CSC326 - Programming Languages

Lab 2
Third Party API Integration, Cloud Deployment & Benchmarking

In lab 1, you have setup the web framework and the crawler. In this lab, you will complete the deployment flow for launching a web application on Amazon Web Service (AWS), and integrating your application with a third party login system, Google Login.

By the end of this lab, you should have a complete workflow for developing, deploying, and benchmarking your search engine in the cloud.

Frontend

F1. Register your web application on Google

In order to use the Google APIs, you must register your web application through Google Developers Console on https://code.google.com/apis/console

You may create a new project with a name of your preference, and obtain a new Client ID by clicking APIs & auth -> Credential tab on the left of the project page. The information of the application can be downloaded as a JSON file, which will be needed by your application to access the Google service.

The redirect URIs and Javascript origins should also be registered for your application. If you are developing the login feature on localhost with port 8080, you may set the redirect URI to http://localhost:8080. Do not use HTTPS in your redirect URIs or Javascript origins.

F2. Google Login

Google login APIs uses the OAuth 2.0 protocol, which has a flow as listed below. (*The code in the following examples uses two libraries oauth2client and googleapiclient, which can be download using "git clone --recursive git://github.com/google/google-api-python-client.git"*.)

1. User sends a sign-in request to your application server from a browser.
2. Application server generates an authentication URL base on the CLIENT_ID of your application. The generated URL is returned to the browser, which is then redirected to the Google login prompt for user authentication.

```
from oauth2client.client import OAuth2WebServerFlow
```

```
from oauth2client.client import flow_from_clientsecrets
from googleapiclient.errors import HttpError
from googleapiclient.discovery import build

@route('/', 'GET')

def home():

    flow = flow_from_clientscrets("client_secrets.json",
                        scope='https://www.googleapis.com/auth/plus.me
https://www.googleapis.com/auth/userinfo.email',
                        redirect_uri="http://localhost:8080/redirect")

    uri = flow.step1_get_authorize_url()

    bottle.redirect(str(uri))
```

3. Once user is authenticated, the browser will be redirected to the Google authorization prompt to grant access permission for the application server.
4. If user authorizes your application server to access the Google services, an one-time code will be attached to the query string when the browser is redirected to the redirect_uri specified in step 2. The one-time code can be retrieved as GET parameter:

```
@route('/redirect')

def redirect_page():

    code = request.query.get('code', '')
```

5. The one-time code can be used to exchange for an access token by submitting an HTTP request with the code, the CLIENT_ID, and the CLIENT_SECRET to Google server.

```
@route('/redirect')

def redirect_page():

    . . .

    flow = OAuth2WebServerFlow( client_id=CLIENT_ID,
                            client_secret=CLIENT_SECRET,
                            scope=SCOPE,
                            redirect_uri=REDIRECT_URI)

    credentials = flow.step2_exchange(code)

    token = credentials.id_token['sub']
```

6. Once your application server receives the access token from Google, your application can retrieve user's data through Google APIs with the credentials.

```
http = httplib2.Http()
http = credentials.authorize(http)

# Get user email
users_service = build('oauth2', 'v2', http=http)
user_document = users_service.userinfo().get().execute()
user_email = user_document['email']
```

For details about the protocol, please see https://developers.google.com/accounts/docs/OAuth2 and https://developers.google.com/api-client-library/python/guide/aaa_oauth

Note that alternative approaches are available for Google login mechanism, such as using Javascripts, and you are free to choose your preference of implementation.

CSC326 - Programming Languages

F3. Session Management In Bottle.py

Every time when a user authenticates the Google account to access your web service, your server should maintain a session for the user, such that the user is not required to login to your website again unless the session is expired or the user signs out explicitly.

The bottle web framework does not have built-in support for session management. You may implement your own session management mechanism; or, you can use the Beaker library. The documentation for Beaker library can be found at http://beaker.readthedocs.org/en/latest/. An example for using the Beaker library is provided at http://bottlepy.org/docs/dev/recipes.html

F4. Requirements

Your web application should be accessible in two modes:

1. Anonymous Mode
    - User may access your web application without signing in;
    - Search history is not displayed in this mode, i.e. Lab 1 without the history table.
2. Signed-In Mode
    - User is required to sign in with Google Login;
    - User is only required to authorize your application to access Google services for the first time of signing in. Subsequent logins does not require user to provide such authorizations;
    - Indication of user identity, e.g. email address, must shown on every page of the website. If your application also retrieve user's Google+ profile, such as picture, user name, and etc., you should not assume all users enabled their Google+ profile, and your application should handle scenarios when such information is not available.
    - Sign-out button must be provided on every page of the website. A proper sign-out must not revoke user authorization for your application.
    - At least 10 most recently search words by the user should be stored, and displayed on the query page.
    - Search history should be accessible after user logs out, and signs in through different browsers or devices.
    - After a user signs out, the user should not be allowed to access the signed-in features, e.g. history table,without signing in again, that is such features will be unavailable if they refresh or go back to the previous page.

