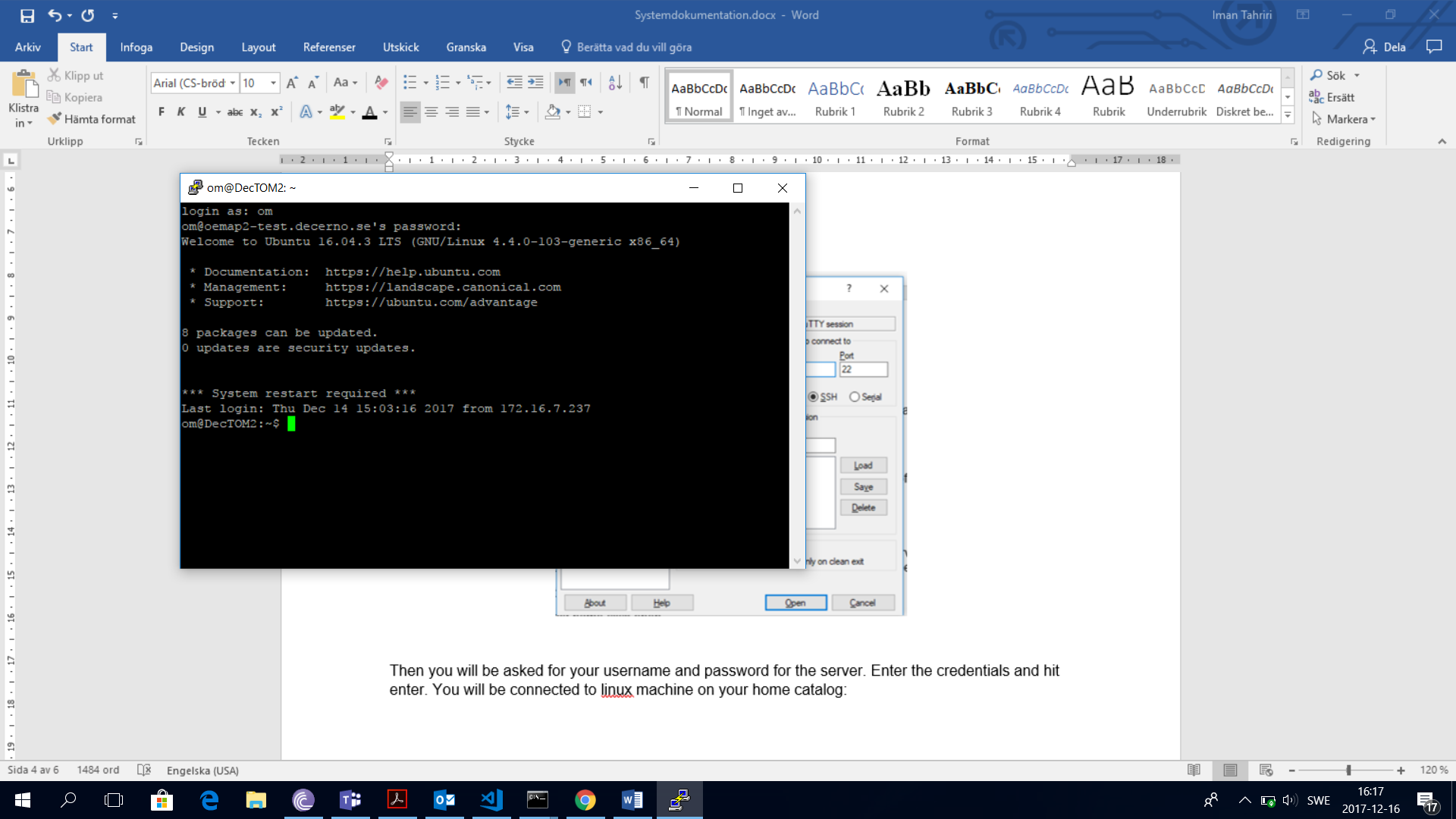
If you are running Windows as the OS, then you need to connect to the apache server using an application that provides such a functionality. Recommended is PuTTY which is free and opensource.

