

Labs 6-9

Student ID: 24188516

Student Name: Dayu Liu

Lab 6

Set up an EC2 instance

[1] Create an EC2 micro instance with Ubuntu and SSH into it.

In the first step, we will use the code in `lab2` to create a EC2 instance, stored the access private key, printed out the public IP address. Then we will SSH into the instance by providing the IP address and private key.

The following Python script uses `boto3` to create the EC2 **instance, security group, key pair, and instance tag**:

Workflow

1. Create Security Group:

Creates a security group (`24188516-sg-1`) using `ec2.create_security_group()`.

2. Authorize SSH/HTTP Inbound Rule:

SSH/HTTP rule is added using `ec2.authorize_security_group_ingress()`. This allows SSH access on port **22** and HTTP access on port **80** from all IP addresses (`0.0.0.0/0`).

3. Create Key Pair:

Key pair (`24188516-key-lab6`) is generated using `ec2.create_key_pair()`, and the private key is saved locally with restricted access permissions using `os.chmod()` to secure it.

4. Create EC2 Instance:

Creates an EC2 instance in the specified security group using `ec2.run_instances()`. The **AMI ID** (`ami-07a0715df72e58928`), **instance type** (`t3.micro`), and **key name** (`24188516-key-lab6`) are provided as parameters.

5. Tag EC2 Instance:

A name tag (`24188516-vm-1`) is created for the EC2 instance using `ec2.create_tags()`, which helps in identifying the instance easily.

6. Retrieve Public IP Address:

The public IP address of the newly created EC2 instance can be retrieved using `ec2.describe_instances()`.

```
# lab6.py
import boto3 as bt
import os

GroupName = '24188516-sg-1'
KeyName = '24188516-key-lab6'
InstanceName= '24188516-vm-1'

ec2 = bt.client('ec2')

# 1 create security group
step1_response = ec2.create_security_group(
    Description="security group for development environment",
    GroupName=GroupName
```

```

)

# 2 authorise ssh inbound rule
step2_response = ec2.authorize_security_group_ingress(
    GroupName=GroupName,
    IpPermissions=[
        {
            'IpProtocol': 'tcp',
            'FromPort': 22,
            'ToPort': 22,
            'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
        },
        {
            'IpProtocol': 'http',
            'FromPort': 80,
            'ToPort': 80,
            'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
        }
    ]
)

# 3 create key-pair
step3_response = ec2.create_key_pair(KeyName=KeyName)
PrivateKey = step3_response['KeyMaterial']
## save key-pair
with open(f'{KeyName}.pem', 'w') as file:
    file.write(PrivateKey)
## grant file permission
os.chmod(f'{KeyName}.pem', 0o400)

# 4 create instance
step4_response = ec2.run_instances(
    ImageId='ami-07a0715df72e58928',
    SecurityGroupIds=[GroupName],
    MinCount=1,
    MaxCount=1,
    InstanceType='t3.micro',
    KeyName=KeyName
)
InstanceId = step4_response['Instances'][0]['InstanceId']

# 5 create tag
step5_repsonse = ec2.create_tags(
    Resources=[InstanceId],
    Tags=[
        {
            'Key': 'Name',
            'Value': InstanceName
        }
    ]
)

# 6 get IP address
step6_response = ec2.describe_instances(InstanceIds=[InstanceId])

# Extract the public IP address

```

```
public_ip_address = step6_response['Reservations'][0]['Instances'][0]
['PublicIpAddress']

print(f"{public_ip_address}\n")
```

Code Breakdown

1. `ec2.create_security_group()`:

- **Description**: Describes the purpose of the security group, here labeled as "security group for development environment".
- **GroupName**: Defines the name of the security group, in this case, `24188516-sg-1`.

2. `ec2.authorize_security_group_ingress()`:

- **GroupName**: Specifies the security group where the rule will be added, in this case, `24188516-sg-1`.
- **IpPermissions**: This parameter contains the rules that specify what type of inbound traffic is allowed.
 - **IpProtocol**: Defines the protocol, here set to `tcp` for SSH access, and `http` for HTTP access.
 - **FromPort and ToPort**: Set to `22` for the SSH port and `80` for the HTTP port.
 - **IpRanges**: Defines the IP range allowed to access the instance. Here, `0.0.0.0/0` allows access from any IP.

3. `ec2.create_key_pair()`:

- **KeyName**: Specifies the name of the key pair, here `24188516-key-1ab6`, generates a new key pair and returns the private key.

4. `file.write()`:

- The private key is saved to a `.pem` file using Python's built-in File library with the `open()` function, and `os.chmod()` is used to set the file's permission to `400` (read-only).

5. `ec2.run_instances()`:

- **ImageId**: Specifies the Amazon Machine Image (AMI) ID, in this case, `ami-07a0715df72e58928`, which contains pre-configured software and settings.
- **SecurityGroupIds**: Lists the security group IDs that will be associated with the instance. Here, the security group is `24188516-sg-1`.
- **MinCount and MaxCount**: Define how many instances to launch. only one instance will be created in our case.
- **InstanceType**: Defines the type of instance to launch, in this case, `t3.micro`.
- **KeyName**: Specifies the name of the key pair, `24188516-key-1ab6`, used for SSH access.

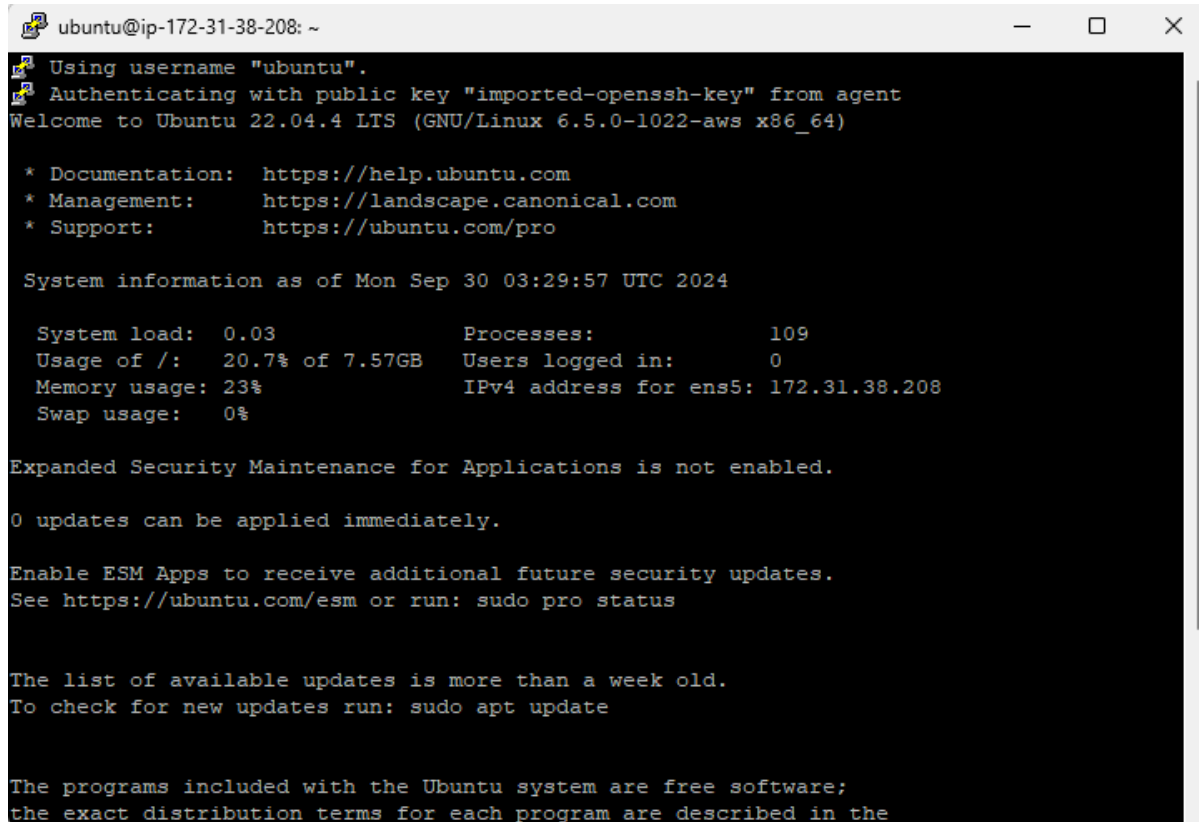
6. `ec2.create_tags()`:

- **Resources**: Specifies the resources to tag, in this case, the instance ID.

- `Tags`: Defines the key-value pairs for tagging. Here, the tag key is `Name` and the value is `24188516-vm-lab6`, which labels the instance for easier identification.
7. `ec2.describe_instances()`:
- `InstanceIds`: Specifies the instance ID to describe details on.

```
liudayubob@Dayu:~/cits5503/lab6$ python3 lab6.py
13.61.7.212
```

Now we can SSH into our instance by accessing `ubuntu@13.61.7.212` and using our generated pem private key.

A terminal window titled 'ubuntu@ip-172-31-38-208: ~' showing the output of an SSH connection. The window has standard Ubuntu window controls (minimize, maximize, close) in the top right. The terminal text includes: 'Using username "ubuntu".', 'Authenticating with public key "imported-openssh-key" from agent', 'Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1022-aws x86_64)', system documentation links, system information (load, processes, memory, etc.), and security update notices.

```
ubuntu@ip-172-31-38-208: ~
Using username "ubuntu".
Authenticating with public key "imported-openssh-key" from agent
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1022-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon Sep 30 03:29:57 UTC 2024

System load:  0.03          Processes:            109
Usage of /:   20.7% of 7.57GB Users logged in:        0
Memory usage: 23%          IPv4 address for ens5: 172.31.38.208
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
```

[2] Install the Python 3 virtual environment package

In this step, we will run the following commands to install virtual environment package and grant sudo permissions to bash operations.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-venv
sudo bash
```

1. Update and Upgrade System Packages:

- `update`: Updates the package lists for available or new versions of packages and their dependencies.
- `upgrade`: Upgrades the installed packages to the latest versions.

2. Install Virtual Environment:

- `install python3-venv`: Installs the `venv` package for Python 3, which is used to create isolated Python environments.

3. Switch to Superuser Mode:

- `sudo bash`: Elevates from current user session to superuser mode, ensuring all following commands are executed with `sudo` privileges without needing to specifying it every time.

```
ubuntu@ip-172-31-38-208:~$ sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-venv
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Fetched 129 kB in 1s (236 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  linux-aws linux-headers-aws linux-image-aws
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-venv is already the newest version (3.10.6-1~22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

[3] Access a directory

Now let's make a directory for our app files, create a directory called `/opt/wwc/mysites` and `cd` into the directory.

```
sudo mkdir -p /opt/wwc/mysites
cd /opt/wwc/mysites
```

1. Create Directories Using `mkdir`:

- `mkdir -p`: Creates the specified directory path (`/opt/wwc/mysites`). The `-p` option ensures that parent directories are created as needed without error if they already exist.

2. Navigate to the Created Directory:

- `cd /opt/wwc/mysites`: Changes the working directory to `/opt/wwc/mysites`. This is the directory where our project will be stored.

```
root@ip-172-31-38-208:/home/ubuntu# sudo mkdir -p /opt/wwc/mysites
root@ip-172-31-38-208:/home/ubuntu# cd /opt/wwc/mysites
root@ip-172-31-38-208:/opt/wwc/mysites#
```

[4] Set Up a Virtual Environment

To create a new isolated python environment, we can run the following command:

```
python3 -m venv myvenv
```

Key Parameters:

- `-m venv`: Uses the `venv` module to create a new virtual environment.
- `myenv`: Specifies the name of the directory to store the virtual environment.

We create a new directory called `myenv`, this allows us to manage dependencies and install packages separately from the global environment.

```
root@ip-172-31-38-208:/opt/wwc/mysites# python3 -m venv myenv
root@ip-172-31-38-208:/opt/wwc/mysites#
```

[5] Source the virtual environment

In this step, we will activate our virtual environment, install and start the Django project and create a Django app

```
source myenv/bin/activate
pip install django
django-admin startproject lab
cd lab
python3 manage.py startapp polls
```

1. Source Virtual Environment:

- `source myenv/bin/activate`: Activates the virtual environment `myenv`, setting the environment for isolated Python package management.

2. Install Django:

- `pip install django`: Installs Django into the virtual environment.

3. Start a New Django Project:

- `django-admin startproject lab`: Uses `django-admin` to create a new Django project named `lab` in the current directory. This generates necessary project files like `manage.py` and a folder structure to build the web application.

4. Create a New Django App:

- `python3 manage.py startapp polls`: Uses Django's `manage.py` as an App entry point to create a new app called `polls`. We can see that the `polls` app will have its own views, models, and URLs.

```

root@ip-172-31-38-208:/opt/wwc/mysites# source myvenv/bin/activate

pip install django

django-admin startproject lab

cd lab

python3 manage.py startapp polls
Collecting django
  Downloading Django-5.1.1-py3-none-any.whl (8.2 MB)
  ..... 8.2/8.2 MB 42.0 MB/s eta 0:00:00
Collecting sqlparse>=0.3.1
  Downloading sqlparse-0.5.1-py3-none-any.whl (44 kB)
  ..... 44.2/44.2 KB 9.2 MB/s eta 0:00:00
Collecting asgiref<4,>=3.8.1
  Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
Collecting typing-extensions>=4
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: typing-extensions, sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.1.1 sqlparse-0.5.1 typing-extensions-4.12.2

ubuntu@ip-172-31-38-208:/opt/wwc/mysites/lab$ ls -R
.:
lab  manage.py  polls

./lab:
__init__.py  __pycache__  asgi.py  settings.py  urls.py  wsgi.py

./lab/__pycache__:
__init__.cpython-310.pyc  settings.cpython-310.pyc

./polls:
__init__.py  admin.py  apps.py  migrations  models.py  tests.py  views.py

./polls/migrations:
init .py

```

Once the commands are executed, Django creates the following structure for our project:

File Structure

- `lab/`: The project parent directory with starting templates
 - `__init__.py`: Tells that our project is written in Python
 - `settings.py`: Contains project settings such as installed apps, middleware, and database configurations.
 - `urls.py`: Declares URLs for routing HTTP requests.
 - `wsgi.py`: For WSGI-compatible web servers
 - `asgi.py`: For ASGI-compatible servers
- `manage.py`: Contains multiple arguments and serves as a command-line collection to interact with our Django project
- `polls/`: The project app directory with later created `polls` app using `startapp`.
 - `migrations/`: Stores database migrations files.
 - `admin.py`: Contains provided admin interface module
 - `apps.py`: Contains configurations for our app
 - `models.py`: Defines the database schemes and models
 - `tests.py`: Contains unit tests for the app

- `views.py`: Contains functions and classes on how to handle request

The files created by Django provide a boilerplate for developing the project. In the later part, we will work on the poll files to build a simple "Hello, world" page.