Backend

In this lab, the backend member is responsible for deploying the web application, both the frontend and the backend, on Amazon Elastic Compute Cloud (EC2) from Amazon Web Services (AWS).

B1. AWS SDK for Python (Boto)

AWS SDK for Python, a.k.a Boto, can be found on http://aws.amazon.com/sdk-for-python/.

CSC326 - Programming Languages

AWS EC2 should be used to run a new server instance. Reference of EC2 API can be found on http://boto.readthedocs.org/en/latest/ref/ec2.html.

To launch an EC2 instance, typical steps are list as following:

On AWS Management Console:

1. Create an AWS account on http://aws.amazon.com
2. Create a new user with AWS Identity and Access Management (IAM).
   Instruction for creating a new user can be found on
   http://docs.aws.amazon.com/IAM/latest/UserGuide/Using_SettingUpUser.html.
3. Download AWS Access Key and Security Key, which are required for using AWS SDKs.

In Python script:

1. Establish connection to region “us-east-1” along with aws_access_key_id and aws_secret_access_key

2. Create Key-Pair with boto.ec2.connection.create_key_pair(), which returns a key-pair object, boto.ec2.keypair.KeyPair. The key must be save as a .pem key file using boto.ec2.keypair.KeyPair.save(<directory>). The .pem key file is needed for SSH the new instances.

3. Create a security group with boto.ec2.connection.create_security_group(), which returns an instance of boto.ec2.securitygroup.SecurityGroup. Security group provides restricted access only from authorized IP address and ports. For more details, see See http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html
4. Authorize following protocols and ports for the security group created in step 3:
   4.1. To ping the server, enable
   protocol: ICMP, from port: -1, to port: -1, CIDR IP 0.0.0.0/0
   4.2. To allow SSH, enable
   protocol: TCP, from port: 22, to port: 22, CIDR IP 0.0.0.0/0
   4.3. To allow HTTP, enable
   protocol: TCP, from port: 80, to port: 80, CIDR IP 0.0.0.0/0
5. Start a new instance with boto.ec2.connection.run_instance().
   To find Amazon Machine Image (AMI) IDs of Ubuntu server images in various regions, please see http://cloud-images.ubuntu.com/releases/14.04.1/release-20140927
   Make sure the property of the selected image matches the instance type and region of your selection. For the specification of different EC2 instance types, see http://aws.amazon.com/ec2/instance-types. Note that for the purpose of this lab, it is sufficient that you use the Micro Instance with the free tier usage.
6. Step 5 returns a reservation object, which contains a list of instances newly create. States of the instance can be retrieved as variable of boto.ec2.instance.Instance.
7. Once the state of the instance is changed to "running", you can access your instance with the key-pair generated in step 1 with the following command
   ```
   $   ssh -i key_pair.pem ubuntu@<PUBLIC-IP-ADDRESS>
   ```
   Note that the default user name for the Ubuntu AMIs is "ubuntu". The public IP address of the instance can be found with boto.ec2.instance.Instance.ip_address
8. To copy a file from your local machine to the AWS instance, you may use the following command.

```
$   scp -i key_pair.pem <FILE-PATH> ubuntu@<PUBLIC-IP-ADDRESS>:~/<REMOTE-PATH>
```

B2. Setup static IP address

Every time you terminate an instance and launch a new one, the IP address may be assigned differently. In order to associate your instance to an static IP address, an EC2 Elastic IP address is needed.

To allocate a new elastic IP address, use boto.ec2.connection.allocate_address(), which returns an instance of boto.ec2.address.Address.

To associate an elastic IP address to a running instance, use boto.ec2.address.associate(). Note that an elastic IP address if free of charge only if it is associated with a running instance.