Download and install it from here: <http://www.putty.org/>

Run it, enter the host name and click open:



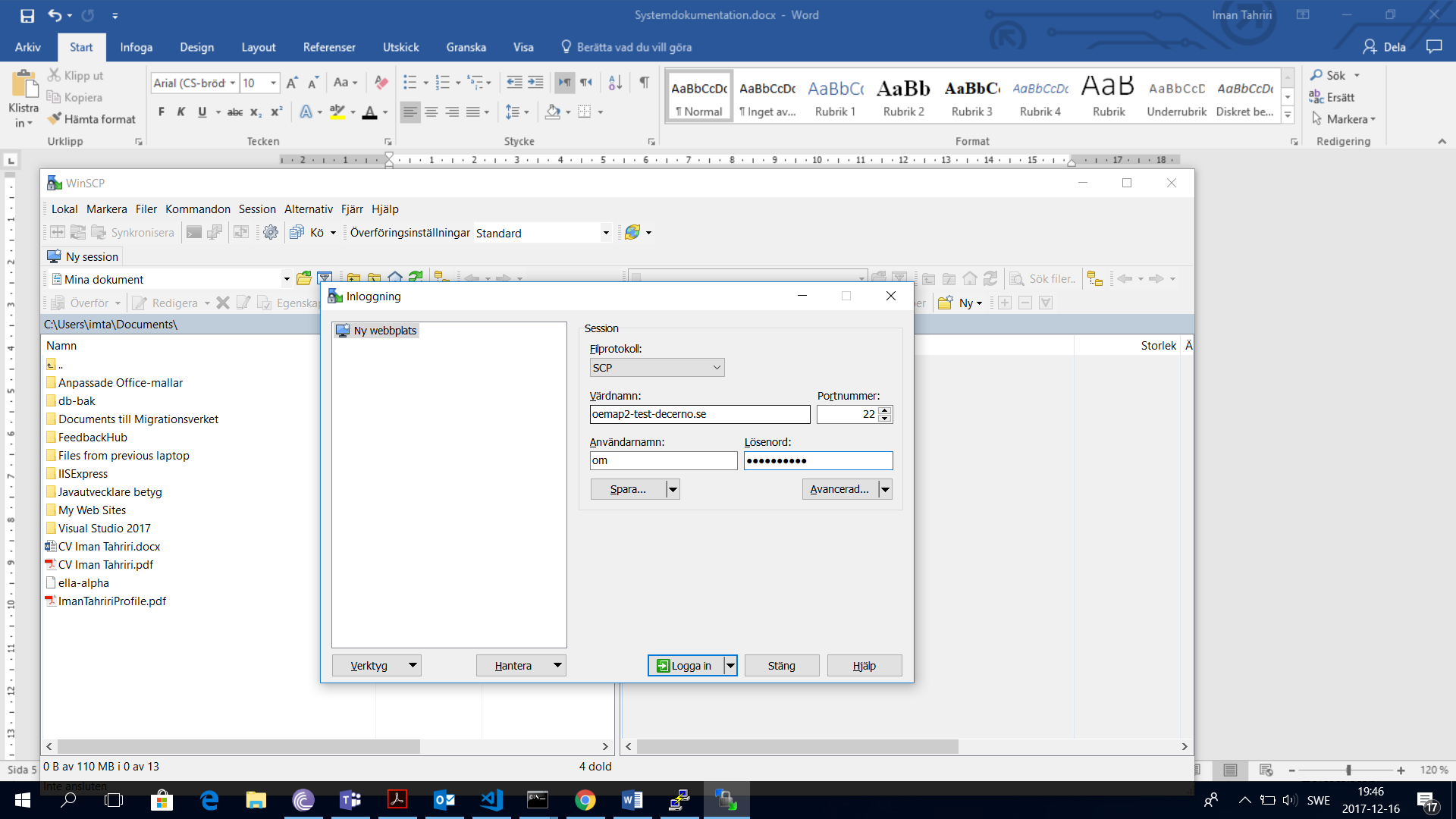
Then you will be asked for your username and password for the server. Enter the credentials and hit enter. You will be connected to the Linux machine on your home catalog:



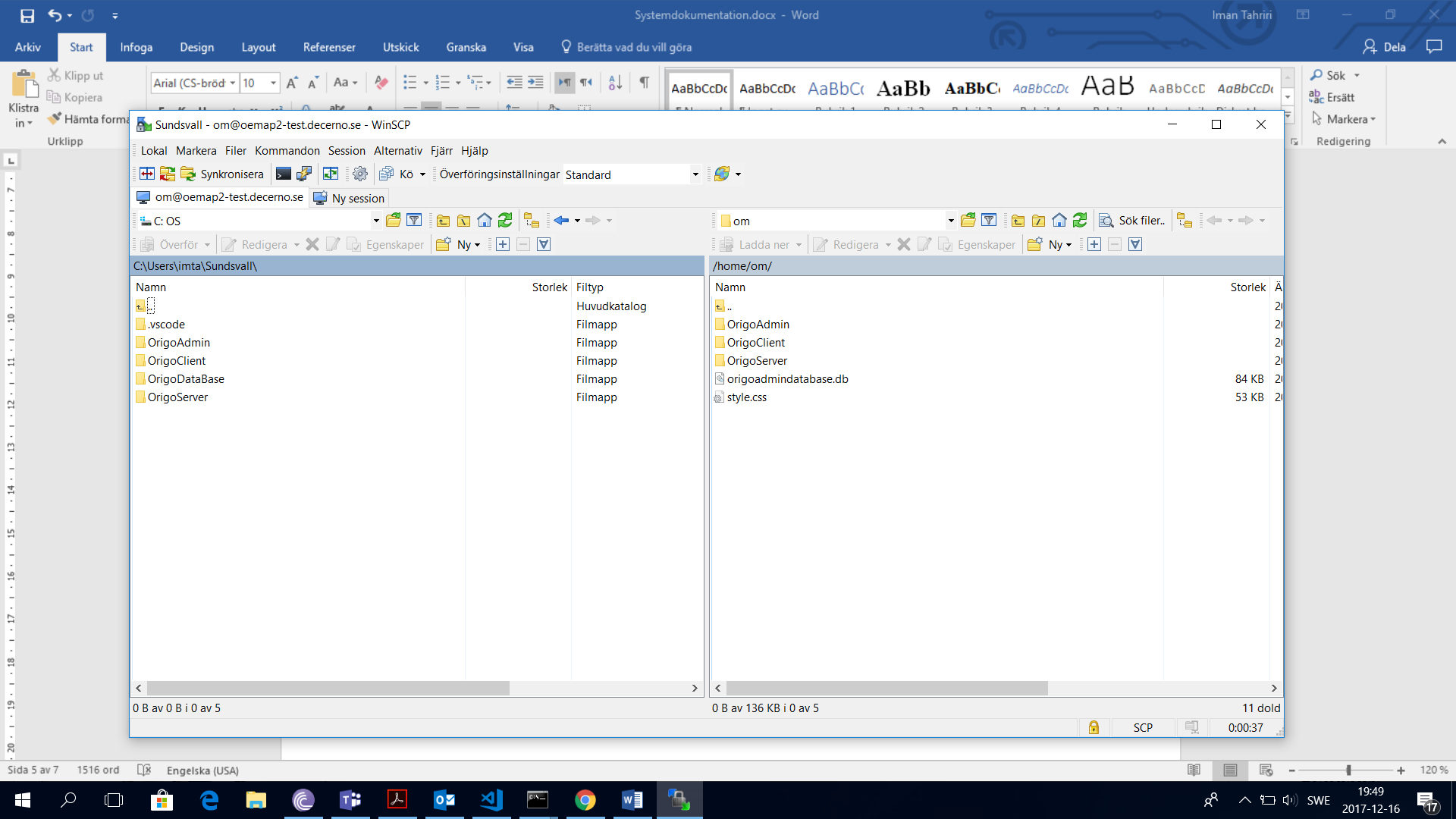
### Transferring files from a Windows machine to a remote Linux machine

To transfer some files from a Windows machine to Ubuntu, download and install WinSCP from this address <https://winscp.net/eng/download.php>

Run the application and provide it host address, your username and password. Choose SCP for protocol:



Hit login and you will have Windows file system in the left panel and Linux file system on the right.



Simply drag and drop the files and folder to the right panel and that is it! You can now find those files in your Linux machine’s home catalog. Now from the terminal you can run the following command to copy all contents of a folder to another one:

* $ sudo cp -r <source folder> <destination folder>

For example

* $ sudo cp -r ~om/OrigoAdmin/\* /var/www/origo/pub/origoadmin

will copy all files and folder from home folder of the user “om”, to the origoadmin folder in the specified address. Note that -r stands for *recursive* and indicates that everything including subfolders should be copied.

These are some other useful commands to work with in the Linux terminal:

* ls <destination> : list of files and folders of the destination folder
* cd <destination> : move to the destination folder.
* cd .. : move one folder up (the parent folder).
* rm <destination> : remove a file or folder.
* mv <source> <destination> : move a file or folder to the destination folder.
* nano <file name> : to edit the contents of a text file.

## Installing Apache

Apache is available within Ubuntu's default software repositories, so we will install it using conventional package management tools.

We will begin by updating the local package index to reflect the latest upstream changes. Afterwards, we can install the apache2 package:

* $ sudo apt-get update
* $ sudo apt-get install apache2

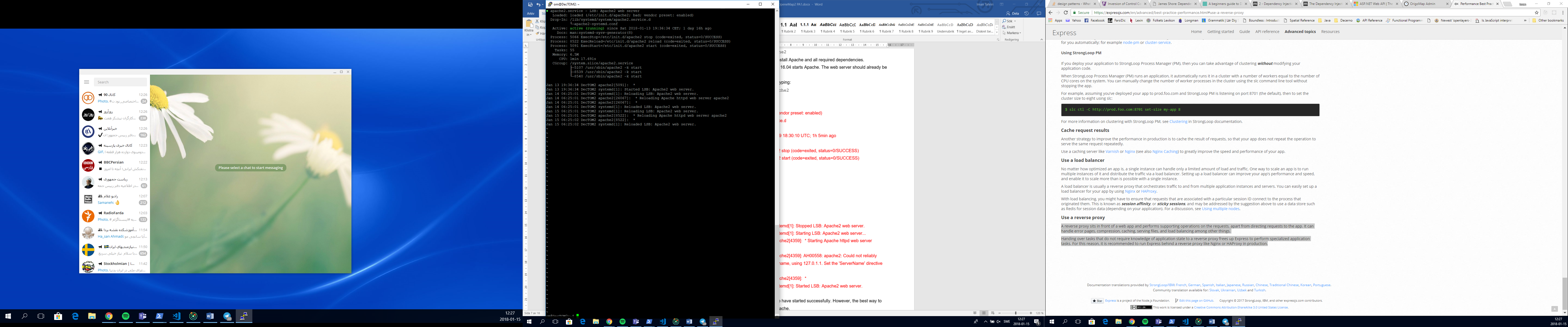
After confirming the installation, apt-get will install Apache and all required dependencies.

At the end of the installation process, Ubuntu 16.04 starts Apache. The web server should already be up and running.

We can make sure the service is running by typing:

* $ sudo systemctl status apache2

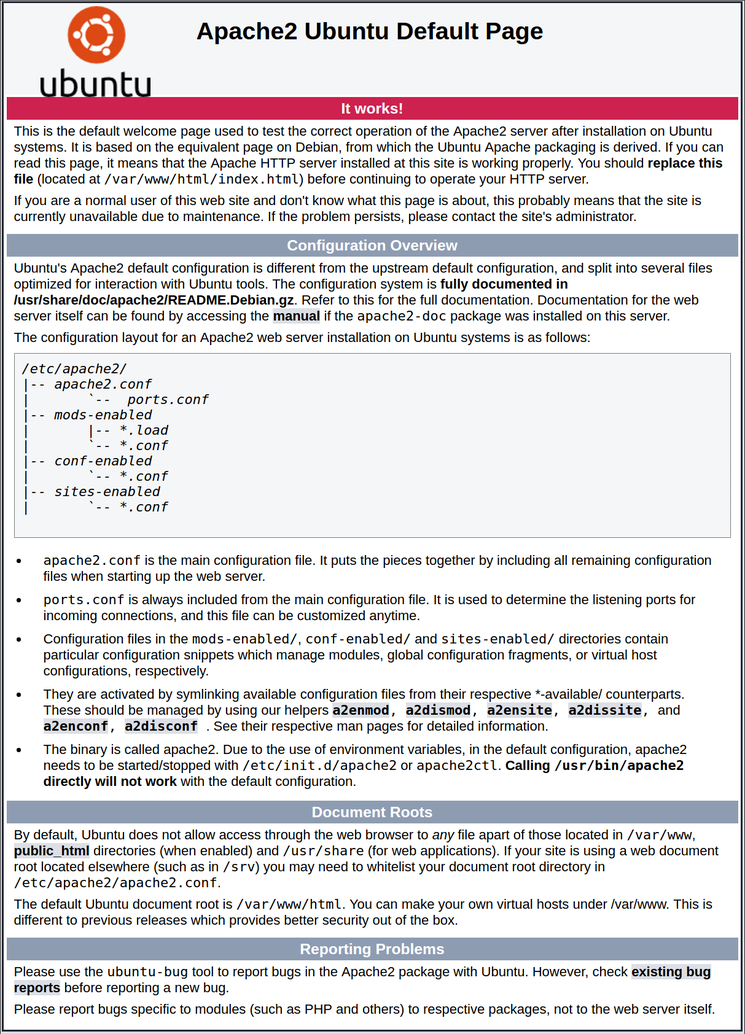
Output should be something like this:



As you can see above, the service appears to have started successfully. However, the best way to test this is to actually request a page from Apache.

You can access the default Apache landing page to confirm that the software is running properly. You can access this through your server's domain name or IP address.

You should see the default Ubuntu 16.04 Apache web page, which should look something like this:



This page is simply included to show that Apache is working correctly. It also includes some basic information about important Apache files and directory locations.

## Manage the Apache Process

Now that you have your web server up and running, we can go over some basic management commands.

To stop your web server, you can type:

* $ sudo systemctl stop apache2

To start the web server when it is stopped, type:

* $ sudo systemctl start apache2

To stop and then start the service again, type:

* $ sudo systemctl restart apache2

If you are simply making configuration changes, Apache can often reload without dropping connections. To do this, you can use this command:

* $ sudo systemctl reload apache2

By default, Apache is configured to start automatically when the server boots. If this is not what you want, you can disable this behavior by typing:

* $ sudo systemctl disable apache2

To re-enable the service to start up at boot, you can type:

* $ sudo systemctl enable apache2

Apache should now start automatically when the server boots again.

