* **A simple user guide** for testing (including system deployment and end user invocation/usage of the systems);

The structure of the directories

Under the project directory, there are four folders.

./Deploy: contains all the files and scripts related to deployment including both Boto and Ansible files.

./TweetsAnalysis: This folder contains the sentiment analyser which is written in Python.

./TweetsReader: This folder contains there different kinds of tweet harvesters including stream harvester, search harvester and keywords harvester. The harvesters are also written in python.

./WebApplication: This folder contains the web application which is using node.js to visualize analysed data to the end user.

Deployment

Prerequisites

To deploy the system, the user should follow the instructions on the Nectar website to sign up a user account. After that the user should upload a public key to the Nectar under the Access and Security tab, which will be set as the default key of the instances for SSH access. At the end, user should download the EC2 credentials from Nectar to configure Boto.

All the files related to this sections are under the Deploy folder.

Step 1: Establish instances using Boto

Firstly, the user should set up the configuration for Boto in the BotoConfig.txt. There are some key value pairs in the file.

|  |  |
| --- | --- |
| Name | Description |
| Access\_key\_id | The access key id to access Nectar which could be found in the EC2 credentials. |
| Secret\_access\_key | The secret access key to access Nectar which could be found in the EC2 credentials. |
| dbNunber | The number of harvester nodes. |
| dbKey | The name of the public key to build VM for harvesters. |
| serverKey | The name of the public key to build VM for web application. |
| dbSecurityGroups | The security groups for harvester nodes. |
| serverSecurityGroups | The security groups for the web application. |

After set up those values, run the command

Python DeployCloud.py

The instances should be set up. And a file containing all the IP addresses of the instance should be printed out in the file ansibleinventory.yaml which will be used in the next section.

Step2: Configure instances with Ansible

The first thing to do is configuring the Ansible inventory. The user should look into the ansibleinventory.yaml file, which contains the IP addresses of all the instances. This file will be used as the inventory file for Ansible. The user should add more groups in the inventory file to specify the type of harvesters in the node.

Group stream: the instances in this group will be running a harvester which is using Tweet stream API.

Group search: the instances in this group will be running a harvester which is using Tweet search API.

Group searchTopic: the instance in this group will be running a harvester which is using Tweet search API to search for tweets against a specific topic.

Group sever: the instance in this group will be running the task of analysing tweets.

Execute Ansible

When the inventory file is correct and satisfy the user’s need, the user could run the command:

*ansible-playbook InstallCouchdb.yml -i ansibleinventory.yaml --private-key YOUR\_PRIVATE\_KEY*

By executing the InstallCouchdb.yml, CouchDB and other dependencies or packages such as git will be installed on the instances with in the inventory group “dbs” and “server”. And also some necessary configurations of the CouchDB shoud be done to be ready for clustering. When the user SSH to one of the remote instance, and execute the command:

*curl* [*http://127.0.0.1:5984*](http://127.0.0.1:5984)

The welcome message of the CouchDB should be returned indicating that the CouchDB is installed correctly.

The next step is to form a cluster for CouchDB. Run the command:

*ansible-playbook FormCluster.yml -i ansibleinventory.yaml --private-key YOUR\_PRIVATE\_KEY*

In the FormCluster.yml, a cluster containing all the instance in the inventory group “dbs” and “server” will be builed.

It is also important to mention that in the file FormCluster.yml, a reverse proxy server will be set up to redirect requests from port 80 which are html requests to the port 3000 which is the port node js running on. By doing so, the user will be able to access the web application just using the IP address without the port number.

When the user SSH into the remote instances and create a database on one node, the database should appear in any other nodes as well, which indicates that the cluster is formed correctly.

Step 3: Execute jobs with Ansible

In the Deploy folder, there are four Ansible playbooks which are for publishing the harvesters and analyser to remote. In each of the playbooks, the structures are generally following the steps as shown bellow.

1. Stop the previous task
2. Remove the previous code
3. Make sure dependencies exist
4. Get source code from git
5. Run the code

The main advantages of having the first two steps are those steps will make the script idempotent and allow the user to republish the new version of code during development.

The playbooks that will be executed in this sections are listed as following:

RepublishTStream.yml: publishing harvester using Tweet stream API on instances within the group “stream”.

RepublishTSearch.yml: publishing harvester using Tweet search API on instances within the group “search”.

RepublishTSearchTrump.yml: publishing harvester using Tweet search API to search tweets against specific topics on instances within the group “searchTopic”.

RepublishAnalyse.yml: publishing tweet analysers on instances within the group “sever”.

RepublishServer.yml: publishing the web application on instances within the group “sever”.

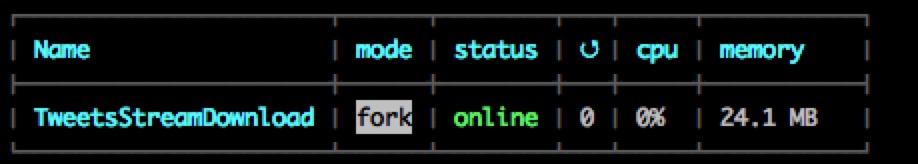
The user could execute the above playbooks sequentially to run the tasks on the specified instances by running the command:

*ansible-playbook FILENAME -i ansibleinventory.yaml --private-key YOUR\_PRIVATE\_KEY*

For running the analyser and harvesters, a tool called PM2 is used to make the analyser and harvesters run as background processes. To list all the tasks currently running at the instance, the user could SSH to each of the instances and run the command:

*(sudo) pm2 list*

By doing so, the user should see the corresponding tasks are running which indicates the harvesters are published correctly.



While PM2 worked fine for other tasks like harvesters and analysers, PM2 may behave incorrectly when the user tries to run the NodeJS server programmatically, which seems to be a unsolved bug of PM2. So the user should SSH into the remote server and manually start the nodejs server by using the following command:

pm2 start comp90024-australiacityanalytics/WebApplication/bin/www