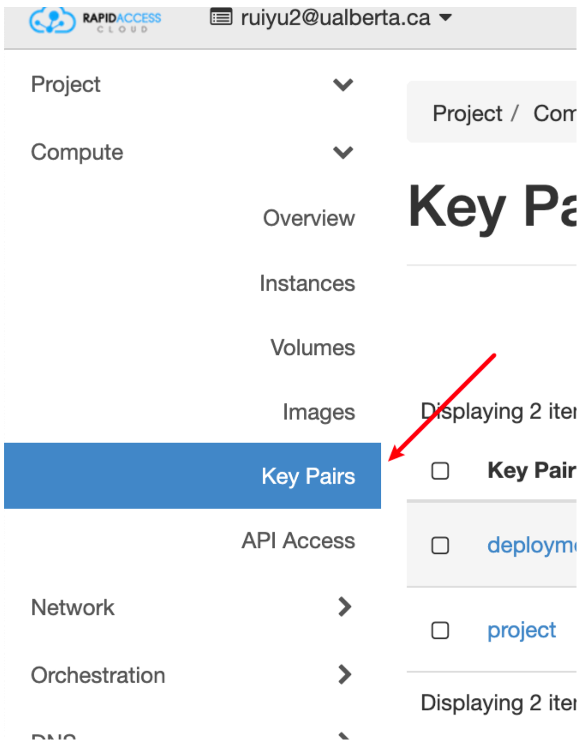
Cybera setting

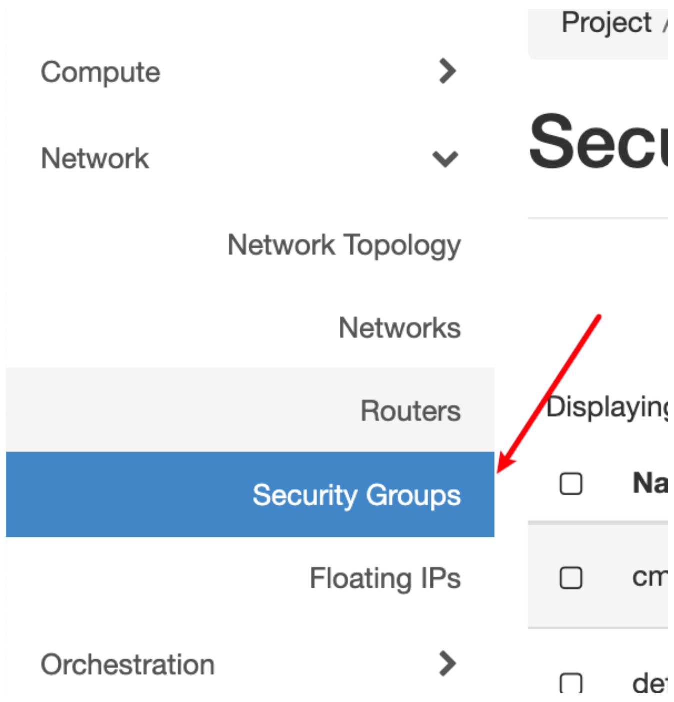
1. Create a key pair

First login to the Cybera account. Select the key pairs in the compute section in the sidebar. Then click Create a key pair and save it as deployment.pem.

