



Course: IT485

Project: HawkinCloud

Members: Aliyev Omar, Haider Zaheer, Michael Hernandez, Nephthro Pierre.

REPORT #3

Management Guidance:

As we approach Presentation II for our project, we are excited to delve deeper into the topic and explore additional aspects that may help improve our work. We understand that our project's success depends heavily on adhering to its specifications and timeline. Therefore, we plan to use this opportunity to discuss our project's specifics and explore ways to make the best use of our time. During our recent meeting on the 3rd of March, we discussed several strategies for managing progress and allocating tasks among group members. We recognized that each process within our project requires a certain level of attention and expertise to ensure that it is executed to the best of our ability. Therefore, we decided to manage progress and assign tasks to each group member based on their skill set and knowledge of each process.

Team Member participant:

We have started creating a base model AWS account and are focusing on individual user integration. Based on our understanding of the concept and our adoption of BOTO3, we have decided that each team member must configure their own AWS key and pairs. To better understand the concept and programming knowledge, we are sharing more documentation in the daily reading section on Discord and encouraging team members to watch online courses. As part of our progressive learning path, we plan to arrange a percentage of the work process for each team member and include it in our report. This will help us to better understand the workload and allocate tasks accordingly. We believe that this approach will help us to work more efficiently and effectively, ensuring that we achieve our goals in a timely manner.

Output Deliverables:

We are currently working on the database structure and tools for our project. To interact with our SQLite3 database, we are deploying Flask SQLAlchemy, which supports many different database systems, including MySQL and Oracle. User authentication has been successfully merged and tested in our system, and we are looking to improve the dashboard based on our available resources (**Source_1**).

o.aliyev001@umb.edu
m.zaheer001@umb.edu
michael.hernandez003@umb.edu
nephthro.pierre001@umb.edu

We plan to integrate Flask Material Dashboard, which can help enhance the front-end design. Currently, we are using the Bootstrap front-end framework for building a lightweight web dashboard. It provides a set of CSS and JS components, such as buttons, forms, dashboard, list div, and navigation bars, which can be improved by us during the development period. There are several benefits of using a requirements file. One of them is that it helps ensure consistency across different environments. For example, if multiple people are working on a project, they may have different versions of the same packages installed. By using a requirements file, everyone can ensure that they have the same versions of the required packages installed, which can help avoid issues with compatibility (Source_2). According to our process, we are running multiple commands to initialize the command prompt and manage dependencies in the project. Anyone who needs to install the same packages can do so easily.

```
app.config['SECRET_KEY'] = " "
app.config['SQLALCHEMY_DATABASE_URI'] = "database.db"
Bootstrap(app)
db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
```

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    form = RegisterForm()
    if form.validate_on_submit():
        # return '<h1>' + form.username.data + ' ' + form.email.data + ' ' + form.password.data + ' </h1>'
        hashed_password = generate_password_hash(form.password.data, method='sha256')
        new_user = User(username=form.username.data, email=form.email.data, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
```

```
git@aliyev@Aliyev: ~$ sqlite3 database.db
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .tables
user
sqlite> select * from user;
1|Hash[hash.com]sha256$snBYTQzJL371VY$ef940b4795c54e0447c90b9a6122ca8f19780453247f09cfc43541fe745e8a7c
sqlite> []
```

Source_1

Source_2

```
HawkinCloud/
├── venv/
│   ├── Source/
│   │   ├── __pycache__
│   │   ├── instance
│   │   ├── static/
│   │   │   ├── pics
│   │   │   ├── dashboard.css
│   │   │   ├── signin.css
│   │   │   └── starter-template.css
│   │   ├── templates/
│   │   │   ├── charts.html
│   │   │   ├── dashboard.html
│   │   │   ├── index.html
│   │   │   ├── login.html
│   │   │   ├── sg_graph.html
│   │   │   └── signup.html
│   │   ├── app.py
│   │   └── requirements.txt
│   ├── Include
│   ├── Lib/
│   │   └── Site-packages/
│   │       └── .....
│   ├── Scripts/
│   │   ├── activate
│   │   ├── activate.bat
│   │   ├── jp.py
│   │   └── pyenv.cfg
```

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

Output Deliverables:

To set up an AWS account as a user in the root account, we need to follow specific guidelines to ensure that users can manage access to AWS resources effectively. AWS is a cloud computing platform that offers a wide range of services and tools, including storage, databases, analytics, security, and more. By creating an AWS account as a user in the

```
PS C:\Users\Aliyev> aws kms list-keys
{
  "Keys": [
    {
      "KeyId": "7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
      "KeyArn": "arn:aws:kms:us-east-1:783210027625:key/7d96c5d7-289e-4bcc-bc5a-6dd21622e796"
    }
  ]
}




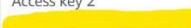
PS C:\Users\Aliyev> aws kms describe-key --key-id 7d96c5d7-289e-4bcc-bc5a-6dd21622e796
{
  "KeyMetadata": {
    "AWSAccountId": "783210027625",
    "KeyId": "7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
    "Arn": "arn:aws:kms:us-east-1:783210027625:key/7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
    "CreationDate": "2023-03-01T21:30:22.067000-05:00",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
  }
}
```