[6] Install Nginx

To install the Nginx web server, run the following command:

```
apt install nginx
```

Key Parameters:

- `nginx`: Installs the `nginx` package from the repository.

This command sets up the Nginx web server, which can be used as a reverse proxy for our applications.

```
(myvenv) root@ip-172-31-38-208:/opt/wwc/mysites/lab# apt install nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nginx is already the newest version (1.18.0-6ubuntu14.5).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

[7] Configure nginx

To configure Nginx to work as a reverse proxy for our Django application, we can go to the Nginx configuration folder located at `/etc/nginx/sites-enabled/default` and add the following.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        proxy_pass http://127.0.0.1:8000;
    }
}
```

Key Parameters:

- `listen 80`: Specifies the port Nginx listens on. Here, **80** is the default HTTP port for web traffic. The second `listen` line is for IPv6.
- `proxy_set_header X-Forwarded-Host $host`: Sets the `X-Forwarded-Host` header to the host of the original request. This header preserves the original `Host` header sent by the client.
- `proxy_set_header X-Real-IP $remote_addr`: Sets the `X-Real-IP` header to the real client IP address. This header helps in passing the original client's IP address to the proxied server.

- `proxy_pass http://127.0.0.1:8000;` : Forwards incoming traffic to `http://127.0.0.1:8000`, where our Django application is running. This allows Nginx to act as a reverse proxy, handling requests and passing them to our Django server.

This configuration ensures that all incoming traffic to our server's port **80** will be passed to the Django app running locally on port **8000**.

[8] Restart nginx

To apply our new configuration, we need to restart the Nginx service again, run the following command:

```
service nginx restart
```

Key Parameters:

- `service` : Manages system services.
- `nginx` : Specifies the Nginx service to be managed.
- `restart` : Restarts the Nginx service, stopping smf then starting it again to apply configuration changes.

This command ensures that any changes made to the Nginx configuration will be applied.

[9] Access our EC2 instance

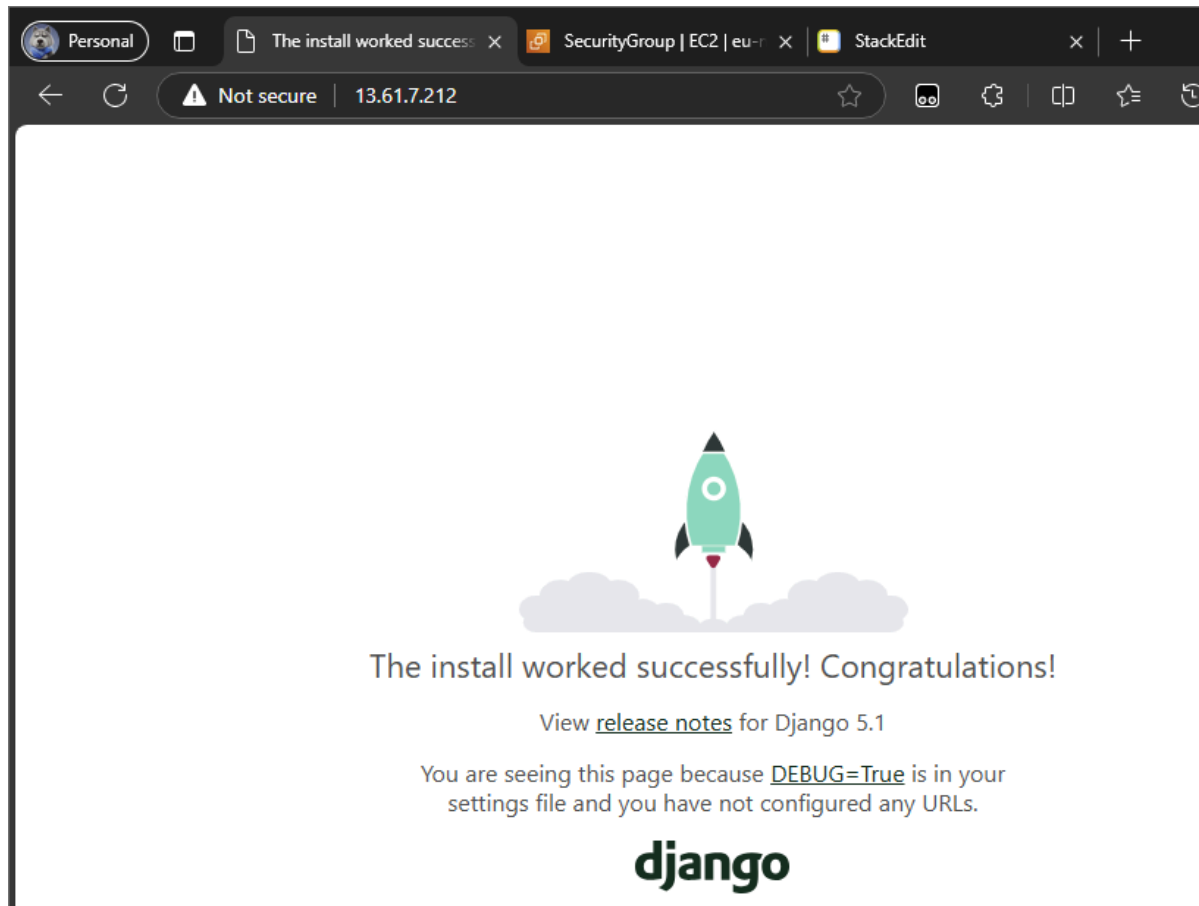
In the app directory `/opt/www/mysites/lab`, run the following command to start our Django application server on port **8000**:

```
python3 manage.py runserver 8000
```

Key Parameters:

- `python3 manage.py` : Runs the script to launch the Django server.
- `runserver 8000` : Specifies the port on which the server will listen for requests. In this case, it's **8000**.

We can now access the web app via `http://13.61.7.212`.



Set up Django App

In this step, we will modify the Django App to display a simple "Hello, World" message when visiting the `/polls` route and also opens the admin interface when visiting `/admin` page.