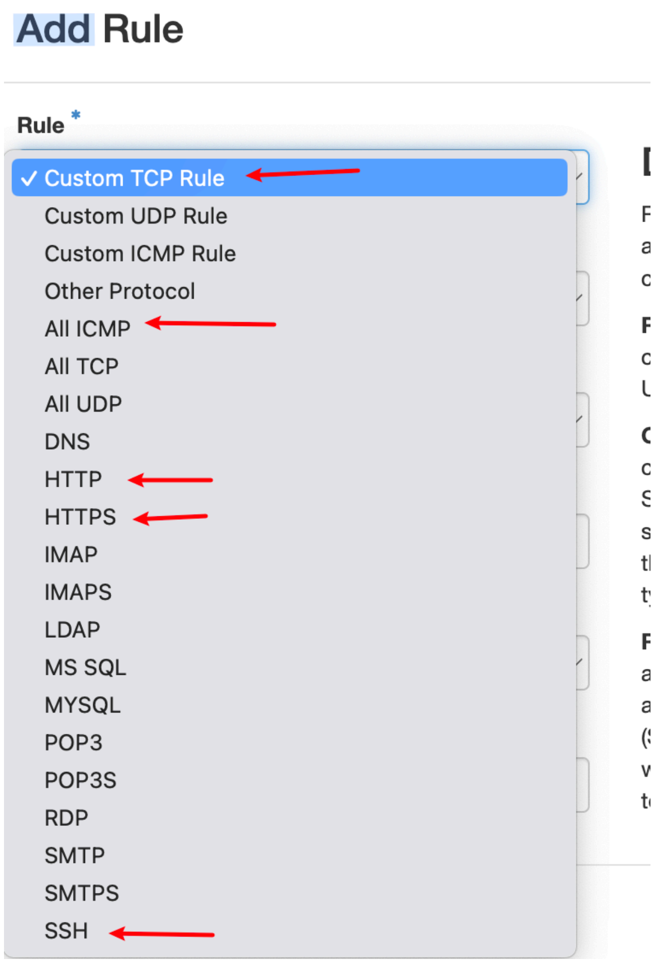


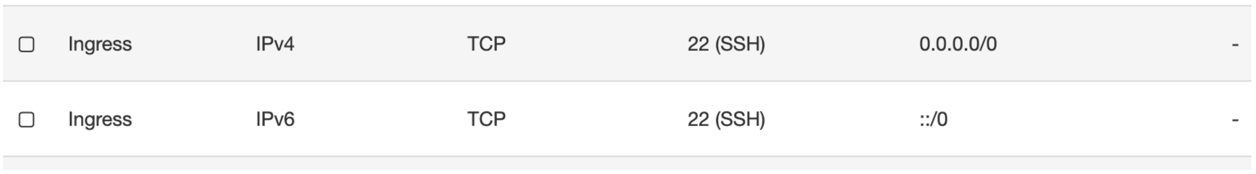
2. Set up the Security Group

Click the Security Groups in the Network section in the sidebar.



Create a new Security Group and add ICMP, HTTPS, HTTP and SSH into the group. Also all port 8000 for the custom TCP rules. For each of the five rules, add both ipv4 and ipv6 by setting CIDR to 0.0.0.0/0 and ::/0.

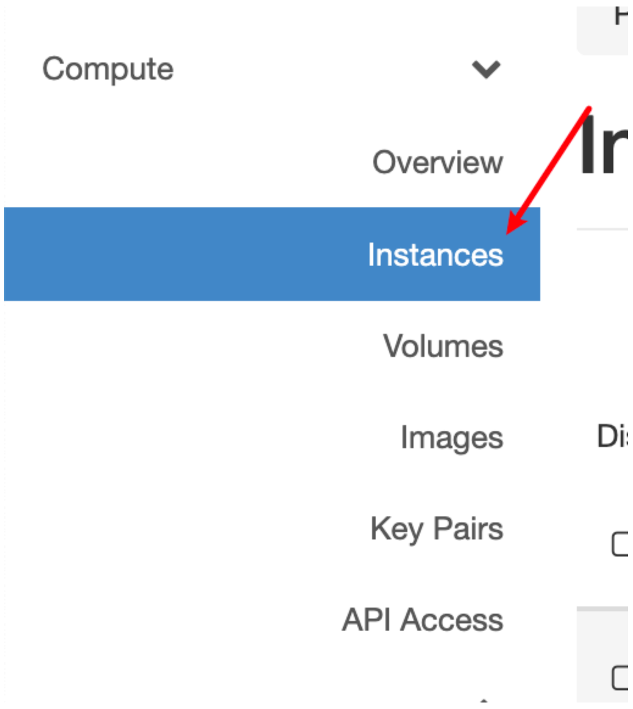


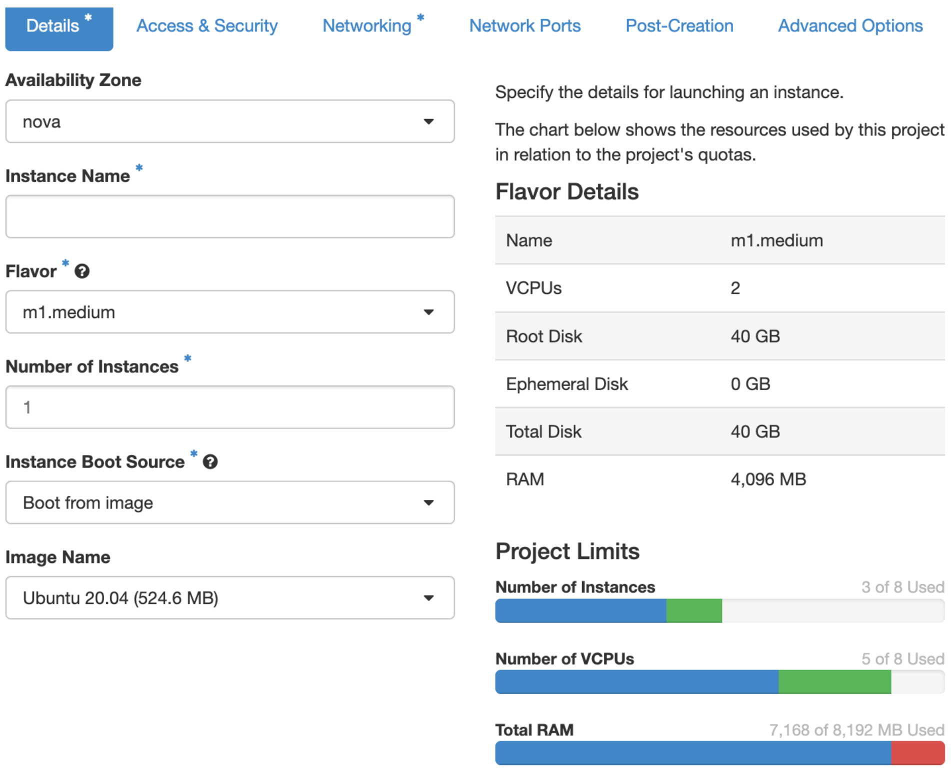


▢	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-
▢	Ingress	IPv6	TCP	22 (SSH)	::/0	-

3. Create a new instance

Go to the instances in the Compute section of the sidebar. launch an instance with flair equals m1.median and boot from the image with Ubuntu 20.04.