```
PS C:\Users\Aliyev> aws kms list-keys
{
  "Keys": [
    {
      "KeyId": "7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
      "KeyArn": "arn:aws:kms:us-east-1:783210027625:key/7d96c5d7-289e-4bcc-bc5a-6dd21622e796"
    }
  ]
}

PS C:\Users\Aliyev> aws kms describe-key --key-id 7d96c5d7-289e-4bcc-bc5a-6dd21622e796
{
  "KeyMetadata": {
    "AWSAccountId": "783210027625",
    "KeyId": "7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
    "Arn": "arn:aws:kms:us-east-1:783210027625:key/7d96c5d7-289e-4bcc-bc5a-6dd21622e796",
    "CreationDate": "2023-03-01T21:30:22.067000-05:00",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
  }
}
```

root account, we can manage access to these services and tools according to our specific needs. To explore appropriate permissions, we are first running the Flask app. Flask is a micro web framework written in Python that enables developers to create web applications quickly and easily. By using Flask, we can build a web application that interacts with AWS services, such as EC2, S3, and RDS. The interact with AWS services from our Flask app, we need to install Boto3 and the AWS CLI on our local machine in our local environment. Boto3 is the AWS SDK (Software Development Kit) for Python, which allows Python developers to write software that makes use of AWS services. The AWS CLI is a tool that enables users to interact with AWS services from the command line. By installing Boto3 and the AWS CLI, we can write Python code that interacts with AWS services and test our code on our local machine. Once we have installed Boto3 and the AWS CLI, we can configure our AWS account

on our local machine. We can enable the AWS Management console, which allows us to create IAM (Identity and Access Management) users' access key and secret key. IAM is a web service that enables users to manage access to AWS resources securely. After creating the IAM user and attaching an

Summary		
ARN   /user/aliyev	Console access Disabled	Access key 1  - Active ⌚ Used today. Created today.
Created February 26, 2023, 17:00 (UTC-05:00)	Last console sign-in -	Access key 2  - Active ⌚ Used today. 6 days old.

appropriate policy, such as AWScloudtrail_ReadOnlyAccess, we can create additional policies to manage access to AWS resources. One such policy is the ListUsers policy, which grants permissions to list IAM users in our account. By using the ListUsers policy, we can monitor and manage IAM users'

o.aliyev001@umb.edu
m.zaheer001@umb.edu
michael.hernandez003@umb.edu
nephthro.pierre001@umb.edu

Permissions policies (5)

Permissions are defined by policies attached to the user directly or through groups.

Find policies

<input type="checkbox"/>	Policy name	Type	Attached via
<input type="checkbox"/>	AWSCloudTrail_ReadOnlyAccess	AWS managed	Directly
<input type="checkbox"/>	KMS	Customer managed	Directly
<input type="checkbox"/>	KMS1	Customer managed	Directly
<input type="checkbox"/>	KMS3	Customer managed	Directly
<input type="checkbox"/>	ListUsers	Customer managed	Directly

access to AWS resources effectively. In addition to policies, we can also create security groups to manage network traffic to our EC2 instances. Security groups act as virtual firewalls that control inbound and outbound traffic to our instances. By creating security groups, we can ensure that only authorized traffic can access our instances, improving our overall security posture. To visualize our security group settings and network traffic, we can use Plotly.js to create interactive charts and graphs. Plotly.js is a JavaScript graphing library that enables developers to create interactive charts and graphs to visualize data. By using Plotly.js, we can create a security group graph chart that shows our security group settings and network traffic. The security group graph chart displays the number of inbound and outbound rules for each security group and the number of instances associated with each security group. We can use this chart to identify security groups with many inbound and outbound rules and instances, which may indicate potential security vulnerabilities. In addition to the security group graph chart, we can also use Plotly.js to create other interactive charts and graphs to visualize our data. For example, we

SG Data



can use Plotly.js to create a chart that shows the number of IAM users in our account and the permissions associated with each user. By using Plotly.js, we can create interactive visualizations that enable us to monitor and manage our AWS resources effectively.

```
* Debugger PIN: 735-547-733
127.0.0.1 - - [05/Mar/2023 14:35:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Mar/2023 14:35:17] "GET /static/starter-template.css HTTP/1.1" 304 -
127.0.0.1 - - [05/Mar/2023 14:35:17] "GET /static/pics/logo_4-removebg.png HTTP/1.1" 304 -
127.0.0.1 - - [05/Mar/2023 14:35:19] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [05/Mar/2023 14:35:21] "GET /charts HTTP/1.1" 200 -
127.0.0.1 - - [05/Mar/2023 14:35:35] "GET /sg_data HTTP/1.1" 200 -
127.0.0.1 - - [05/Mar/2023 14:35:44] "GET /sg_data HTTP/1.1" 200 -
```