[1] Edit `polls/view.py`

In `polls/views.py`, we will create a view that returns a simple HTTP response "Hello World":

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world.")
```

Key Parameters:

- `HttpResponse`: Returns a simple HTTP response containing the string `"Hello, world."`.

In `polls/urls.py`, we will map the URL pattern to `view.py` when accessing the index page in the `/polls` path:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Key Parameters:

- `path('', views.index, name='index')`: Routes the root URL of the `polls` app to the `index` view function.

In `lab/urls.py`, add the `polls` app URLs and path for the admin interface:

```
from django.urls import include, path
from django.contrib import admin

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Key Parameters:

- `include('polls.urls')`: Includes URL configurations under the path `polls/`.
- `admin.site.urls`: Sets up the admin interface routings under the path `admin/`.

[2] Restart the web server

Now we can apply the changes and restart the server to see the changes.

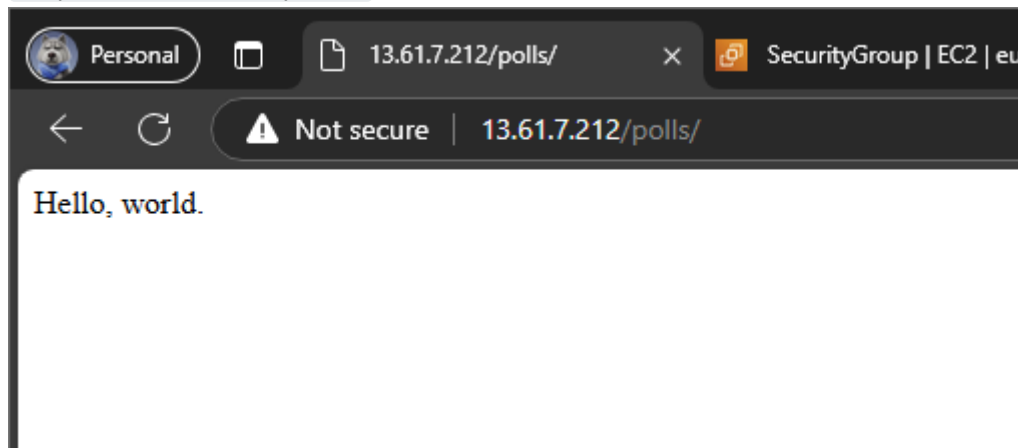
```
python3 manage.py runserver 8000
```

Key Parameters:

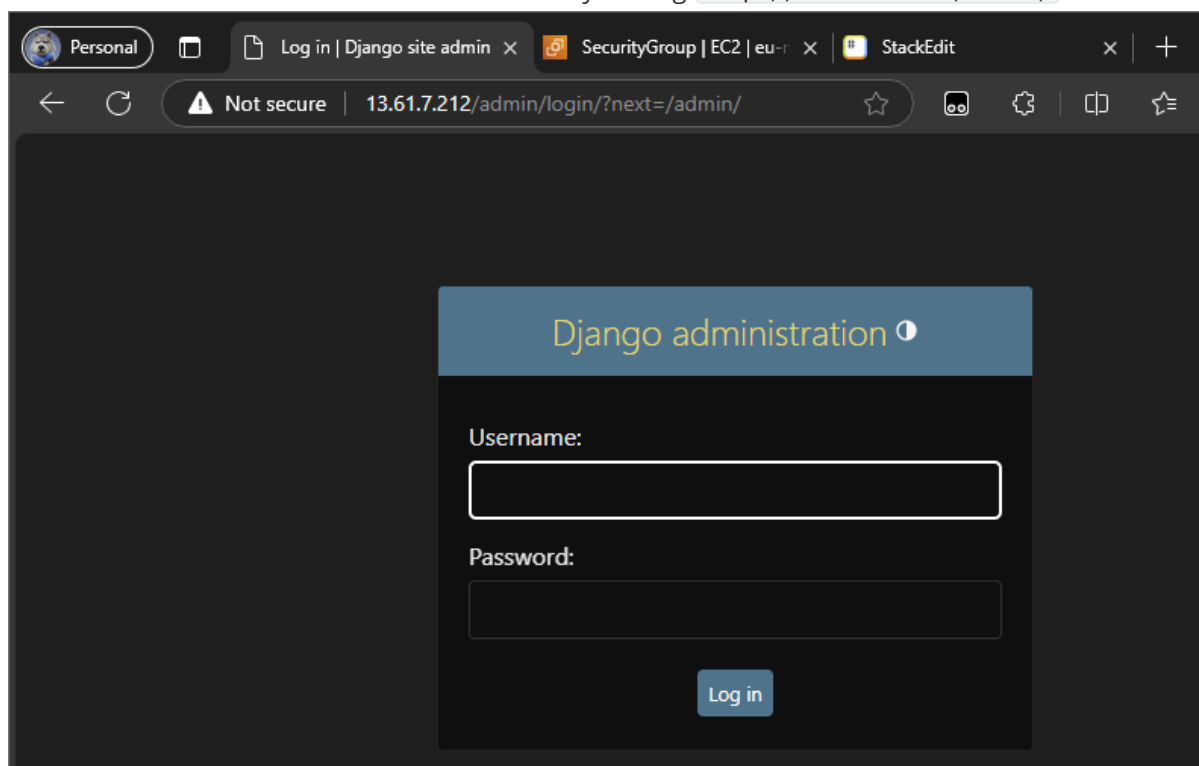
- `python3 manage.py`: Runs `manage.py` to launch our Django server.
- `runserver 8000`: Specifies the port on which the server will listen for requests. In this case, it's **8000**.

[3] Access the EC2 instance

We can access the polls index page with `Hello, world` message by visiting `http://13.61.7.212/polls/`.



We can also access the built-in admin module by visiting `http://13.61.7.212/admin/`



Set up an ALB

[1] Create an application load balancer & Health check

We will use the code in `Tab5` as a start to create the load balancer, the only difference is this time we apply a health check on the `/polls/` path of our hosted website every 30 seconds.

Workflow

1. Initialize Clients and Define Variables:

- Uses **boto3** to initialize EC2 and Elastic Load Balancing (ELB) clients.
- Defines constants for security group, key pair, instance ID, load balancer name, and target group name.

2. Fetch Subnets for the EC2 Instance:

- Retrieves subnets in the `eu-north-1` region for the load balancer.

3. Create Application Load Balancer:

- Uses `elbv2.create_load_balancer()` to create an ALB in the specified subnets, using the security group to allow HTTP traffic.

4. Create Target Group for Health Checks:

- Uses `elbv2.create_target_group()` to create a target group for the EC2 instance.
- Specifies traffic rules, that we use HTTP protocol and use port 80 for forwarding traffic.
- Sets up a DNS health check on the `/polls/` path to be performed every 30 seconds.

5. Register EC2 Instances as Targets:

- Uses `elbv2.register_targets()` to register the EC2 instance to the target group .