## Getting Familiar with Important Apache Files and Directories

If you have no experience working with Apache before, it is good to take a few minutes to familiarize yourself with a few important directories and files.

### Content

* /var/www/html: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the /var/www/html directory. We are going to change this later on by altering Apache configuration files.

### Server Logs

* /var/log/apache2/access.log: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
* /var/log/apache2/error.log: By default, all errors are recorded in this file. The LogLevel directive in the Apache configuration specifies how much detail the error logs will contain.

## Folder Structure

The default home page for apache lies under this forlder: /var/www/html.

Following is the recommended folder structure which is uses for decerno’s test server (oemap2-test.decerno.se) as well:

* /var/www/origo: database file origoadmindatabase.db will be in this folder. In this way database file will be existing independent of the server and if server needs to be reinstalled for some reason, data will remain intact.
* /var/www/origo/origoserver: in this folder lies the Node.js server that will run on a local host and takes care of database communications.
* /var/www/origo/origomapstate: in this folder lies json files that hold information about a very specific state of the map that can be shared with other users like zoom level, turned on layers etc.
* /var/www/origo/pub: our apache home folder which is open to access from the internet.
* /var/www/origo/pub/origomap: will serve origomap static files.
* /var/www/origo/pub/origoadmin: will serve origoadmin static files.

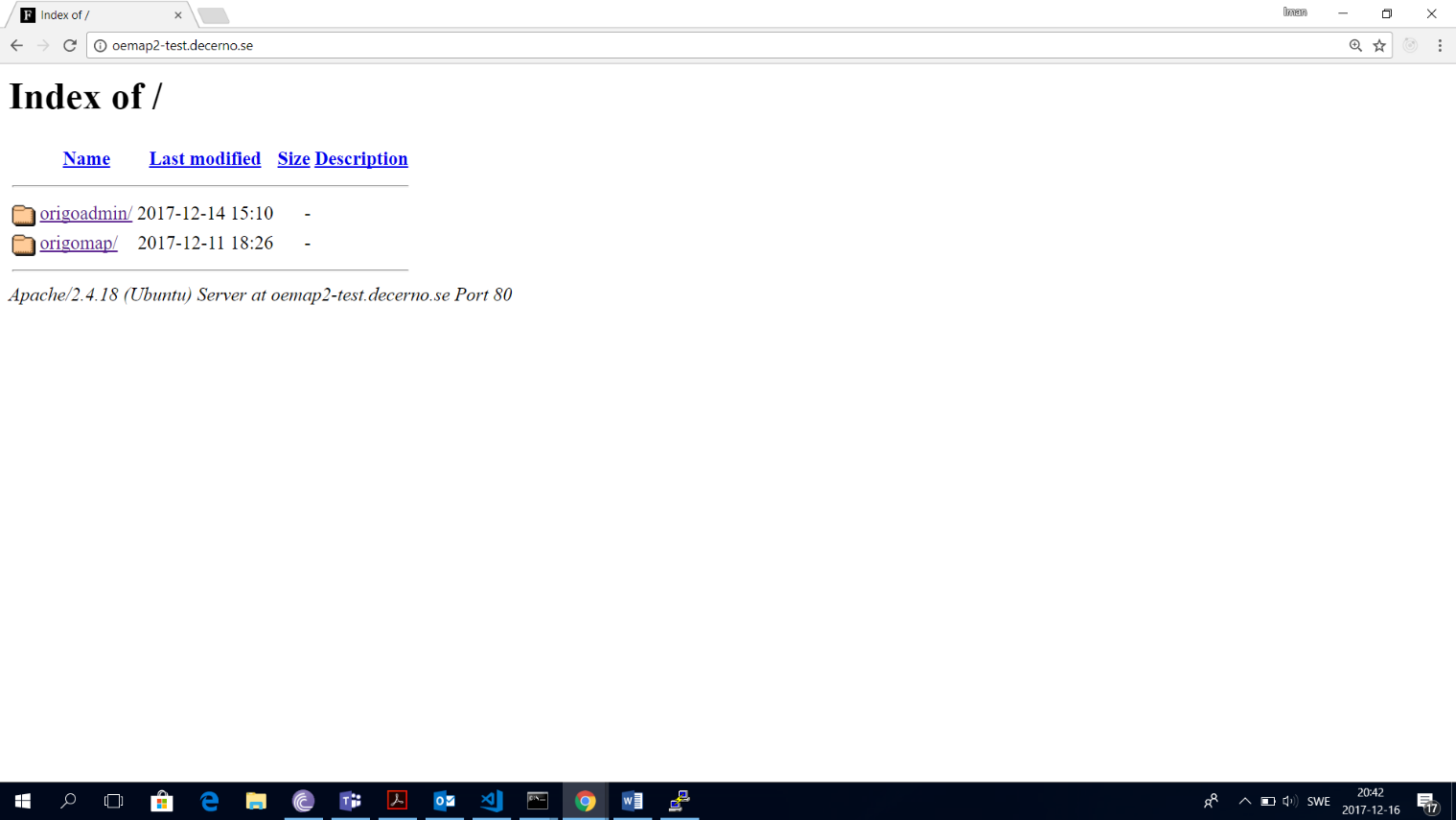
Now we need to change the home folder to /var/www/origo/pub so that only origoadmin and origoserver are open to access from the internet, not the database and not the origoserver.