Availability Zone: nova

Instance Name *:

Flavor *: m1.medium

Number of Instances *: 1

Instance Boot Source *: Boot from image

Image Name: Ubuntu 20.04 (524.6 MB)

Specify the details for launching an instance.

The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.medium
VCPUs	2
Root Disk	40 GB
Ephemeral Disk	0 GB
Total Disk	40 GB
RAM	4,096 MB

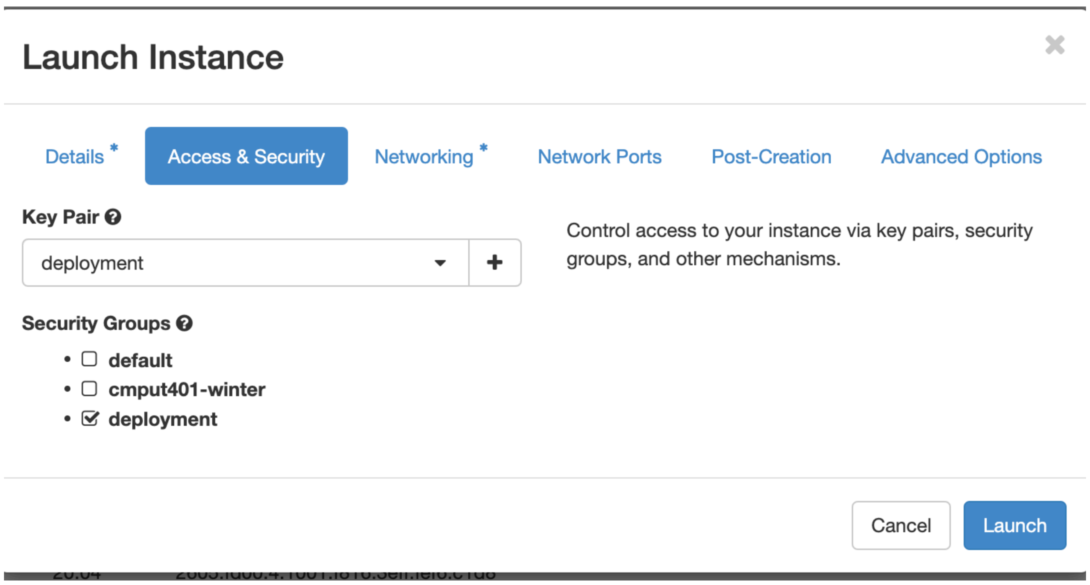
Project Limits

Number of Instances: 3 of 8 Used

Number of VCPUs: 5 of 8 Used

Total RAM: 7,168 of 8,192 MB Used

The check the Access & Security setting and select both key pair and security groups we made before.



Launch Instance

Details * | **Access & Security** | Networking * | Network Ports | Post-Creation | Advanced Options

Key Pair

deployment

Control access to your instance via key pairs, security groups, and other mechanisms.

Security Groups

- ☐ default
- ☐ cmput401-winter
- ☑ deployment

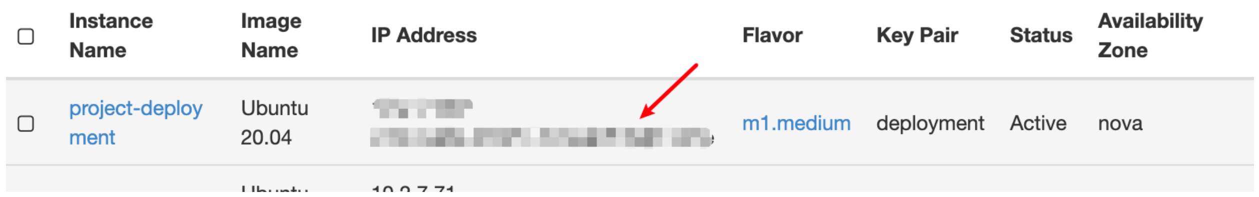
Cancel | Launch

4. Connect to the machine

To connect to the instance, go to the terminal and input the following command:

```
ssh -i deployment.pem ubuntu@<instance_addr>
```

The <instance_addr> should be replaced with the actual ipv6 address. It can be found in the instance page.



☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone
☐	project-deployment	Ubuntu 20.04	[redacted]	m1.medium	deployment	Active	nova

Backend deployment

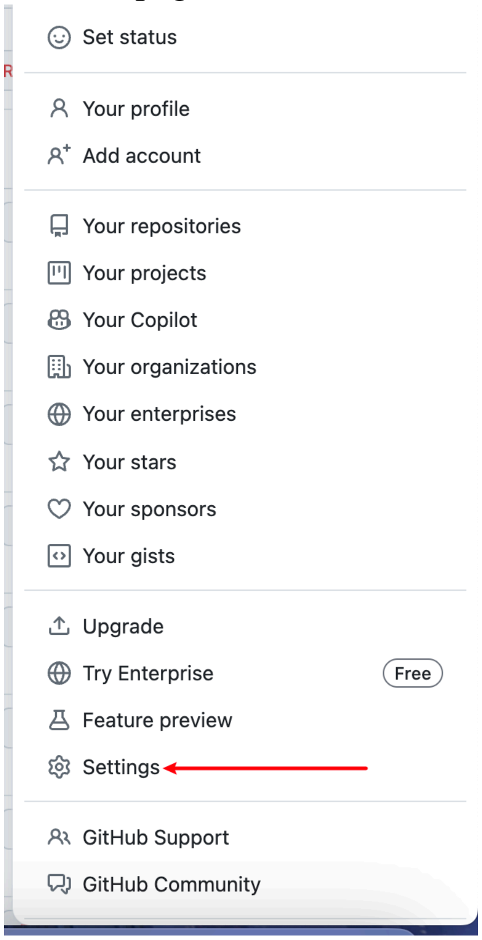
1. Clone project

After logging into the instance, we can start cloning the project to the instance from GitHub. It can be done by running the following command:

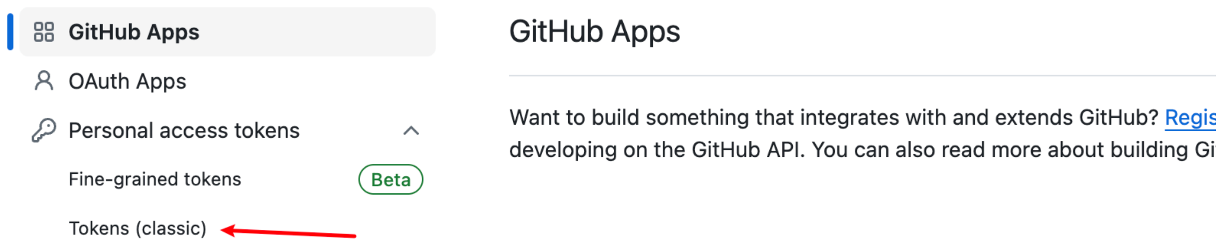
```
git clone https://github.com/UAlberta-CMPUT401/w24project-sic_desk_management.git
```

It will then ask for the username and password. The username is the actual username of the GitHub account. The password is the token. It can be acquired from GitHub by following the steps.

First, go to the settings of the GitHub page.

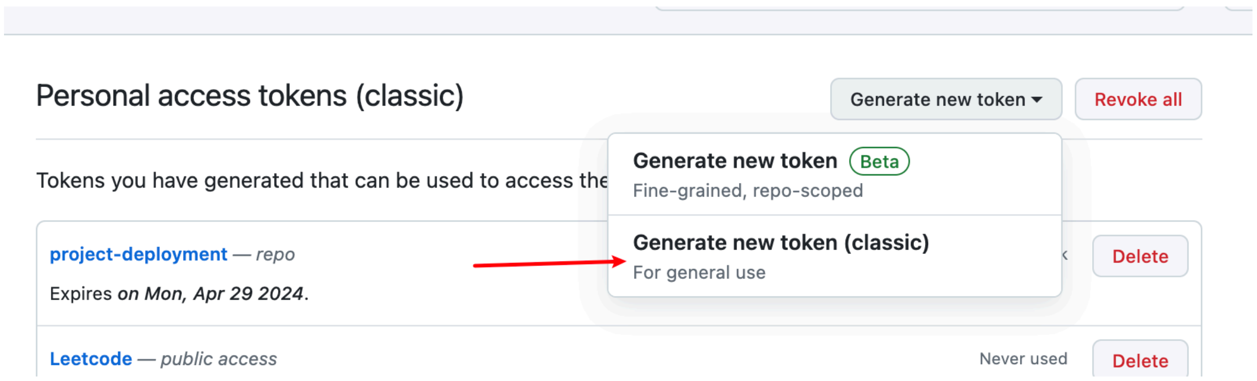


Then go to the bottom of the setting options, click “developer settings” and select “Tokens (classic)”



Want to build something that integrates with and extends GitHub? Regis
developing on the GitHub API. You can also read more about building Gi

Select Generate new token (classic).



Personal access tokens (classic)

