Lab 2 - Web Server, Firewall Access, Monitor Cloud Usage

Introduction:

Welcome to Lab 2! In this lab you’ll learn how to set up a web server and host a web page. You’ll learn how to Setup firewall rules. You’ll also learn to monitor cloud usage.

**General Instructions:**

1. Ensure your AWS Educate account is set up, in case of any issues contact your lab in-charge.
2. Evaluation for the labs will be mixed, few labs will involve uploading screenshots and few might be auto-evaluated.
3. All the screenshots (for each task as mentioned in this document) and any other deliverable (as applicable to the experiment and mentioned in the instructions) must be uploaded to Edmodo before the stipulated deadline.
4. Every task has a number of questions, these will **not** be evaluated unless you are explicitly asked to submit the answers but are extremely important in understanding the concepts.
5. Try to solve the tasks by yourselves, all relevant information to complete the tasks have been provided. Watch demo video pertaining to this experiment. In case you are stuck and not able to solve the issue, feel free to ask your doubts on Edmodo or send email to your lab incharge or pesu\_cc\_lab\_support@googlegroups.com

# **Task A:**

**Objective:**

The Objective of this task is to setup up a web server on the AWS EC2 Instance and host a sample web page on the instance. You are also required to load test the web page by using ApacheBench and monitor the EC2 instance using AWS CloudWatch service

# Create a EC2 Instance

Deliverables:

* + 1. 1a.png Showing that Instance is Running

# Install Apache web Server on AWS EC2 Instance

Deliverables:

* + 1. 2a.png Showing that apache server status is active

1. Setup Firewall Rules

Deliverables:

* + 1. 3a.png showing the list of rules updated
    2. 3b.png SS of the local browser showing EC2 Instance’s Apache2 Ubuntu Default page ( Use EC2’s public DNS)

1. Host a Sample web page
   * 1. 4a.png SS of the hosted webpage(Should be accessed from your local computer)
2. Apachebench and Cloudwatch
   * 1. 5a.png SS of the Apachebench output report After complete Execution
     2. 5b.png SS of the All set of graphs.
     3. 5c.png SS of cpu utilization

* **Reading –** 20 mins
  + [What is a Web Server?](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server) : Understand what a web server is.

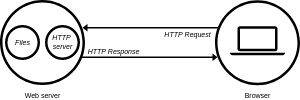
## Sub Task 1: Launch a AWS EC2 Instance

* Launch a EC2 instance with the below Mentioned Configuration
* Follow **lab 1** steps to launch the instance.

**EC2 instance specs:**

|  |  |
| --- | --- |
| **AMI** | Ubuntu Server 18.04 LTS (HVM), SSD Volume Type |
| **Instance Type** | t2.medium |
| **Instance Details** | Default values |
| **Storage** | Default values |

## Sub Task 2: install apache web server on the instance



It is important to understand the concept of web server before we jump into installing one.

The term web server can refer to hardware or software, or both of them working together.

* **On the hardware side**, a web server is a computer that stores web server software and a website's component files. (for example, HTML documents, images, CSS stylesheets, and JavaScript files) A web server connects to the Internet and supports physical data interchange with other devices connected to the web.
* **On the software side**, a web server includes several parts that control how web users access hosted files. At a minimum, this is an HTTP server. An HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

**Why Apache Web Servers?**

Apache is considered open source software, which means the original source code is freely available for viewing and collaboration. One of the pros of Apache is its ability to handle large amounts of traffic with minimal configuration. It scales with ease and with its modular functionality at its core, you can configure Apache to do what you want, how you want it. You can also remove unwanted modules to make Apache more lightweight and efficient.

steps:

* SSH to the instance
* install [Apache2](https://www.liquidweb.com/kb/install-apache-2-ubuntu-18-04/)
* check the apache2 status

Sub Task3: Setup Http and Firewall access

Normally, Amazon computers only allow shell logins via ssh (port 22 access). If we want to run a Web service or something else, we need to give the outside world access to other network locations on the computer.

Steps:

* Find “Security Groups” in the lower pane of your instance’s information page, and click on “launch-wizard-N” hyperlink
* Edit the Inbound Rules
* Add a new rule: HTTP, 80, Source Anywhere and save them.
* Wait for few seconds and copy paste the Public DNS of your instance in your local browser. You should be able to see apache2 home page.

Refer this [link](https://ec2-tutorials.readthedocs.io/en/latest/configure-firewall.html) for detailed explanation.

Sub Task4: Host a Sample Web page

* You need to host a simple html Web Page in your instance using Apache Web Server and you should be able to access that webpage from your local web browser
* Create a Folder inside your “\var\www\html\” folder of your instance
* Name the folder by your SRN
* Create a simple html Page having your Name and SRN
* Access the Webpage from your local Browser using Your Instances PUBLIC DNS.

Sub Task5: Apache Benchmark and Cloudwatch

**What is Apachebench?**

**ApacheBench** is a single-threaded command line computer program for measuring the performance of HTTP web servers. Originally designed to test the Apache HTTP Server, it is generic enough to test any web server.