To do so, we need to change the contents of the configuration file of the Apache. By default this files in named 000.default.conf and lies in /etc/apache2/sites-available. Run the following command:

* $ sudo nano /etc/apache2/sites-available/000.default.conf

and edit the line DocumentRoot /var/www/html to DocumentRoot /var/www/origo/pub, save and exit.

Now if you navigate to the host name from the browser, contents of the pub folder will be served. In our case origoadmin and origomap:



# Installing Node.js on Ubuntu

## Introduction

Node.js is an open source JavaScript runtime environment for easily building server-side and networking applications. Node.js applications can be run at the command line, but we'll focus on running them as a service, using PM2 process manager, so that they will automatically restart on reboot or failure, and can safely be used in a production environment.

## Prerequisites

This guide assumes that you have the following:

* An Ubuntu 16.04 server, configured with a non-root user with sudo privileges.
* A domain name pointed at your server's public IP.

Let's get started by installing the Node.js runtime on your server.

## Installing Node.js

We will install the latest LTS release of Node.js, using the [NodeSource](https://github.com/nodesource/distributions) package archives.

First, you need to install the NodeSource PPA in order to get access to its contents. Make sure you're in your home directory, and use curl to retrieve the installation script for the Node.js 6.x archives:

* $ cd ~
* $ curl -sL https://deb.nodesource.com/setup\_6.x -o nodesource\_setup.sh

You can inspect the contents of this script with nano (or your preferred text editor):

* $ nano nodesource\_setup.sh

And run the script under sudo:

* $ sudo bash nodesource\_setup.sh

The PPA will be added to your configuration and your local package cache will be updated automatically. After running the setup script from nodesource, you can install the Node.js package:

* $ sudo apt-get install nodejs

The *nodejs* package contains the *nodejs* binary as well as *npm*, so you don't need to install *npm* separately. However, in order for some *npm* packages to work (such as those that require compiling code from source), you will need to install the build-essential package:

* $ sudo apt-get install build-essential

The Node.js runtime is now installed, and ready to run any JavaScript application.

To confirm that it is installed, or to check if it is already installed on a system, run the following command:

* $ node -v

you should see the version of installed Node.js application in the output.

**Note:** When installing from the NodeSource PPA, the Node.js executable is called nodejs, rather than node.

When running, our node server will be listening on the specified address and port which is localhost:3001, and takes care of database communications. By database communications we mean reading, writing, updating and removing data. Since we're listening on **localhost**, remote clients still won't be able to connect to our application until we configure the proxy settings for Apache. See section [hosting a Origo nodejs server in apache](#_Hosting_Origo_Node.js).

# Installing Origo Server

As described in section 2.6, we are going to install Origo server under this path:

* /var/www/origo/origoserver.

First of all, we need to copy the source code of the Origo server to this folder. the easiest way is to directly clone the repository to this folder. type the following command on the apache terminal:

* $ sudo git clone <http://decdtfs:8080/tfs/DefaultCollection/OpenEmap2/_git/OrigoServer>

you need to provide it with your git credentials to be able to clone the repository.

If you do not have access to the git repository, then you need to copy the latest version of the Origo server from ftp server. Copy and paste it to /var/www/origo/origoserver. See section 2.2.2 for more information regarding how to copy files to your Linux machine.

After the source is copied either way, run the following command in the terminal to install all dependency packages needed to run the server:

* $ sudo npm install

and after that run the following command to start the server:

* $ sudo npm start

If you encountered an error message like this:

**ORMError: Connection lost - driver does not support reconnection**

**at ORM.<anonymous> (C:\Users\imta\Sundsvall\OrigoServer\node\_modules\orm\lib\ORM.js:172:31)**

**at emitOne (events.js:96:13)**

**at Database.emit (events.js:188:7)**

it means that server cannot find the database in the specified path. Make sure that your configurations points to the correct path to the database. To do so, under the folder *conf* open the file *config.js:*

* $ sudo nano /var/www/origo/origoserver/conf/config.js

make sure that relative path for the data base is as follows:

adminDataBase: {

'relativePath' : './origoadmindatabase.db'

}

Save, exit and start the server again.

