Lab 3 - Deployment & Benchmarking

In lab 1 and lab 2, you have finished the development of a basic search engine. In this lab, you will deploy your private search engine on Amazon Web Services (AWS) infrastructure.

In order to access your search engine, a user is required to sign in with their Google account. Once they are signed in, your web site should provide the search engine service that you developed from Lab1 and Lab2.

Frontend

Register your web application on Google

Before you can use the Google APIs, you must register your web application through Google API Console at https://code.google.com/apis/console.

If your application is registered successfully, a client Id and a client secret will be assigned to your application. You can access the information about your application by clicking the "API Access" tab on the left of the Google API Console.

You should also register redirect URIs and Javascript origins 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.

Google Login

Google login implements the OAuth 2.0 Protocol.The protocol has the following steps, and related APIs are provided from the oauth2client library.

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

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

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

flow = flow_from_clientscrets("client_secrets.json",
                        scope='https://www.googleapis.com/auth/plus.me
                               https://www.googleapis.com/auth/userinfo.email',
                        redirect_uri='localhost:8080/redirect'')
uri = flow.step1_get_authorize_url()
bottle.redirect(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 the 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 back to your application server from the Google server. The request will be sent the the page specified at redirect_uri in step 2. The code can be retrieved as GET parameter:

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

5. Your application can exchange the one-time code for an access token by submitting an http request with the one-time code, the CLIENT_ID, and the CLIENT_SECRET to Google server.

```
flow = OAuth2WebServerFlow( client_id=CLIENT_ID,
                            client_secret=CLIENT_SECRET,
                            scope=SCOPE,
                            redirect_uri=REDIRECT_URL)
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 access token.

```
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']

# Get user name
users_service = build('plus', 'v1', http=http)
profile = users_service.people().get(userId='me').execute()
user_name = profile['displayName']
user_image= profile['image']['url']
```

For details about the protocol, you may visit https://developers.google.com/accounts/docs/OAuth2 and

https://developers.google.com/api-client-library/python/guide/aaa_oauth

Alternatively, you may use Google+ Sign In mechanism with Javascript. Details can be found at https://developers.google.com/+/web/signin/

Google+ Login Example

An example provided by Google can be found at https://developers.google.com/+/quickstart/python. Note that the example uses the Flask web framework, which is similar but not identical to the Bottle web framework.

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 till the session is expired or the account is signed out by the user manually. The bottle web framework does not have built-in support for session management. However, you may use the Beaker library to implement the sessions. 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

Requirements

- Your frontend should provide a login page with a "Sign in with Google" button, which redirects user to sign in with their Google account. If you decide to use Google+ Sign-In, you must follow the Google+ Sign-In button branding guidelines from https://developers.google.com/+/branding-guidelines
- Users should not be allowed to access your search engine feature without signing in their Google accounts.
- Your frontend should provide a button for users to sign out their Google account.
- After a user is signed out, the user should not be allowed to access your search engine feature without signing in again, i.e. features will be unavailable if they refresh the previous page.

Backend

In this lab, the backend member is responsible for deploying the entire search engine, both frontend and backend, to Amazon Web Services.

AWS Command Line Interface Deployment

Instructions of AWS deployment in this lab handout uses the AWS Command Line Interface (CLI). You may find the documentation of AWS CLI on http://aws.amazon.com/cli. You must download the credential for the new user to obtain the access key and secret key to use the CLI.

Create AWS account

You need to sign up an account on http://aws.amazon.com, and create a 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.

Setup AWS CLI

Download botocore package. Botocore provides a low-level interface to AWS.

```
$   git clone https://github.com/boto/botocore.git
$   cd botocore
$   python setup.py install --user
```

Download the command line interface package for AWS.

```
$   git clone https://github.com/aws/aws-cli.git
$   cd aws-cli
$   python setup.py install --user
```

You may add aws-cli/bin to $PATH for convenience.

To set up the configuration values for the CLI environment, the configuration file should include the AWS ACCESS KEY and AWS SECRET ACCESS KEY for a user created by IAM. Save the configuration file at <PATH-TO-CONFIG-FILE>, and add it to environment variable. Create a configuration file with following lines.

```
[default]
aws_access_key_id=<key_id>
```

```
aws_secret_access_key=<secret>
region=us-east-1
```

Add following to ~/.bashrc if you use bash shell.

```
export AWS_CONFIG_FILE=<PATH-TO-CONFIG-FILE>
```

or following line to ~/.cshrc if you use csh shell

```
setenv AWS_CONFIG_FILE <PATH-TO-CONFIG-FILE>
```

Execute the following commands after the files are modified.