Tokens you generated that can be used to access the

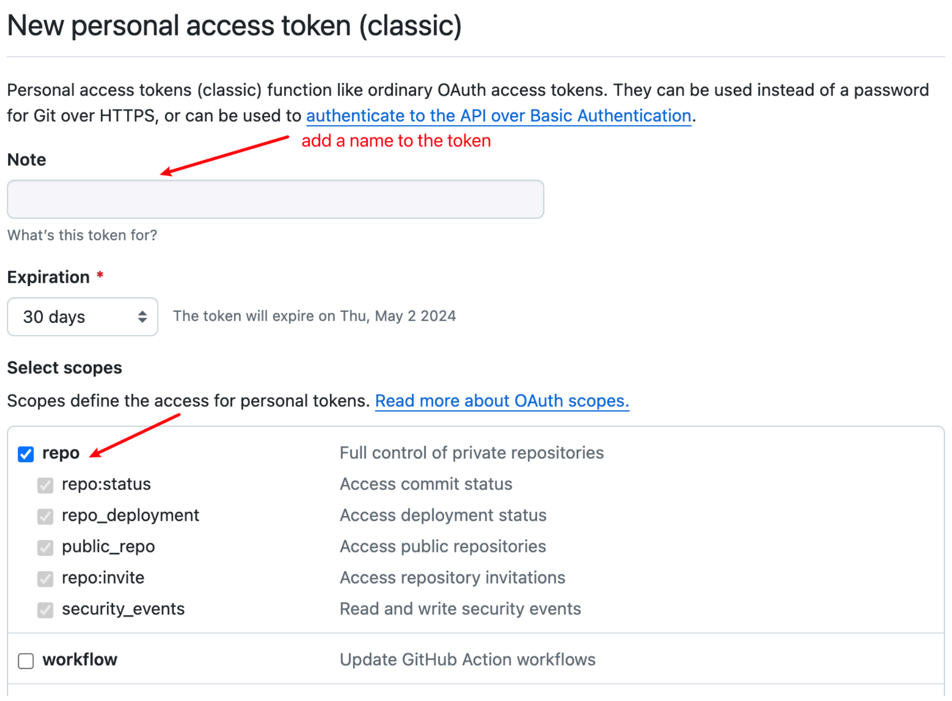
project-deployment — *repo*

Expires *on Mon, Apr 29 2024*.

Leetcode — *public access*

Never used

Give the token a name, check all boxes for repo access and click generate token at the bottom.



New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

add a name to the token

Note

What's this token for?

Expiration *

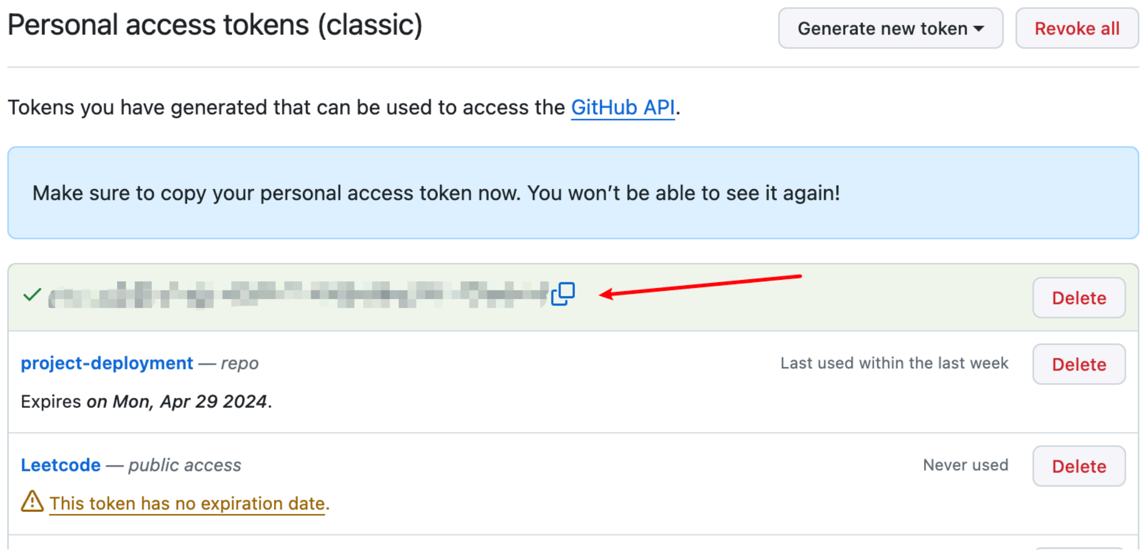
30 days — The token will expire on Thu, May 2 2024

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events
☐ workflow	Update GitHub Action workflows

The token will show up here. It is recommended to copy the token and store it in a file.



Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

✓ [redacted]

project-deployment — *repo*

Expires *on Mon, Apr 29 2024*.

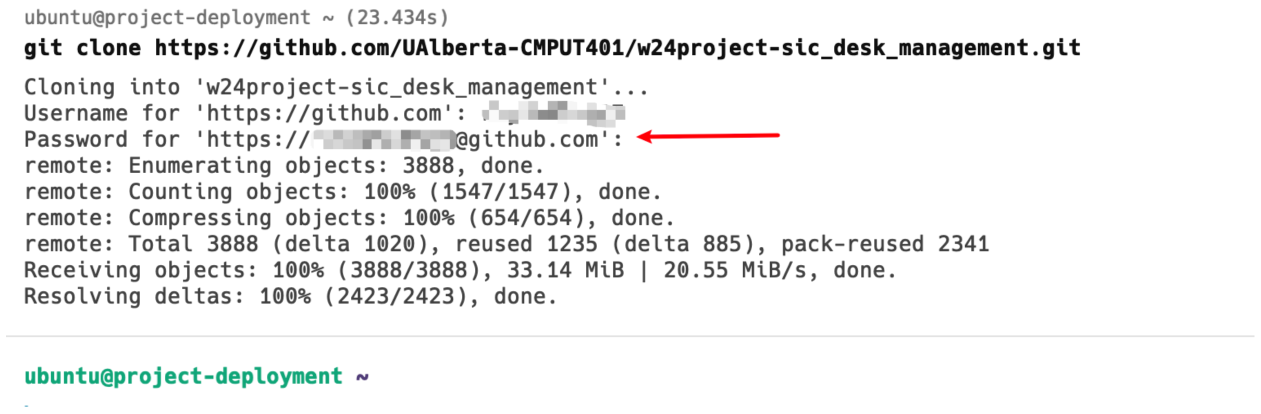
Last used within the last week

Leetcode — *public access*

This token has no expiration date.