Moreover, make sure that the path to *origomapstate* folder is set:

mapState: {

'storagePath': './../origomapstate'

}

Note that if you are installing the server on a Linux machine, you need to create the folder and set permission to read and write its content. On a Windows machine folder will be created automatically.

If you see the following output, the application is working properly and listening on the proper address and port:

Origo server listening at http://:::3001

**Note:** Running a Node.js application in this manner will block additional commands until the application is killed by pressing **Ctrl-C**.

If you need to use the command line for some reason, you should open another terminal session on your server.

If you need to stop the server for some reason, from the same terminal that server is running, kill the application by pressing **Ctrl+C**.

Note: running the server in this manner can cause security issues and is NOT recommended. See the section 7: Hosting Origo Node.js Server in Apache.

# Building OrigoMap (Client) and Hosting the Static Files

Clone the Origo client source code repository on your local machine using the command:

* $ sudo git clone [http://decdtfs:8080/tfs/DefaultCollection/OpenEmap2/\_git/Origo](http://decdtfs:8080/tfs/DefaultCollection/OpenEmap2/_git/OrigoAdmin)Client

If you do not have access to the git repository, then you need to copy the latest version of the origomap (or origoclient) zip file from ftp server. Copy, paste and unzip it. Then running the command

* $ sudo npm install

Will install all necessary packages. Then, in order to build minified static files for production, run the following command:

* $ npm run build

Doing so will create all necessary materials under the foler dist. Now all files and folders in this folder should be copied to the server to the address /var/www/origo/pub/origomap. Now if you navigate to the */origomap* folder from the browser, you should see the OrigoMap starting page.

If the contents are then not served properly, check out the network log of the browser, it is possible that you need to copy some other files and folders too. Namely:

The *css/png* and *css/svg* folders from source to the *css* folder of the server.

The *css/styles.css* file from source to the *css* folder of the server.

The *img* folder from source to */origomap* folder.

*Index.json* file from source folder to the server.

It is possible to need a small change in the file *index.html*. Make sure that in the *src* property of the script tag there is no prefix to the file like a folder name or slash. Also, make sure that it is called *origo.min.js*. It should look like this:

* <script src="origo.min.js"></script>

# Building OrigoAdmin and Hosting the Static Files

Clone the OrigoAdmin source code repository on your local machine using the command:

* $ sudo git clone <http://decdtfs:8080/tfs/DefaultCollection/OpenEmap2/_git/OrigoAdmin>

If you do not have access to the git repository, then you need to copy the latest version of the *origoserver* zip file from ftp server. Copy, paste and unzip it. Then running the command

* $ sudo yarn install

Will install all necessary packages.

Before building for production, make sure that in the file *adminApi.js*, the variable *serverUrl* is pointing to the server address and NOT *localhost*. It should look like this:

* const serverUrl = 'http://oemap2-test.decerno.se/admin';

Then, in order to build minified static files for production, run the following command:

* $ yarn build:prod

Doing so will create all necessary materials under the foler *dist/prod*. Now all files and folders in this folder should be copied to the server to the address /var/www/origo/pub/origoadmin.

It is possible to need a small change in the file *index.html*. If the contents of the page are not delivered properly make sure that in the *src* property of the script tag there is no prefix to the file like a folder name or slash. It should look like this but with a different number in file name:

* <script type="text/javascript" src="app.8eafde1f032c16cf3c92.bundle.js"></script>

# Hosting Origo Node.js Server in Apache

## Introduction

This part consists of two tasks, first to run node application as a service so that it runs silently in the background and will restart automatically if it crashes. And then we explain how to configure Apache so that some specific calls to apache webserver will be routed to our local node server. It is called a reverse proxy.

A reverse proxy sits in front of a web app and performs supporting operations on the requests, apart from directing requests to the app. It can handle error pages, compression, caching, serving files, and load balancing among other things.

Handing over tasks that do not require knowledge of application state to a reverse proxy frees up Express, the framework in which Origo server is developed, to perform specialized application tasks. For this reason, it is recommended to run Express behind a reverse proxy in production. That is why we installed Apache, and now we are going configure Origo server to run behind it.

## Running the Server as a Service

### Installing PM2

Now we install PM2, which is a process manager for Node.js applications. PM2 provides an easy way to manage and daemonize applications (run them in the background as a service).

In the terminal run the following command:

* $ sudo npm install -g pm2

The *-g* option tells *npm* to install the module globally, so that it's available system-wide.

