[DAD] Tutorial: Deploy to Ubuntu VM

The purpose of this tutorial is to walk you through the setup and deploy of your evaluation project on an Ubuntu 20.04 VM (the same as provided for each group on the DEI data center).

This tutorial has it's steps grouped by each component of the tech stack we are using LNMP + Node. These can be done out of this order but it's recommended that you follow the structure provided.

1 - Pre Tasks

1.1 - SSH Connection

You should have received an email with the SSH keys associated with your VM, and the instructions on how to connect to that VM. All commands shown are to be done on an SSH connect to the VM, unless specifically stated.

Start by updating the local environment:

```
sudo apt-get update
sudo apt-get upgrade
```

The next step is to create the base structure for our project files. This tutorial assumes the following structure:

```
/
|___var
|   |___www
|   |   |
|   |   |___project
|   |   |   |___laravel     # laravel api code
|   |   |   |___vue         # vue frontend code
|   |   |   |___websocket   # node websocket server code
```

Run the following command to create that infrastructure

```
sudo mkdir -p /var/www/project/{laravel,vue,websocket}
```

2 - PHP

First we need to install PHP and Laravel's required PHP modules.

```
sudo apt-get install php php-cli php-fpm php-common php-dev php-zip php-mbstring php-xml php-apcu php-gd php-curl
```

Then the composer package manager.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cba5b96ac8b64115ac402c8cd292b8a03482574915d1a8') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"

sudo mv composer.phar /usr/local/bin/composer
```

If the command above gives an error, get the command from its source here.

2.1 Check

To check that every thing is working run the following commands:

```
php -v
composer
```

The first should give you the PHP version (>= 7.4.3) and the second should display the composer help.

3 - MySQL

Now for the database. Install the MySQL server.

```
sudo apt-get install mysql-server php-mysql
```

Then run the MySQL security setup script.

```
sudo mysql_secure_installation
```

And provide the following answers:

- Validate Password Component: your choice, yes means passwords need to comply with minimum complexity requirements (and you need to change the root password from the command bellow)
- Define root password: again your choice of password
- Remove anonymous users: Yes
- Disallow root login remotely: Yes (we'll show you how to connect remotely)
- Remove Test Databases: Yes
- Reload Privileges: Yes

The MySQL version on Ubuntu's repositories is 8.0+, and it has a default authentication mechanism that PHP cannot communicate with, so we need to change it. Run the following command.

```
sudo mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'dad';"
```

This modifies the authentication mechanism for the root user and assigns it the password "dad"

3.1 - Check

To check that every thing is working run the following commands:

```
mysql -u root -p
```

And inside the MySQL shell.

```
CREATE DATABASE dad;
```

If the database creation command worked everything is ready.

4 - Node

Next it's time for Node. In this production environment we will only use Node for the Web Sockets component of the project as the Vue application will be deployed as a static build.

For managing the runtime of our Node app we will use the PM2 package.

So install all the necessary components. First Node itself, and in Ubuntu's case we will be using NodeSource for the binary install.

```
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
sudo apt-get install nodejs
```

Before installing PM2 we need to configure a global node modules folder for our user, if not we need rely on sudo to install and use global modules.

```
mkdir ~/.npm_global
npm config set prefix '~/.npm_global'
echo 'export PATH=$PATH:~/.npm_global/bin' >> ~/.bashrc
source ~/.bashrc
```

And then install PM2.

```
npm install --global pm2
```

4.1 - Check

Run these commands:

```
node -v
npm -v
pm2
```

The first two should return their respective versions (16+, 8+) and the last shows the PM2 help information.

5 - Nginx

The last piece of the puzzle is Nginx. This is going to be our Web Server (for the Vue Frontend) and Reverse Proxy (for the Laravel Backend and Web Sockets Server).

Let's install it.

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

On the Ubuntu Server, Apache (another web server) is installed by default, so the next step is to disable it and enable Nginx.

```
sudo systemctl stop apache2
sudo systemctl disable apache2
sudo systemctl enable nginx
sudo systemctl start nginx
```

5.1 Check

To check if everything is OK run the following commands and open your VM's IP address on an web browser.

```
sudo systemctl status nginx
```

The command should show the "Active:" property with the "active (running)" value, and on the web browser you should see a web page (the web page might be from the Apache project but this is only because that's the index.html file that's present on the default directory).

6 - Deploying Code

Before deploying the actual code we are going to add configuration options to our Vue application. This is needed because the API domain and Web Socket server connections are going to be different.

6.1 - Vue Config

Create two files: .env and .env.production with the following contents.

```
# .env
VUE_APP_API_DOMAIN=<YOUR API DOMAIN>
VUE_APP_WS_CONNECTION=<YOUR WS CONNECTION>
```

```
# .env.prodution
NODE_ENV=production
VUE_APP_API_DOMAIN=http://<ip_of_vm>
VUE_APP_WS_CONNECTION=http://<ip_of_vm>
```

Next change your main.js file to add the code bellow.

```
...
const apiDomain = process.env.VUE_APP_API_DOMAIN
const wsConnection = process.env.VUE_APP_WS_CONNECTION
...
const socketIO = new VueSocketIO({
    ...
    connection: wsConnection,
    ...
```

```
})
...
axios.defaults.baseURL = `${apiDomain}/api`
app.config.globalProperties.$serverUrl = apiDomain
...
```

Now let's change the package.json

```
...
"scripts":{
    ...
    "build": "vue-cli-service build --mode production",
    ...
}
...
```

This should allow us to build the Vue app and apply the production configuration. Please keep in mind that if you want more variables for some other scenario they need to be prefixed with VUE_APP

6.2 - Initial Deploy

Now that every dependency is install let's deploy our code. This has two stages, the first deploy, where we configure everything, and subsequent code updates in which we are only concerned with getting new code to run.

In this part of the tutorial we assume that locally you have the same folder structure that we created in the Pre Tasks section of this document on your local development environment, if not please adjust accordingly.

The first step is to get our code onto to the VM, and the simplest way to do this is transfer it over. We will be using the rsync command (Instructions for Windows users to use rsync in a latter section).

Since, at least, the Vue application will be deployed differently, we will do then one by one.

Before we start copying code we will configure the permissions on our folders. Run these commands on the server.