Never used

Put this token into the password position and then we can clone the project from github.



```
ubuntu@project-deployment ~ (23.434s)
git clone https://github.com/UAlberta-CMPUT401/w24project-sic_desk_management.git
Cloning into 'w24project-sic_desk_management'...
Username for 'https://github.com': [redacted]
Password for 'https://[redacted]@github.com':
remote: Enumerating objects: 3888, done.
remote: Counting objects: 100% (1547/1547), done.
remote: Compressing objects: 100% (654/654), done.
remote: Total 3888 (delta 1020), reused 1235 (delta 885), pack-reused 2341
Receiving objects: 100% (3888/3888), 33.14 MiB | 20.55 MiB/s, done.
Resolving deltas: 100% (2423/2423), done.

ubuntu@project-deployment ~
```

We need to switch to the deployment branch. It can be done with the following command:

```
git branch -a
git checkout -b deployment origin/deployment
```

Also, we need to put the credentials file into the root of our project

```
cd /w24project-sic_desk_management/
```

Place the service_credentials.json in the root folder, which can be found here.

2. Install dependencies

1). Check python3 version

First, check the Python version with the following command:

```
python3.10 --version
```

Our project works on python3.10. If the version is not installed, we need to install python3.10 first.
To install run the following commands. The method is to get from here.

```
sudo apt update
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.10
```

Then check the python3.10 version, if it exists then we are good.

2). Install python dependency

First, create a python virtual environment of python3.10. The library can be installed with following command:

```
sudo apt-get install python3.10-dev
sudo apt install python3-pip
sudo apt install python3.10-venv
```

Then, create a pythton3.10 virtual environment. (It is recommended to create the virtual environment outside project folder w24project-sic_desk_management/ so that we don't need to update the configuration file later)

```
python3.10 -m venv venv
```

Enter the virtual environment with the command:

```
source venv/bin/activate
```

We can exit the virtual environment with the command:

```
deactivate
```

After entering the virtual environment, we need to get into the project folder. It can be done by following the command:

```
cd w24project-sic_desk_management/
```

Check the files in the folder folder with the command “ls” and there should be a requirement.txt file in the list. Then run the following command to install the Python dependencies.

```
pip install -r requirements.txt
```

3. Deploy backend

1). Install nginx and uwsgi

Install nginx with the command:

```
sudo apt install nginx
```

Install uwsgi with the command:

```
pip install wheel
pip install uwsgi
```

2). Config nginx

Open the nginx configuration file with the command:

```
sudo vim /etc/nginx/sites-enabled/default
```

Then we need to comment out all existing code and put the following settings in the file. We can use “#” to comment out the line and use “:wq” to save and quit from vim.

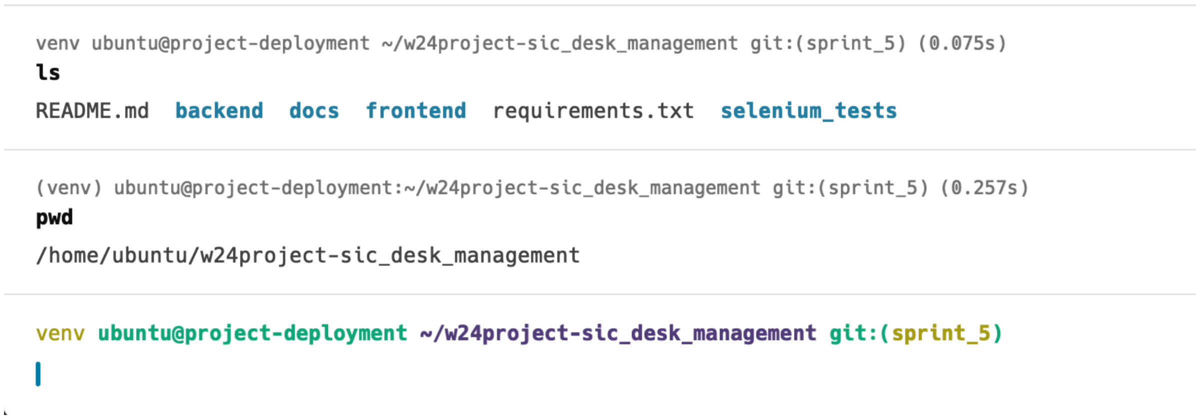
```
upstream django {
      server 127.0.0.1:8001;  # must be same as what set in uwsgi socket
}
server {
      listen       80; # ipv4 listening port number
      listen          [::]:80;    # ipv6 listening port number
      server_name  test;
      charset      utf-8;
      location /static {
          autoindex on;
          alias /home/ubuntu/w24project-sic_desk_management/backend/static;
      }

       location / {
       root /home/ubuntu/w24project-sic_desk_management/frontend/dist;
       index index.html;
       try_files $uri $uri/ /index.html;

       }

      location ~ /api/* {
             uwsgi_pass 127.0.0.1:8001;
             include
/home/ubuntu/w24project-sic_desk_management/backend/uwsgi_params;
      }
}
```

In the previous configuration file, there are two directories. W24project-sic_desk_management is the project directory and we need to make sure it is correct. To get the directory path, cd into the project directory and type "pwd".



```
venv ubuntu@project-deployment ~/w24project-sic_desk_management git:(sprint_5) (0.075s)
ls
README.md  backend  docs  frontend  requirements.txt  selenium_tests

(venv) ubuntu@project-deployment:~/w24project-sic_desk_management git:(sprint_5) (0.257s)
pwd
/home/ubuntu/w24project-sic_desk_management

venv ubuntu@project-deployment ~/w24project-sic_desk_management git:(sprint_5)
```

The /api/ is for the backend. uwsgi_params and the static folder will be created with the following instructions. The default / is for the front end. The dist folder will be created in the frontend deployment instructions below.