6. Create Listener for the Load Balancer:

- Uses `elbv2.create_listener()` to set up a listener on port 80 to forward HTTP requests to the target group .

```
import boto3 as bt
import os

GroupId = 'sg-0ef7af6d7bf260d42'
KeyName = '24188516-key-lab6'
InstanceId = 'i-039c0b853dc14f418'
LoadBalancerName = '24188516-elb'
TargetGroupName = '24188516-tg'

# Initialize EC2 and ELBv2 clients
ec2 = bt.client('ec2', region_name='eu-north-1')
elbv2 = bt.client('elbv2')

subnet_response = ec2.describe_subnets()['Subnets']
Subnets = [subnet['SubnetId'] for subnet in subnet_response]

# 6. Create application load balancer
loadbalancer_response = elbv2.create_load_balancer(
    Name=LoadBalancerName,
    Subnets=Subnets,
    SecurityGroups=[GroupId],
    Scheme='internet-facing',
    Type='application'
)
LoadBalancerArn = loadbalancer_response['LoadBalancers'][0]['LoadBalancerArn']
LoadBalancerDnsName = loadbalancer_response['LoadBalancers'][0]['DNSName']
```

```

# 7. Create target group
vpcId = ec2.describe_vpcs()['Vpcs'][0]['VpcId']
targetgroup_response = elbv2.create_target_group(
    Name=TargetGroupName,
    Protocol='HTTP',
    Port=80,
    VpcId=vpcId,
    TargetType='instance',
    HealthCheckProtocol='HTTP',
    HealthCheckPort='80',
    HealthCheckPath='/polls/',
    HealthCheckIntervalSeconds=30
)
TargetGroupArn = targetgroup_response['TargetGroups'][0]['TargetGroupArn']

# 8. Register instances as targets
elbv2.register_targets(
    TargetGroupArn=TargetGroupArn,
    Targets=[{'Id': InstanceId}]
)

# 9. Create a listener for the load balancer
elbv2.create_listener(
    LoadBalancerArn=LoadBalancerArn,
    Protocol='HTTP',
    Port=80,
    DefaultActions=[{
        'Type': 'forward',
        'TargetGroupArn': TargetGroupArn
    }]
)

# Printouts
print(f"Instance ID: {InstanceId}")
print(f"Load Balancer ARN: {LoadBalancerArn}")
print(f"Target Group ARN: {TargetGroupArn}")
print(f"Load Balancer DNS Name: {LoadBalancerDnsName}")

```

Code Breakdown

1. `elbv2.create_load_balancer()`: Creates an internet-facing application load balancer.
 - `Name`: Specifies the name of the load balancer.
 - `Subnets`: Provides the subnets where our ALB load balancer will distribute traffic.
 - `SecurityGroups`: Assigns the security group to the load balancer for traffic control.
 - `Scheme`: Specifies that the load balancer used for internet-facing.
 - `Type`: Specifies the type of our load balancer as `application`.
2. `elbv2.create_target_group()`: Creates a target group for the load balancer with a health check.

- `Name`: Specifies the name of the target group.
 - `Protocol` and `Port`: Specifies HTTP and port 80 for forwarding requests.
 - `VpcId`: Specifies the ID of the VPC that hosts the EC2 instances.
 - `HealthCheckProtocol` and `HealthCheckPort`: Specifies the HTTP protocol and port 80 for health checks.
 - `HealthCheckPath`: Specifies the path for health checks (`/polls/`).
 - `HealthCheckIntervalSeconds`: Specifies the interval for health checks (every 30 seconds).
3. `elbv2.register_targets()`: Registers our EC2 instance to the target group.
 - `TargetGroupArn`: Specifies the ARN of the target group to register targets.
 - `Targets`: Specifies a list of target instance IDs to be registered.
 4. `elbv2.create_listener()`: Creates a listener to route incoming HTTP traffic on port 80.
 - `LoadBalancerArn`: Specifies the ARN of the load balancer to add listeners.
 - `Protocol` and `Port`: Specifies the HTTP protocol and port 80 for listening.
 - `DefaultActions`: Specifies the actions for forwarding requests to the target group.

After the load balancer is initialized and up in action, we can go to AWS console and see the result of health check.

Targets

Monitoring

Health checks

Attributes

Tags

Registered targets (1) Info

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets target groups with at least 3 healthy targets.

Q Filter targets

| <input type="checkbox"/> | Instance ID | Name | Port | Zone | Health status | Health |
|--------------------------|-------------------------------------|---------------|------|-------------|------------------------|--------|
| <input type="checkbox"/> | i-039c0b853dc14f418 | 24188516-vm-1 | 80 | eu-north-1b | ✔ Healthy | - |

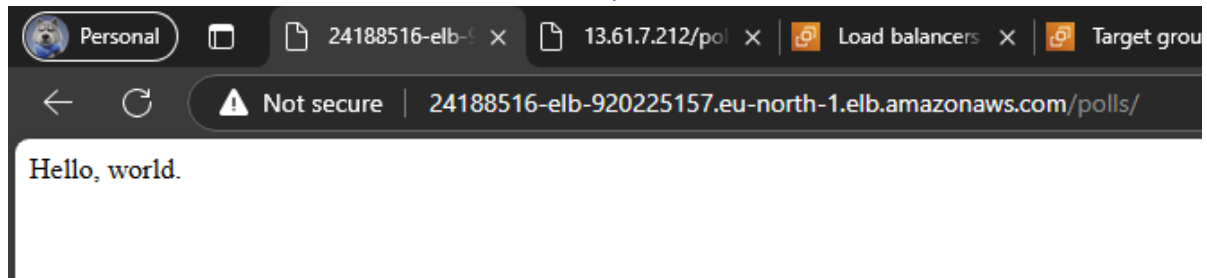
[3] Access the mapped DNS name

We can get the ALB's DNS name from `print(f"Load Balancer DNS Name:`

`{LoadBalancerDnsName}").`

```
liudayubob@Dayu:~/cits5503/Lab6$ python3 createloadbalancer.py
Instance ID: i-039c0b853dc14f418
Load Balancer ARN: arn:aws:elasticloadbalancing:eu-north-1:489389878001:loadbalancer/app/24188516-elb/80d8de1b2725ab55
Target Group ARN: arn:aws:elasticloadbalancing:eu-north-1:489389878001:targetgroup/24188516-tg/03358c87c82ae5d5
Load Balancer DNS Name: 24188516-elb-920225157.eu-north-1.elb.amazonaws.com
```


Now we can access its url with path `/polls/` to see if the mapping works properly: <http://24188516-elb-920225157.eu-north-1.elb.amazonaws.com/polls/>



Lab 7

Set up Fabric Connection

[1] Create EC2 Instance

In the first step, we use our script from **Lab 6** to create a new EC2 instance. We will not elaborate on the code base because it's already covered in previous lab. Run the following command in our local Ubuntu machine:

```
python3 createinstance.py
```

This script automates the creation of the EC2 instance with the required configuration for SSH access and HTTP access. After the instance is successfully created, it will print out the public IP address.

```
liudayubob@Dayu:~/cits5503/lab7$ python3 createinstance.py
16.171.206.115
```

[2] Install Fabric

In this step, we install the **Fabric** package, which is used for automating SSH-based tasks such as managing remote servers.

```
pip install fabric
```

Key Parameters:

- **fabric**: Installs the Fabric package, enabling us to automate remote server management and deployment tasks.

```
Installing collected packages: wrapt, pycparser, invoke, decorator, bcrypt, deprecated, cffi, pynacl, paramiko, fabric
Successfully installed bcrypt-4.2.0 cffi-1.17.1 decorator-5.1.1 deprecated-1.2.14 fabric-3.2.2 invoke-2.2.0 paramiko-3.5
.0 pycparser-2.22 pynacl-1.5.0 wrapt-1.16.0
```