```
sudo usermod -a -G www-data dad
sudo chown -R www-data:www-data /var/www/project
sudo chmod -R g+rw /var/www/project
```

To have the changes (the group) applied you need to log off and log back on to the server.

6.2.1 Laravel Application

We want to copy the files over without the vendor folder and the /public/storage symbolic link. Run the following code on the root of your project (one folder behind the laravel folder).

```
# with ssh-add
rsync -av --exclude "vendor" --exclude "public/storage" --exclude "composer.lock" laravel
dad@<vm_ip_address>:/var/www/project/
# without ssh-add
rsync -av --exclude "vendor" --exclude "public/storage" --exclude "composer.lock" -i <path_to_your_ssh_key>
laravel dad@<vm_ip_address>:/var/www/project/
```

On the server we need to include the dependencies, create the database and seed the data. Check the values of the local .env (database and root with be dad) and run these commands on the laravel folder in the server.

```
composer install
php artisan migrate
```

```
php artisan db:seed
php artisan passport:install
php artisan storate:link
```

Copy the client secret to your .env file, and change the passport url to http://localhost.

6.2.2 Vue App

Locally (vue app folder) run these commands to build and push the build application to the server.

```
npm run build
# with ssh-add
rsync -av dist/ dad@<vm_ip_address>:/var/www/project/vue/
# without ssh-add
rsync -av -i <path_to_your_ssh_key> dist/ dad@<vm_ip_address>:/var/www/project/vue/
```

In this case this is enough as the Nginx server will be responsible for serving the files.

6.2.3 Web Sockets App

On the project folder locally run.

```
# with ssh-add
rsync -av wesocket dad@<vm_ip_address>:/var/www/project/
# without ssh-add
rsync -av -i <path_to_your_ssh_key> wesocket dad@<vm_ip_address>:/var/www/project/
```

On the server in the websocket folder.

```
npm install
pm2 start index.js --watch
```

6.3 Nginx Configuration

All that's left is to configure Nginx to handle the apps. We will be using the reverse proxy feature of Nginx to handle requests to these paths.

```
/api -> Laravel App
/storage -> Laravel App
/oauth -> Laravel App
/socket.io -> WebSocket Server
```

By default PHP FPM (which is what we are using as a PHP "server") runs as an UNIX socket and we set the Web Socket server to run on 8080, so our config file for Nginx is going to be the following.

```
upstream websocket {
    server localhost:8081;
    keepalive 64;
}

server {
    listen 80;
    server_name <SERVER_IP> default;
    root /var/www/project/laravel/public;
```

```
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location ~ ^/(api|storage|oauth)/.* {
        root /var/www/project/laravel/public;
        try_files $uri /index.php?$is_args$args;
    }
    location ~ ^/socket.io/.* {
        proxy_pass http://websocket;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt  { access_log off; log_not_found off; }

    location / {
        root /var/www/project/vue;
        index  index.html;
        try_files $uri $uri/ /index.html;
    }

    error_page 404 /index.php;

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

Create a file on /etc/nginx/sites-available named dad with the contents above.

```
sudo vim /etc/nginx/sites-available/dad
```

Setup, check and restart Nginx

```
sudo rm /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/dad /etc/nginx/sites-enabled/dad
sudo nginx -t
sudo systemctl restart nginx
```

6.4 Final Changes

Each time we push code to the server permissions might make then inaccessible to Nginx, so after each push we need to run the following.

```
sudo chown -R www-data:www-data /var/www/project
sudo chmod -R g+rw /var/www/project
```

Test the project by opening the IP address of the VM in a browser

6.5 Subsequent Deploys

After you change code on your local environment to push to production you will need to run the appropriate rsync.

```
# Laravel: on the project folder
rsync -av --exclude "vendor" --exclude "public/storage" --exclude "composer.lock"  laravel

# Vue: on the vue folder
rsync -av dist/ dad@<vm_ip_address>:/var/www/project/vue/

# WebSockets: on the project folder
rsync -av wesocket dad@<vm_ip_address>:/var/www/project/
```

And go to the server and reset the permissions (this step could be avoided but it would increase the complexity of the configurations)

```
sudo chown -R www-data:www-data /var/www/project
sudo chmod -R g+rw /var/www/project
```

And all should be running.

7 - Tools and Troubleshooting

7.1 - rsync on Windows

The simplest way to get rsync on Windows is to use the Chocolatey package manager. To install it check out these install instructions.

After the installation all you have to do is to run this command on an administrative privilege Powershell.

```
choco install rsync
```

7.2 Check OpenSSH

7.2 Useful commands on the VM

```
# Service Status
sudo systemctl status nginx
sudo systemctl status php7.4-fpm
pm2 status

# Service Restarts
sudo systemctl restart nginx
sudo systemctl restart php7.4-fpm
pm2 restart <id>  # you can get the id from the status but it should be 0
```

```
# Logs
tail -f /var/log/nginx/error.log
sudo journalctl -u nginx
sudo journalctl -u php7.4-fpm
pm2 log <id> # you can get the id from the status but it should be 0

# Permissions Reset
sudo chown -R www-data:www-data /var/www/project
sudo chmod -R g+rw /var/www/project

# MySQL
mysql -u root -p
```