B3. Terminate V.S. Stop an instance

If you start an EC2 instance with EBS block devices and stop it, all data on the EBS device persist after it is restarted. To verify that your instance uses EBS as root device, you may check the rootDeviceName attribute of the instance. When an instance is stopped, all data on a non-EBS device will be removed permanently, and will NOT be accessible after the instance is restarted.

When an instance is terminated, all changes or data in the instance will be removed permanently.

B4. Requirement

- An active instance with your web application must be started and stay online for 1 week from the due date of this lab.
- You should provide a python script for launching a new instance on AWS. You may remove the ACCESS KEY and SECRET KEY in the submission files. Operations for binding an elastic IP does not need to be included in the script.
- Security group must be name as “csc326-group<group_number>”

Preliminary Benchmarking

After the deployment of a web service, it is important to analyze the performance of the application on the server. There are many different metrics you may use to evaluate your application. For example, the average response time to process one request, the number of requests handled per second, the throughput of the server, and etc.

To evaluate the performance of your web application on AWS, you may use the Apache benchmarking tool, ab. You can install it on Ubuntu with following command:

```
$ sudo apt-get install apache2-utils
```

You may use the following command to run a simple benchmark:

```
$ ab -n <number of request to perform>  -c <number of concurrent connection>
http://hostname/path
```

For example, the following command sends 50 concurrent identical requests with keywords “helloworld foo bar” to http:/localhost:8080/ with a total of 1000 requests.

```
$ ab -n 1000 -c 50 http://localhost:8080/?keywords=helloworld+foo+bar
```

Alternative benchmark tools may be used. For example, wrk can be used to generate http traffic with dynamic URIs by using a Lua script. (source code: https://github.com/wg/wrk)

Monitor Resource Utilization

To monitor the status of your server, various tools can be used to collect the utilization of the CPU, memory, network, disk, and etc. Tools such as vmstat, mpstat, iostat, dstat, and etc. can be helpful. To install these tools in Ubuntu, use the following command:

```
$ sudo apt-get install sysstat dstat
```

Be aware that the benchmark driver should not run on the same machine with your web application since it also consume significant amount of resources on the server. Resource overhead for the monitoring tools may be ignored.

Requirements:

Benchmark driver should send requests to the server as in Anonymous Mode to avoid the complexity of managing credentials. Report following measurements:

- Maximum number of connections that can be handled by the server before any connection drops.
- Maximum number of requests per second (RPS) that can be sustained by the server when operating with maximum number of connections.
- Average and 99 percentile of response time or latency per request
- Utilization of CPU, memory, disk IO, and network when max performance is sustained

Deliverable

1. Source code of the frontend with integrated Google Login API
2. Python script for launching EC2 instance on AWS.
3. README file with following information:
    a. public IP address of your live web server;
    b. enabled Google APIs, if applicable;
    c. benchmark setup;
4. RESULT file with the preliminary benchmark result for the web application
    a. bonus mark may be rewarded to those with impressive performance results using the minimal hardware on AWS, i.e. t2.micro.
    b. Hint: non-blocking event handling for request processing

You may remove credentials of the Google and AWS API from the submission file by replacing corresponding values by “xxxxxxxxxx”. Do not remove such variables directly from your source code.

Submission

Compress all your files, including the source codes and text files, and name it lab2_group_<group_number>.tar.gz

To submit your package, use the following command on EECG machine:

```
$ submitcsc326f-lab 2 lab2_group_<group_number>.tar.gz
```

Expenses of AWS EC2

The AWS Free Tier provides new registered accounts free AWS services with limited usage, which should be sufficient for the purpose of the labs. For information about the free tier, please see http://aws.amazon.com/free/.

Important: It is expected that you are responsible to the usage of the account so that it is within the limit of free tiers. For pricing when usage limit is exceeded, please see http://aws.amazon.com/ec2/pricing.

Important: note that the network traffic between different EC2 instances within the same region is free of charge. You should monitor the network traffic usage carefully when benchmarking your web application to avoid unexpected extra charges.

Important: For AWS security key pairs, please note that you must not put it to any public repositories, such as github. There is a known crawler that will scan all repo on github for AWS secret keys. Once the secret keys are stolen, the victim account will start as many machines as possible in all AWS regions. Although AWS may detect and stop such operations, the minimum charge on the bill will be $3000+. AWS may waive the charge, but the account will be suspended for a few weeks.