[3] Configure Fabric

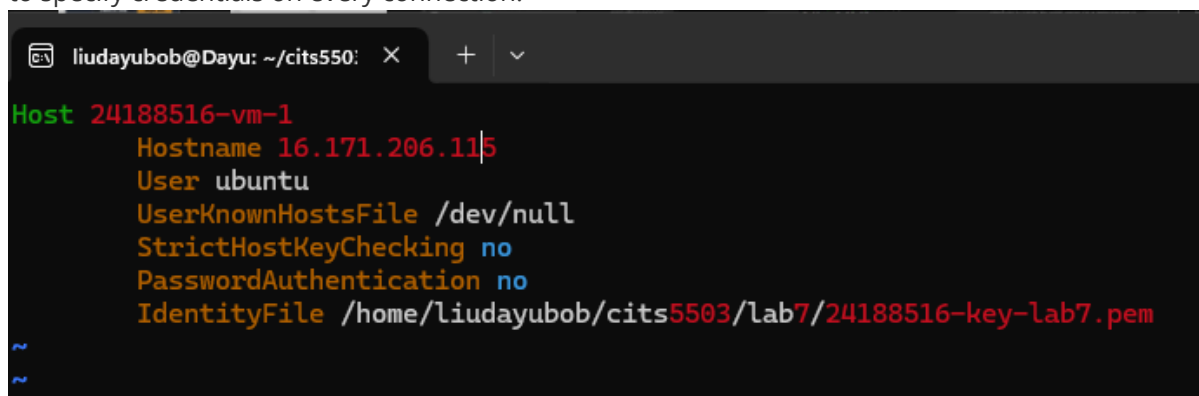
To enable Fabric to connect to our EC2 instance, we need to configure an SSH connection by creating a config file at `~/.ssh/config`. This configuration file stores connection details on the host, IP address, and identity file. Use `vi ~/.ssh/config` to open the config file and edit as following:

```
Host 24188516-vm-1
    Hostname 16.171.206.115
    User ubuntu
    UserKnownHostsFile /dev/null
    StrictHostKeyChecking no
    PasswordAuthentication no
    IdentityFile /home/liudayubob/cits5503/lab7/24188516-key-lab7.pem
```

Key Parameters:

1. `Host`: Defines the alias for our EC2 instance, which will be used when calling the Fabric connection function.
2. `Hostname`: Specifies the public IP address (in this case, `16.171.206.115`) of our EC2 instance.
3. `User ubuntu`: Specifies the default username for EC2 instances based on the Ubuntu AMI image.
4. `IdentityFile`: Specifies the path to our private key file (generated during instance creation) for authentication.
5. `UserKnownHostsFile /dev/null` and `StrictHostKeyChecking no`: Declares that we disable SSH host key checking, preventing the need for manual approval when connecting.

By creating a host configuration, we can use Fabric to connect to the EC2 instance without needing to specify credentials on every connection.



```
liudayubob@Dayu: ~/cits550: X + v
Host 24188516-vm-1
    Hostname 16.171.206.115
    User ubuntu
    UserKnownHostsFile /dev/null
    StrictHostKeyChecking no
    PasswordAuthentication no
    IdentityFile /home/liudayubob/cits5503/lab7/24188516-key-lab7.pem
~
~
```

[4] Test Fabric Connection

We will use the following Fabric code to establish a connection to the EC2 instance. Fabric looks up the host file and uses the connection configuration for `24188516-vm-1`. After establishing the connection, we will run a simple command to verify it.

The command `c.run('uname -s')` will return "Linux" as output, confirming that the connection is successful and commands can be executed on the instance.

```
python3
>>> from fabric import Connection
>>> c = Connection('24188516-vm-1')
>>> result = c.run('uname -s')
Linux
```

Key Parameters:

- `Connection()`: Uses the SSH configuration from Host file to connect to the EC2 instance using the alias `24188516-vm-1`.
- `c.run('uname -s')`: Runs the `uname -s` command, confirming the operating system on the remote instance is Linux.

```
liudayubob@Dayu:~/cits5503/lab7$ python3
Python 3.10.12 (main, Jul 29 2024, 16:56:48) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from fabric import Connection
>>> c = Connection('24188516-vm-1')
>>> result = c.run('uname -s')
Linux
```

Automation for creating Django App

In this section, we will use `Fabric` to automate the process of setting up a virtual environment, configuring Nginx, and creating a Django app within the EC2 instance.

The commands from **Lab 6** will be converted to Fabric's `c.run()` for regular commands and `c.sudo()` for commands requiring admin privileges.

Additionally, file editing will be handled using `echo`. We will use file I/O to write Nginx configuration to avoid issues with `$` placeholders.

Due to the fact that each `c.run()` command is runned isolately, to persist the sourced virtual environment, we will re-source the environment before running further commands.

Workflow:

1. Install Packages:

- Update and upgrade system packages.
- Install the Python virtual environment package (`python3-venv`).
- Install Nginx web server.

2. Set Up Virtual Environment:

- Create a project directory and assign necessary permissions.
- Set up a virtual environment within the project directory and install Django.

3. Create Django Project and App:

- Start a new Django project and app (`polls`) inside the virtual environment.
- Modify the views, URLs, and settings to display "Hello, world" from the `polls` app.

4. Configure Nginx Server:

- Make a Nginx configuration file to act as a reverse proxy, forwarding traffic from port 80 to the Django app running on port 8000.

5. Run Django Server:

- Start the Django server in the background, checking that our app is accessible on port 8000.

Here is the script that automates these steps:

```
from fabric import Connection

EC2_INSTANCE_NAME = '24188516-vm-1'
PROJECT_DIR = '/opt/www/mysites/lab'

def install_prerequisites(c):
    # Update and upgrade system packages
```

```

c.sudo('apt-get update -y')
c.sudo('apt-get upgrade -y')
c.sudo('apt-get install python3-venv -y')
c.sudo('apt install nginx -y')

def set_virtual_env(c):
    # Create project directory and navigate to it
    c.sudo(f'mkdir -p {PROJECT_DIR}')
    # Grant permissions to user
    c.sudo(f'chown -R ubuntu:ubuntu {PROJECT_DIR}')
    # Create env and source env
    c.run(f'cd {PROJECT_DIR} && python3 -m venv myenv')
    c.run(f'cd {PROJECT_DIR} && source myenv/bin/activate && pip install
django')

def setup_django_app(c):
    # Need to cd and source env again
    c.run(f'cd {PROJECT_DIR} && source myenv/bin/activate && django-admin
startproject lab .')
    c.run(f'cd {PROJECT_DIR} && source myenv/bin/activate && python3 manage.py
startapp polls')

    # Polls app
    c.run(f'echo "from django.http import HttpResponse" >
{PROJECT_DIR}/polls/views.py')
    c.run(f'echo "def index(request): return HttpResponse(\'Hello, world.\')" >>
{PROJECT_DIR}/polls/views.py')

    # Admin app and routing
    c.run(f'echo "from django.urls import path\nfrom . import views\nurlpatterns
= [path(\'\', views.index, name=\'index\')]" > {PROJECT_DIR}/polls/urls.py')
    c.run(f'echo "from django.urls import include, path\nfrom django.contrib
import admin\nurlpatterns = [path(\'polls/\', include(\'polls.urls\')),
path(\'admin/\', admin.site.urls)]" > {PROJECT_DIR}/lab/urls.py')

def configure_nginx(c):
    nginx_config = '''
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        proxy_pass http://127.0.0.1:8000;
    }
}
'''

    # Write the nginx config locally and upload ($placeholder were messed up with
echo)
    with open("nginx_temp.conf", "w") as f:
        f.write(nginx_config)
    c.put("nginx_temp.conf", "/tmp/nginx_temp.conf")
    c.sudo('mv /tmp/nginx_temp.conf /etc/nginx/sites-enabled/default')