3). Config uwsgi

Go to the directory "/w24project-sic_desk_management/backend" and vim into the uwsgi_config.ini.

```
vim uwsgi_config.ini
```

Then update the field "virtualenv" with the actual virtual environment path. If the virtual environment is created outside the w24project-sic_desk_management/

folder, the default configuration should be fine.

```
[uwsgi]
socket = 127.0.0.1:8001
chdir = /home/ubuntu/w24project-sic_desk_management/backend
wsgi-file = backend/wsgi.py
processes = 4
threads = 2
vacuum = true
buffer-size = 65536
virtualenv = /home/ubuntu/venv
```

4). Add uwsgi_params file

Still under the directory “/w24project-sic_desk_management/backend”, vim into uwsgi_params file.

```
vim uwsgi_params
```

Add the following code into it.

```
wsgi_param QUERY_STRING $query_string;
uwsgi_param REQUEST_METHOD $request_method;
uwsgi_param CONTENT_TYPE $content_type;
uwsgi_param CONTENT_LENGTH $content_length;
uwsgi_param REQUEST_URI $request_uri;
uwsgi_param PATH_INFO $document_uri;
uwsgi_param DOCUMENT_ROOT $document_root;
uwsgi_param SERVER_PROTOCOL $server_protocol;
uwsgi_param HTTPS $https if_not_empty;
uwsgi_param REMOTE_ADDR $remote_addr;
uwsgi_param REMOTE_PORT $remote_port;
uwsgi_param SERVER_PORT $server_port;
uwsgi_param SERVER_NAME $server_name;
```

4). Export statics

Go to the directory “/w24project-sic_desk_management/backend” and run the following command:

```
python manage.py collectstatic
```

5). Run uwsgi

Still in the directory “/w24project-sic_desk_management/backend” run the command:

```
uwsgi --ini uwsgi_config.ini
```

Then, go to the browser and type in the address of the instance in the format “http://[instance_addr]/admin”. If we can see the admin page, then the deployment is successful.
We can add -d to previous command to let it run backend.

```
uwsgi -d --ini uwsgi_config.ini
```

Each time we make updates to the files, we can run the following command to stop the uwsgi service and redo the previous command to restart the service.

```
sudo killall -9 uwsgi
```

Frontend deployment

1. Install the frontend dependencies

First, go into “/w24project-sic_desk_management/frontend”. Then check the version of npm with the command:

```
npm -v
```

Then update the npm with the command:

```
sudo npm install -g npm@latest
```

Check the nvm with the following command:

```
nvm -v
```

If it does not find nvm, run the following command to source the nvm script.

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

Update nvm with the command:

```
nvm install --lts
```

Check the version of the node.

```
node -v
```

Finally, install dependencies and build the project. It will generate a dist folder with static files of frontend

```
npm install
npm run build
```

