## **Steps to Setup in local system**

Step 1: cd LnS\_Assignment

Step 2: Activate virtual environment:

venv/Scripts/activate

Step 3: Again cd LnS\_Assignment

Step 4: Install dependencies

pip install -r requirements/local.txt

Step 5: Database setup using comand below:

python manage.py makemigrations

python manage.py migrate

Step 6: Create super user to login into admin dashboard

python manage.py createsuperuser

Step 7: Start application using command below:

python manage.py runserver

Copy base url: [http://localhost:8000](http://127.0.0.0:8000) with end points call api (required api key for authorization)

**Djnago Rest Api Key**

To get api key, open [http://localhost:8000/admin](http://127.0.0.0:8000/admin) in browser and login using superuser credentials to add an "API Key Permissions" section to the Django admin site where you can create, view and revoke API keys.

then clients must make authorized requests using:

X-Api-Key: \*\*\*\*\*\*\*\*

where \*\*\*\*\*\*\*\* refers to the generated API key

Api-Key Example: WGlerakY.ZE8tM2vJeDEhng4YY3eki4wujkIAG4kM.

#### **Upload Sample Data**

python manage.py loaddata --format=json contact

## 

## 

## **Test the API**

Route: http://localhost:8000/signup/

Request Type: POST

Data:

{

"name":"Amit Kumar",

"phone":"6200002221",

"email":"qewq@gmail.com",

"password":"qwertyuiop",

}

Route: http://localhost:8000/login/

Request Type: POST

Data:

{

"phone":"77778787887",

"password":"qwertyuiop"

}

#### **JWT AUTH**

You will receive a JWT Auth Token after signIn. Include the JWT token in headers as a Bearer Token for all private requests

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhbmt1ciJ9.eQicFQy2nYric9Gl2mhqOH4l8An7B\_Kf2CKoJmZrPcA

#### **To view all the contacts**

Public Route: http://localhost:8000/contact/

Request Type: GET

#### **To mark a contact as SPAM**

Private Route: http://localhost:8000/spam/

Request Type: POST

Data:

{

"phone": "9988689898"

}

#### **To search a contact by name**

Private Route: http://localhost:8000/search\_by\_name?name=Kumar

Request Type: GET

#### **To search a contact by phone**

Private Route: http://localhost:8000/search\_by\_phone?phone=9000002222

Request Type: GET

## **Project structure and deployment on Cloud**

### **Clone the template repository**

I prepared a template repository of a Django project, you can clone it at the following address before proceeding:

git clone https://github.com/baxeico/django\_ansible.git

In the repository you’ll find three directories:

#### **LnS\_Assignment**

This directory is a Django project created with the usual:

django-admin.py startproject LnS\_Assignment

So inside *LnS\_Assignment* sub-directory you’ll find a *settings* package containing different modules for different deploy environments:

* *base.py* – This is the common settings which will be inherited (and possibly overridden) from all deploy environments;
* *local.py* – This will be the *DJANGO\_SETTINGS\_MODULE* used in your local development environment. Nothing special here, it only makes Django to use sqlite as DB for local development;
* *production.py* – This will be the *DJANGO\_SETTINGS\_MODULE* used in production. Here you’ll find that the database used by Django is defined using [dj-database-url](https://github.com/kennethreitz/dj-database-url) package, by means of an environment variable. Also the *STATIC\_ROOT* settings variable is defined by means of an environment variable.

#### **requirements**

This directory contains the minimum Python requirements of your project, you’ll find three files there:

* *base.txt* – contains the basic requirements for all deploy environment (only Django in the example, it’s a Django project after all, isn’t it?);
* *local.txt* – it inherits from *base.txt* and adds Ansible as a requirement for your local development environment. Version 2.3.0 is the latest Ansible version at the time of writing;
* *production.txt* – it also inherits from *base.txt* and adds [dj-database-url](https://github.com/kennethreitz/dj-database-url) as a requirement for your production environment. It will be used to configure the database used by Django by means of the environment variable *DATABASE\_URL*.

You can install the local requirements using this command:

pip install -r requirements/local.txt

Use [Virtualenv](https://virtualenv.pypa.io/en/stable/), together with [virtualenvwrapper](https://virtualenvwrapper.readthedocs.io/en/latest/) to create and manage an isolated environment for your project.

#### **ansible**

In this directory you’ll find the set of Ansible playbooks to automate the installation and configuration of the server and the deployment of your project.

The first thing to customize is the hosts file. This is the file where you could list all the hosts controlled by Ansible.

Then you can proceed to rename the file *host\_vars/yourserver*, using the label you gave to your server in the *hosts* file. Here you’ll also find some variables used in the playbooks.

Here is a brief description of each playbook you’ll find:

* *config\_files.yaml* – Copy nginx and uwsgi configuration files on remote server.
* *deploy.yaml* – Deploy your Django project on the server, pulling the master branch from your GIT repository, installing all needed production requirements, running migrate and collectstatic and restarting uwsgi.
* *packages.yaml* – Install needed software packages on remote server using *apt*.
* *postgresql.yaml* – Create and configure the access to a [Postgresql](https://www.postgresql.org/) database used by your Django project on remote server. Notice how the database password is randomly generated and stored in the local file *postgresql*. The password will be automatically inserted in the *DATABASE\_URL* environment variable, to be used in Django production settings.
* *system.yaml* – Create an user *ubuntu* on remote server, together with a private/public SSH key pair. The public key is returned as output when you run the playbook, to be used as a “deploy key” on the server. More details later on this step.
* *upgrade.yaml* – Upgrade all *apt* packages on remote server.

### **Ansible automation begins!**

Inside the *ansible* directory, run the following command:

./ansible.sh system.yaml

You should see an output similar to this:

ok: [yourserver] => {

"changed": false,

"msg": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC1kvkW9... ansible-generated on yourserver.example.com\n"

}

What you find in the *msg* variable is the public SSH key generated for the *ubuntu* user on remote server. You should copy the public key and add it as a “deploy key” in the settings of your GIT repository.

A deploy key is a read-only SSH key that will be used to clone your repository from the remote server. You can find more details in the [Bitbucket](https://confluence.atlassian.com/bitbucket/use-access-keys-294486051.html) and [Github](https://developer.github.com/guides/managing-deploy-keys/#deploy-keys) documentation.

Now you are ready to complete the deploy of your Django project! Run the following commands inside the *ansible* directory:

./ansible.sh packages.yaml

./ansible.sh postgresql.yaml

./ansible.sh config\_files.yaml

./ansible.sh deploy.yaml

If all goes well you should be able to reach your Django project on your remote server public address, on port 80.

**App**

In this directory you’ll find the sub-directories fixtures, migrations, tests and Files admin.py, apps.py, models.py, serializers.py, urls.py, views.py.