```

```

# Restart Nginx to apply changes
c.sudo('service nginx restart')

def run_django_server(c):
    # Start Django development server in the background
    c.run(f'cd {PROJECT_DIR} && source myvenv/bin/activate && python3 manage.py
runserver 8000')

if __name__ == "__main__":
    fabric = Connection(EC2_INSTANCE_NAME)

    install_prerequisites(fabric)
    set_virtual_env(fabric)
    setup_django_app(fabric)
    configure_nginx(fabric)
    run_django_server(fabric)

```

Code Breakdown:

1. `install_prerequisites()`:

- `apt-get update` and `apt-get upgrade`: Updates and upgrades system packages.
- `apt-get install python3-venv`: Installs Python virtual environment package.
- `apt install nginx`: Installs the Nginx server for handling HTTP traffic.

2. `set_virtual_env()`:

- `mkdir -p`: Creates the project directory to store our virtual environment settings and django app.
- `python3 -m venv myvenv`: Creates an isolated virtual environment for our Django app.
- `pip install django`: Installs Django package in `myvenv` virtual environment.

3. `setup_django_app()`:

- `django-admin startproject lab`: Creates the Django project named `lab`.
- `python3 manage.py startapp polls`: Creates the `polls` app.
- `echo xxx`: Modifies the `views.py`, `urls.py`, and `lab/urls.py` with proper HTML contents and routings for displaying "Hello, world" page and the admin app.

4. `configure_nginx()`:

- `nginx_config = {}`: Modifies a configuration file to forward requests from port **80** to Django on port **8000**.
- `service nginx restart`: Restarts Nginx service to apply the changes.

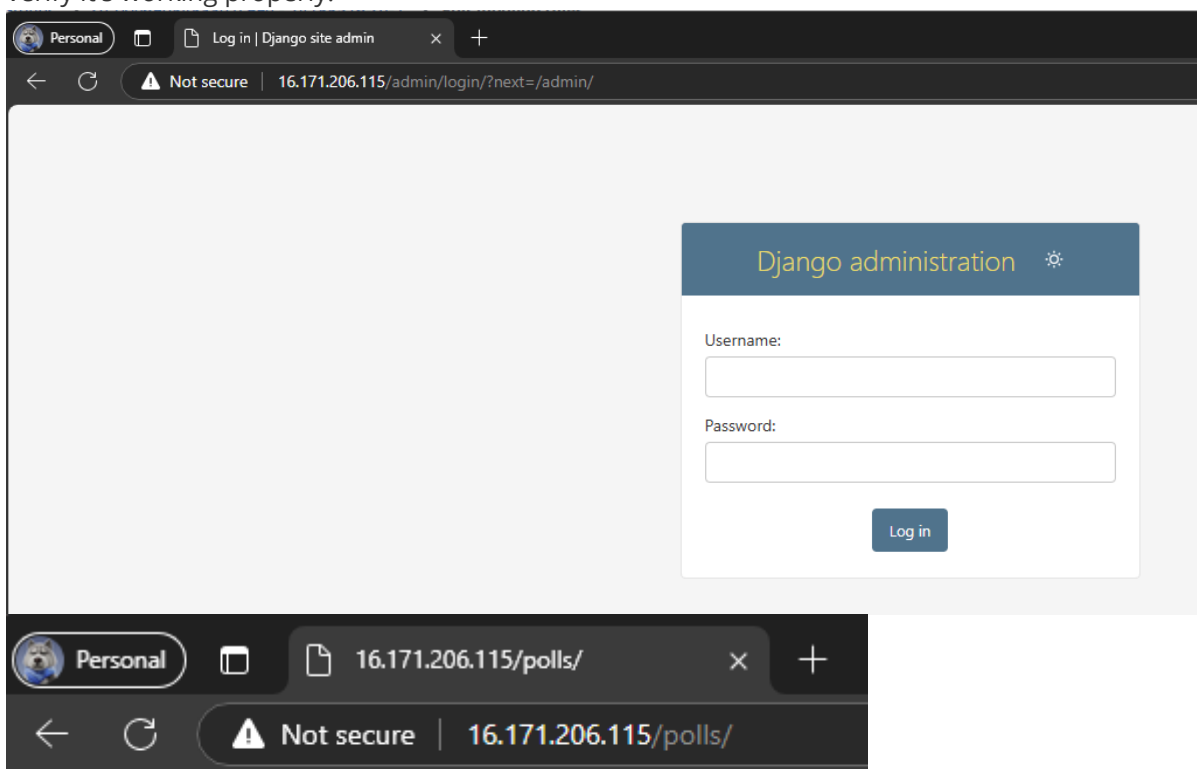
5. `run_django_server()`:

- `manage.py runserver`: Runs the Django server on port 8000.

```
liudayubob@Dayu: ~/cits550: x + v
Downloading Django-5.1.2-py3-none-any.whl (8.3 MB)
8.3/8.3 MB 49.3 MB/s eta 0:00:00
Collecting sqlparse>=0.3.1
Downloading sqlparse-0.5.1-py3-none-any.whl (44 kB)
44.2/44.2 KB 11.2 MB/s eta 0:00:00
Collecting asgiref<4,>=3.8.1
Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
Collecting typing-extensions>=4
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: typing-extensions, sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.1.2 sqlparse-0.5.1 typing-extensions-4.12.2
Watching for file changes with StatReloader

[14/Oct/2024 05:22:47] "GET /polls/ HTTP/1.1" 200 13
[14/Oct/2024 05:24:24] "GET /admin HTTP/1.0" 301 0
[14/Oct/2024 05:24:24] "GET /admin/ HTTP/1.0" 302 0
[14/Oct/2024 05:24:24] "GET /admin/login/?next=/admin/ HTTP/1.0" 200 4160
[14/Oct/2024 05:24:25] "GET /static/admin/css/dark_mode.css HTTP/1.0" 200 2804
[14/Oct/2024 05:24:25] "GET /static/admin/css/base.css HTTP/1.0" 200 22092
[14/Oct/2024 05:24:25] "GET /static/admin/css/login.css HTTP/1.0" 200 951
[14/Oct/2024 05:24:25] "GET /static/admin/js/theme.js HTTP/1.0" 200 1653
[14/Oct/2024 05:24:25] "GET /static/admin/css/nav_sidebar.css HTTP/1.0" 200 2810
[14/Oct/2024 05:24:25] "GET /static/admin/css/responsive.css HTTP/1.0" 200 17972
[14/Oct/2024 05:24:25] "GET /static/admin/js/nav_sidebar.js HTTP/1.0" 200 3063
Not Found: /favicon.ico
[14/Oct/2024 05:24:25] "GET /favicon.ico HTTP/1.0" 404 2354
```

Now with the Django App started and the server is online, we can go to `/admin/` and `/polls` to verify it's working properly.