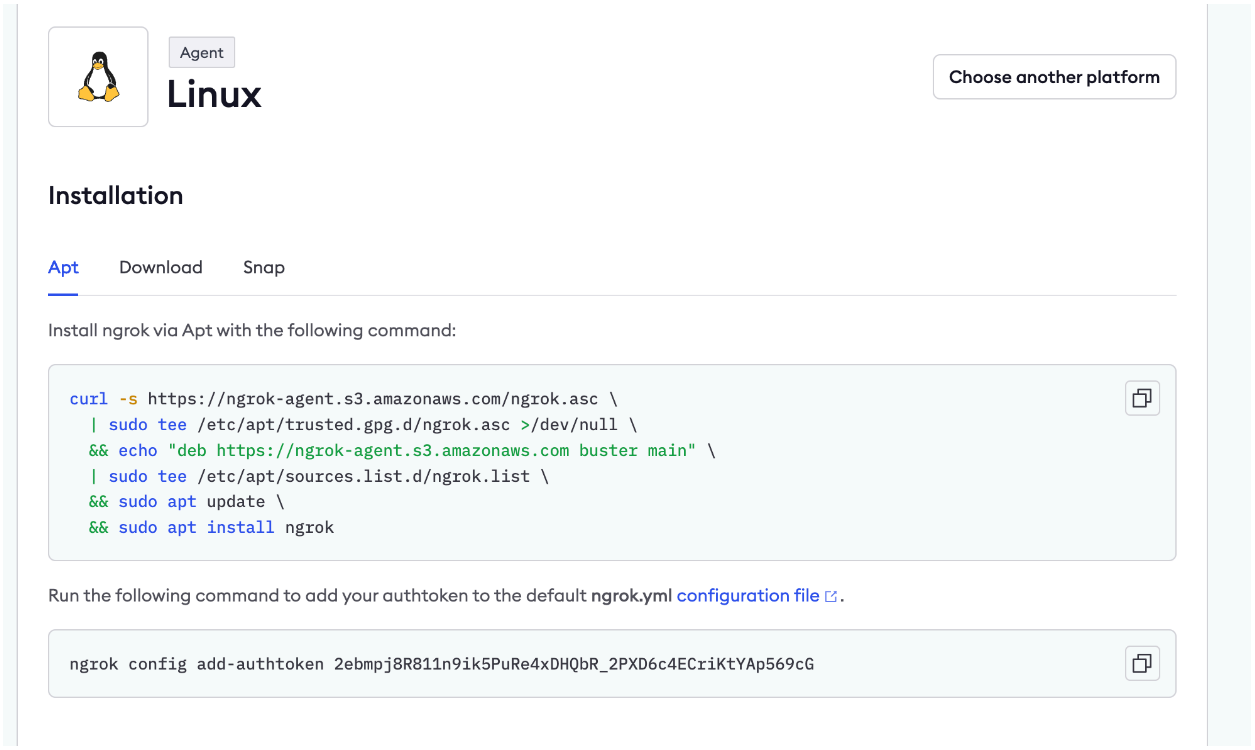
2. Nginx configuration

Go into the Nginx configuration file above and check the address of the root folder. It should be set to the dist folder.

```
sudo vim /etc/nginx/sites-enabled/default
```

Ngrok setting

Ngrok can be used to get a free domain name for the app. The link it here. After logging in, follow the instructions for Linux to install the Ngrok on the instance.



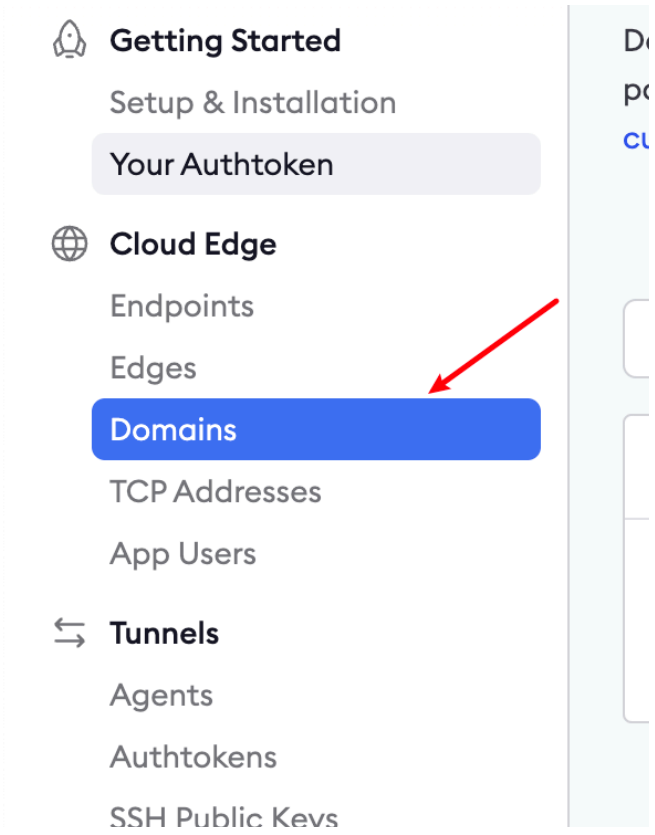
Install ngrok via Apt with the following command:

```
curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
  | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
  && echo "deb https://ngrok-agent.s3.amazonaws.com buster main" \
  | sudo tee /etc/apt/sources.list.d/ngrok.list \
  && sudo apt update \
  && sudo apt install ngrok
```

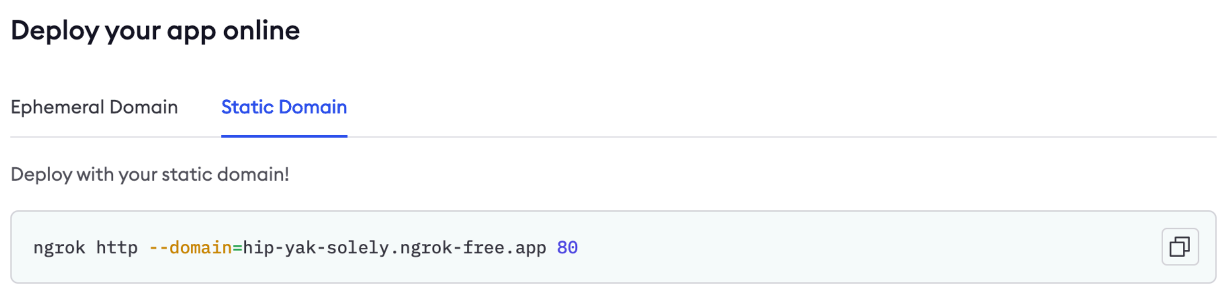
Run the following command to add your authtoken to the default **ngrok.yml** configuration file.

```
ngrok config add-authtoken 2ebmpj8R811n9ik5PuRe4xDHQbR_2PXD6c4ECriKtYAp569cG
```

Then we can apply for a static domain by choosing domains and creating a new domain.



Then run the command to enable the Ngrok service.



```
ngrok http --domain=hip-yak-solely.ngrok-free.app 80
```

The service is running live. To keep it running after closing the terminal, we can use tmux. Do the following command to create a new session.

```
tmux new-session -s {{session_name}}
```

Then run the previous command to run Ngrok. We can type "control-b d" to detach from the tmux. To go back to the session, use the following command.

```
tmux ls
tmux attach-session -t {{session_name}}
```