### Starting Application

Run this command to start your application running in the background:

* $ pm2 start /var/www/origo/origoserver/app.js

This also adds your application to PM2's process list, which is outputted every time you start an application:

[PM2] Spawning PM2 daemon

[PM2] PM2 Successfully daemonized

[PM2] Starting app.js in fork\_mode (1 instance)

[PM2] Done.

┌──────────┬────┬──────┬──────┬────────┬─────────┬────────┬─────────────┬──────────┐

│ App name │ id │ mode │ pid │ status │ restart │ uptime │ memory │ watching │

├──────────┼────┼──────┼──────┼────────┼─────────┼────────┼─────────────┼──────────┤

│ app │ 0 │ fork │ 3524 │ online │ 0 │ 0s │ 21.566 MB │ disabled │

└──────────┴────┴──────┴──────┴────────┴─────────┴────────┴─────────────┴──────────┘

Use `pm2 show <id|name>` to get more details about an app

PM2 automatically assigns a *app* ***name*** (based on the filename, without the .js extension) and a *PM2****id***. PM2 also maintains other information, such as the **PID** of the process, its current status, and memory usage.

Applications that are running under PM2 will be restarted automatically if the application crashes or is killed, but an additional step needs to be taken to get the application to launch on system startup (boot or reboot). PM2 provides an easy way to do this, by using the startup subcommand.

The startup subcommand generates and configures a startup script to launch PM2 and its managed processes on server boots:

* $ pm2 startup systemd

The last line of the resulting output will include a command that you must run with superuser privileges:

[PM2] Init System found: systemd

[PM2] You have to run this command as root. Execute the following command:

sudo env PATH=$PATH:/usr/bin /usr/lib/node\_modules/pm2/bin/pm2 startup systemd -u om --hp /home/om

Run the command that was generated (similar to the highlighted output above, but with your username instead of om) to set PM2 up to start on boot (use the command from your own output):

* $ sudo env PATH=$PATH:/usr/bin /usr/lib/node\_modules/pm2/bin/pm2 startup systemd -u --hp /home/om

This will create a *systemd* **unit** which runs PM2 for your user on boot. This PM2 instance, in turn, runs ***app.js***.

### Other PM2 Usage

PM2 provides many subcommands that allow you to manage or look up information about your applications. Note that running PM2 without any arguments will display a help page, including example usage, that covers PM2 usage in more detail.

Stop an application with this command (specify the PM2 *App name* or *id*):

* $ pm2 stop app\_name\_or\_id

Restart an application with this command (specify the PM2 *App name* or *id*):

* $ pm2 restart app\_name\_or\_id

The list of applications currently managed by PM2 can also be looked up with the list subcommand:

* $ pm2 list

More information about a specific application can be found by using the *info* subcommand or *show* subcommand (specify the PM2 App name or id):

* $ pm2 show app\_name\_or\_id
* $ pm2 info app\_name\_or\_id

The PM2 process monitor can be pulled up with the *monit* subcommand. This displays the application status, CPU, and memory usage:

* $ pm2 monit

Now the Node.js application is running, and managed by PM2. Next thing to do is to set up a reverse proxy.

## Configuring Apache to Serve as a Reverse proxy

The main part of this lies in Apache’s “Reverse Proxy” functionality.

Apache’s “Reverse Proxy” functionality allows remote servers to be mapped into the space of the local server. The local server does not act as a proxy in the conventional sense but appears to be a mirror of the remote server. The local server is often called a reverse proxy or gateway.

Open up your apache2 configuration. By default, it is located in /etc/apache2/sites-available folder (with a separate file per vhost/site) and has the name 000-default.conf.

This command will open it in an editor:

* $ sudo nano /etc/apche2/000.default.config

Add the following lines inside the <VirtualHost \*:80> directive:

* ProxyRequests off
* ProxyPass /admin <http://127.0.0.1:3001/admin>

Now all requests sent to */admin* will be instead re-routed to the node app, but all other requests will go through to the usual document root. Note that apache will automatically reformat the URL; that is if the user requests */api/a/b/c* then the node app will only receive the request for */a/b/c* without the */api* part as that is not mentioned in the ProxyPass directive for the 2nd parameter.

For all of this to work we need to enable *mod\_proxy* and *mod\_proxy\_http* modules on the Apache server. Run the following commands to do so:

* $ sudo a2enmod proxy
* $ sudo a2enmod proxy\_http

# Conclusion

We now have a Node.js server running behind a reverse proxy on an Apache2 webserver. This reverse proxy setup is flexible enough to provide your users access to other applications or static web content that you want to share. Origo server is running in the background as a service and will restart automatically if it crashes sometime. Static contents are served from /origoadmin and /origomap folders for OrigoAdmin and OrigoMap respectively.