```
$ source ~/.bashrc
```

```
Or
```

```
$ source ~/.cshrc
```

To verify that the environment variables are set correctly, echo the variable in the shell and ensure that it returns the corresponding value.

```
$ echo $AWS_CONFIG_FILE
```

Create a new server instance

1) Create EC2 key-pairs.

```
$   aws ec2 create-key-pair --key-name <key-name>
```

A key-pair is used to access your new instance after it is created. If a key-pair is created successfully, it returns a JSON string with 3 keys, KeyMaterial, KeyName, and KeyFingerprint. The value of KeyMaterial must be copied and pasted to a file, e.g. key_pair.pem. The "\n" string must be replaced to the new line character. Alternatively, you may set the format of the return output to plain text with option "--output text".

See http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html for details about Amazon EC2 key-pairs.

2) Create security group.

```
$   aws ec2 create-security-group --group-name <name> --description "<description>"
```

A security group defines the protocols and ports that can be used to access the instance. It acts

as a firewall to control the traffic to your server.

If the security group is created successfully, it returns a JSON string with 2 keys, return and GroupId, where the return value should be "true"

See http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html for details about Amazon security group.

3) Configure the security group with a file specifying ip permissions and protocols for accessing your instance. Below is an example of the configuration file for security group in JSON format.

```
[{
"IpProtocol":"icmp",
"FromPort":-1,
"ToPort":-1,
"IpRanges":[{"CidrIp":"0.0.0.0/0"}]
},
{
"IpProtocol":"tcp",
"FromPort":80,
"ToPort":80,
"IpRanges":[{"CidrIp":"0.0.0.0/0"}]
},
{
"IpProtocol":"ssh",
"FromPort":22,
"ToPort":22,
"IpRanges":[{"CidrIp":"0.0.0.0/0"}]
},
{
"IpProtocol":"udp",
"FromPort":0,
"ToPort":65535,
"IpRanges":[{"CidrIp":"0.0.0.0/0"}]
}]
```

The above configuration give permission for any IP addresses (0.0.0.0/0) to send ICMP, TCP, SSH, and UDP requests to corresponding ports.

Use the following command to configure the ip permissions defined in the file.

```
$   aws ec2 authorize-security-group-ingress --group-name csc326 --ip-permissions
file://<PATH-TO-SECURITY-GROUP-CONFIG-FILE>
```

4) Once you have set up the key-pair and security group, you are now ready to launch a new instance with following command.

```
$   aws ec2 run-instances --image-id <AMI_ID> --count 1 --key-name <KEY_PAIR_NAME>
--security-groups <SECURITY_GROUP_NAME> --instance-type <INSTANCT_TYPE>
```

Example:

```
$   aws ec2 run-instances --image-id ami-d0f89fb9 --count 1 --key-name csc326
--security-groups csc326 --instance-type t1.micro
```

Above command launches one new "t1.micro" instance using image "ami-d0f89fb9" using Ubuntu 12.04 LTS 64-bits, with key pair named "csc326" and security group named "csc326".

If the instance is launched successfully, the command returns an instance description similar to following:

```
{
    "OwnerId": "2408207xxxxx",
    "ReservationId": "r-0c17316a",
    "Groups": [
        {
            "GroupName": "csc326",
            "GroupId": "sg-f15fd59a"
        }
    ],
    "Instances": [
        {
            "Monitoring": {
                "State": "disabled"
            },
            "PublicDnsName": null,
            "KernelId": "aki-88aa75e1",
```

```
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "EbsOptimized": false,
            "LaunchTime": "2013-09-03T03:05:23.000Z",
            "ProductCodes": [],
            "StateTransitionReason": null,
            "InstanceId": "i-b5c876dd",
            "ImageId": "ami-d0f89fb9",
            "PrivateDnsName": null,
            "KeyName": "csc326",
            "SecurityGroups": [
                {
                    "GroupName": "csc326",
                    "GroupId": "sg-f15fd59a"
                }
            ],
            "ClientToken": null,
            "InstanceType": "t1.micro",
            "NetworkInterfaces": [],
            "Placement": {
                "Tenancy": "default",
                "GroupName": null,
                "AvailabilityZone": "us-east-1a"
            },
            "Hypervisor": "xen",
            "BlockDeviceMappings": [],
            "Architecture": "x86_64",
            "StateReason": {
                "Message": "pending",
                "Code": "pending"
            },
            "RootDeviceName": "/dev/sda1",
            "VirtualizationType": "paravirtual",
            "RootDeviceType": "ebs",
            "AmiLaunchIndex": 0
        }
    ]
```

```
}
```

Following commands may be used to retrieve the status of your instances.