**What is AWS Cloudwatch?**

**Amazon CloudWatch** is a monitoring and management service that provides data and actionable insights for AWS, hybrid, and on-premises applications and infrastructure resources. With CloudWatch, you can collect and access all your performance and operational data in form of logs and metrics from a single platform.

**Step**

* In this task you are supposed to create load on your web server which you have setup on the ec2 Instance by using apachebench from your local computer.
* Then Monitor the load using Amazon cloudwatch( By enabling Detailed Monitoring)
* Identify the available metrics
* Vary the load using Apachebench,Following Load should be generated :
  + Users:10 Hits: 2000
  + Users: 20 Hits: 4000
  + Users: 30 Hits: 6000
  + Users: 40 Hits: 10000
* Identify the Metrics Whose values are changing and Take Screenshot of the Graphs.
* In order to monitor CPU Utilization Install stress-ng on your ec2 instance and then Create stress load of 60% and 80%. Take SS of CPU utilization curve.

Reference Link:

* [Apachebencmark](https://stackoverflow.com/questions/12732182/ab-load-testing)
* [Cloudwatch](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch-new.html) your EC2 Instance (Look for enabling Detailed Monitoring for an existing EC2 Instance)

**TASK B: Deploying Flask app On AWS Elastic Beanstalk**

**Objective:**

In this task you’ll be asked to create a AWS Elastic Beanstalk environment and deploy the flask application provided to you. You are required to configure a RDS along with your Beanstalk environment and create a few tables and databases.

**Deliverables:**

1. B1.png SS of Environment Health
2. B2.png SS of DB configuration(RDS)
3. B3.png Register User Request(Postman SS)
4. B4.png Verify User Request(Postman SS)

**What is AWS Elastic Beanstalk?**

**AWS Elastic Beanstalk** is an orchestration service offered by Amazon Web Services for deploying applications which orchestrates various AWS services, including EC2, S3, Simple Notification Service, CloudWatch, autoscaling, and Elastic Load Balancers.

**AWS Elastic Beanstalk** is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

Flask is a lightweight Web Server Gateway Interface WSGI web application framework that was created to make getting started easy and making it easy for new beginners. With the tendency to scale up to complex applications.

**FLASK?**

**Flask** has its foundation around Werkzeug and Jinja2 and has become one of the most popular Python web application frameworks.

As a developer in developing a web app in python, you may be using a framework to your advantage. A framework is a code storage that should help developers achieve the required result by making work easier, scalable, efficient and maintainable web applications by providing reusable code or extensions for common operations.

References:

1. [Elastic Beanstalk – Deploy Web Applications](https://aws.amazon.com/elasticbeanstalk/)
2. [Introduction to AWS Elastic Beanstalk](https://www.youtube.com/watch?v=uiM1xzOX8Qg)

**Note: In this Experiment you’ll be provided with two files**

1. **requirements.txt** having the list of libraries required by the flask app.
2. **application.py** Flask application file having two API’s.

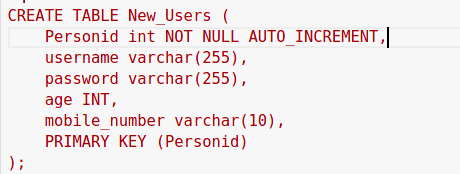
**Steps:**

1. Launch a AWS Beanstalk environment.
   1. Choose sample application code.
   2. Setup Application and Environment name (Application name should be your SRN).
   3. Select the environment type.

Reference for [Step1](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.CreateApp.html).

1. Adding an Amazon RDS DB instance to your environment
   1. Setup Username(It should be your SRN) and Passwords
   2. Setup a MySQL Instance.
   3. After creating RDS, goto RDS instance info page
   4. Edit the securitygroup of your RDS Instance like you did in TASK A for EC2 Instance. Edit the Inbound rules to allow traffic from Anywhere.

Reference for [Step2](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html).

1. Write a Python Script to connect to RDS Instance Then you need to create a Database and Table in the RDS Instance.
   1. Create a DB of your Name Make changes in the application.py based on it.
   2. Create a table in that DB as follows:
      1. 
2. Deploy the Flask App on Beanstalk.( Deploy using GUI)

Watch this [video](https://www.youtube.com/watch?v=P5kGTr2zfn4) to make your life simpler.

1. The Flask App has two REST API’s One API is used to register Users And Another API is used to verify the registered user.
   * 1. API- Register User
        1. JSON Input

{

"uname":"Ram",

"pswd":"qwerty",

"age":22,

"mob": 8050569402

}

* + - 1. Accepted Method: POST
    1. API- Verify User
       1. JSON Input

{

"uname":"UserName",

"pswd":"Password"

}

* + - 1. JSON Output (ID, Username, Password, Age, Mobile Number)

[

6,

"UserName",

"Password",

22,

"8898734571"

]

* + - 1. Accepted Method: POST

1. Test the API’s in the deployed Application Using POSTMAN. Create your own User and verify the user. SS needs to be taken.

Reference for [postman](https://www.guru99.com/postman-tutorial.html)