```
$   aws ec2 describe-instances
$   aws ec2 describe-instance-status
```

For a list of Ubuntu 12.04LTS Amazon Machine Image (AMI) Ids in different regions, see http://cloud-images.ubuntu.com/releases/precise/release-20131024/

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.

5) 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 command "aws ec2 describe-instances".

6) To copy a file from your local machine to the AWS instance, you may use the following command.

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

Setup static IP address

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

Use the following command to allocate a new Elastic IP address:

```
$   aws ec2 allocate-address
```

When an Elastic IP address is allocated, you can associate it with a running instance, or you can keep the IP address if no instance is running.

Use the following command to associate the Elastic IP address to your instance:

```
$   aws ec2 associate-address --public-ip <PUBLIC_IP> --instance-id <INSTANCE_ID>
```

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 RootDeviceType using "aws ec2 describe-instances". 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.

Crawler

For this lab, you may create static databases before the AWS deployment, and your crawler is not required to update the databases in real time when the server is deployed on AWS.

Requirement

- An instance with your search engine application must be started and stay online for 2 weeks from the due date of this lab.
- You should provide an installation package, named *installPackage.tar.gz*, with all the files needed to setup your search engine.
- You should provide a bash script named, *install.sh*, to install your application as well as other utility packages or libraries needed for setting up a new server using an unmodified Ubuntu AMI. Such that a server can be setup in any new instance with following commands:

  ```
  $   tar -zxvf installPackage.tar.gz
  $   bash install.sh
  ```

Benchmarking

After the development of your web service is completed, it is important to analyze the performance of your application. 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.

Google provides PageSpeed Insights to analyze the performance of your web pages and provide suggestions for optimizing your HTML, CSS, and Javascripts files. It can be accessed on https://developers.google.com/speed/pagespeed/insights/

To evaluate the performance of your web server, 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 requests to http:/localhost:8080/ with 1000 requests in total.

```
$ ab -n 1000 -c 50 http://localhost:8080/
```

To monitor the status of your server, you should also collect that statistics of the CPU and memory usage. Tools such as vmstat, mpstat, and etc. can be used to collect required data. To install these tools in Ubuntu, use the following command:

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

To run the tools and sends data to log,

```
$ vmstat 1 | tee vm.log
$ mpstat 1 | tee mp.log
```

Requirement:

Collect and report the following measurement

- Maximum number of connections that can be handled by your server before any

connection drops.

- Maximum number of requests that can be handled by your server before your server crashes
- Illustrate the changes of the CPU and memory usage over time in graphs

Deliverable

1. IP Address of your live web server
2. Source code of the frontend
3. installPackage.tar.gz for installing your web application on unmodified Ubuntu AMI based EC2 instance.
4. Lab Report:
    a. Methodologies used to benchmark your web application
    b. Report the statistic collected from the benchmarking
    c. Describe the contribution for each member
    d. Instruction to install and run your code on a new EC2 instance

Submission

Code

Compress all your files, including the source code and the report, and name it lab3_group_<group_number>.tar.gz with tar.

For example, for group 4, use the following command.

```
$ tar -zcvf lab3_group_4.tar.gz <all files>
```

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

```
$ submitcsc326f 3 lab3_group_<group_number>.tar.gz
```

Progress Report

Add the Lab 3 report to the document you created for Lab 1.

This document should be used as a working document for all your labs, i.e. at the end of the project, this document should include all your lab work done from Lab 1 to Lab 4.

Think of the final report as the documentation for the whole project (with Architecture, Design, Implementation, Testing, Build and Deploy instructions. In addition to these sections, it should have a Table of Contents and Revision History table at the beginning of the report).

Grades

Google Login	4
AWS Deployment	4
Benchmarking	2
Report	2
Total grade for Lab 3	12

Attendance and Grace Period

Attendance will be taken at the beginning of each lab. For those who have attended all the labs, they will have a total of 2 grace days for lab submission in the semester. For those who have missed only one lab in the semester, they will have 1 grace day. If you miss more than one lab, you will have no grace days.

All your lab submission will be graded. However, if you submit a lab late, and you have not acquire any grace period at the end of the semester. Your late submission will receive a penalized grade.