Hello, world.

Lab 8

Install and Run Jupyter Notebook

In this step, we will install Jupyter Notebooks and use it for AI training. Jupyter Notebooks provide an interactive environment to run Python code on the go.

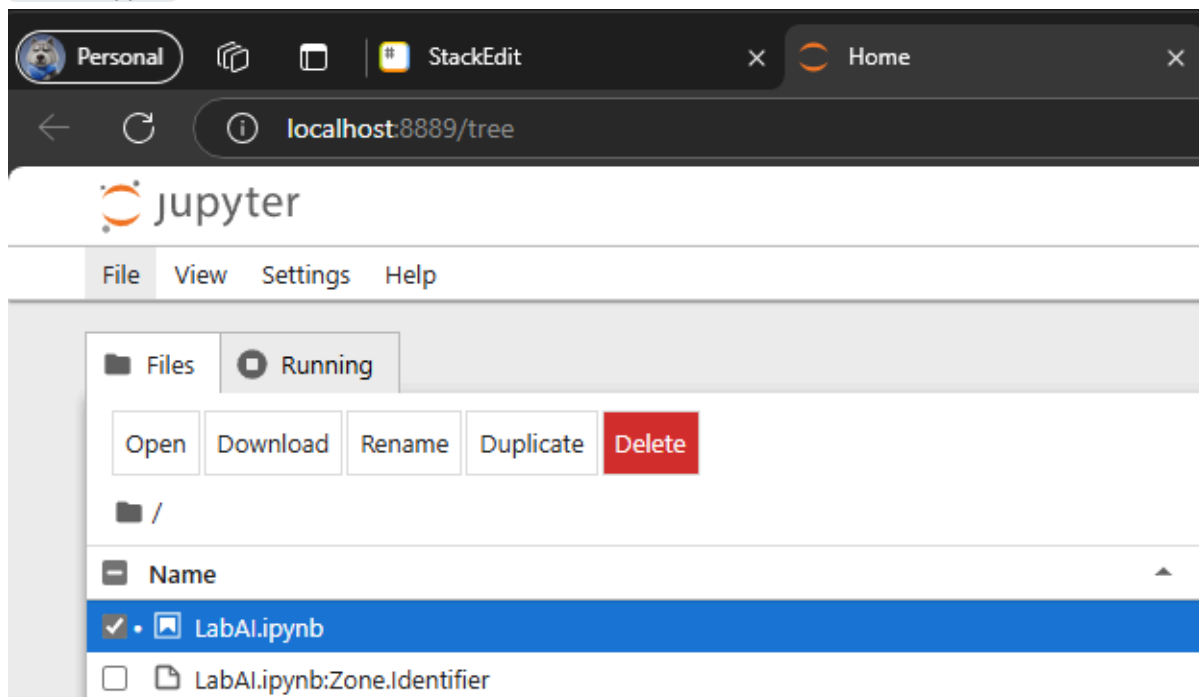
```
pip install notebook
jupyter notebook
```

Key Parameters

- `pip install notebook`: Installs the Jupyter Notebook package
- `jupyter notebook`: Starts the Jupyter Notebook server and opens a web interface in our browser. It launches at `http://127.0.0.1:8889` in our case, because the default port is already in use.

```
liudayubob@Dayu:~/cits5503/lab8$ jupyter notebook
[I 2024-10-14 14:59:47.159 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-10-14 14:59:47.162 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-10-14 14:59:47.165 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-10-14 14:59:47.167 ServerApp] notebook | extension was successfully linked.
[I 2024-10-14 14:59:47.398 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-10-14 14:59:47.405 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-10-14 14:59:47.406 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-10-14 14:59:47.407 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-10-14 14:59:47.409 LabApp] JupyterLab extension loaded from /home/liudayubob/.local/lib/python3.10/site-packages/jupyterlab
[I 2024-10-14 14:59:47.410 LabApp] JupyterLab application directory is /home/liudayubob/.local/share/jupyter/lab
[I 2024-10-14 14:59:47.410 LabApp] Extension Manager is 'pypi'.
[I 2024-10-14 14:59:47.447 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-10-14 14:59:47.449 ServerApp] notebook | extension was successfully loaded.
[I 2024-10-14 14:59:47.449 ServerApp] The port 8888 is already in use, trying another port.
[I 2024-10-14 14:59:47.450 ServerApp] Serving notebooks from local directory: /home/liudayubob/cits5503/lab8
[I 2024-10-14 14:59:47.450 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2024-10-14 14:59:47.450 ServerApp] http://localhost:8889/tree?token=ee4324b96e5533e319b11d8167973e2400f3e1aad2a0f3d5
[I 2024-10-14 14:59:47.450 ServerApp] http://127.0.0.1:8889/tree?token=ee4324b96e5533e319b11d8167973e2400f3e1aad2a0f3d5
```

After running the above commands, we can see that the Jupyter server has launched, and the `LabAI.ipynb` notebook file is visible on the file server interface.



Install ipykernel

In this step, we will install the `ipykernel` package, this is the kernel package for Python coding in Jupyter Notebooks.

```
pip install ipykernel
```

Key Parameters

- `ipykernel`: This package allows Jupyter to communicate with the Python interpreter.

```
Python 3.12.12 -> ipykernel (1.16.0)
Requirement already satisfied: asttokens>=2.1.0 in /home/liudayubob/.local/lib/python3.10/site-packages (from stack-data->ipython>=7.23.1->ipykernel) (2.4.1)
Requirement already satisfied: pure-eval in /home/liudayubob/.local/lib/python3.10/site-packages (from stack-data->ipython>=7.23.1->ipykernel) (0.2.3)
Requirement already satisfied: executing>=1.2.0 in /home/liudayubob/.local/lib/python3.10/site-packages (from stack-data->ipython>=7.23.1->ipykernel) (2.1.0)
```

Necessary Changes

In this section, we will modify the provided code `LabAI.ipynb` within the Jupyter notebook to make it work in our environment.

[1] Modify Region, Student ID, and Bucket Name

First, we need to change the constant variables for the AWS region, our student ID, and the name of the S3 bucket accordingly.

```
region = 'eu-north-1' # use the region us are mapped to
student_id = '24188516' # use our student ID
bucket = '24188516-lab8' # use <studentid-lab8> as our bucket name
```

[2] Create an S3 Bucket

We will also create an S3 bucket to store the training and testing datasets. Here, we use the `boto3` library to create a bucket in the specified region `eu-north-1` and add an object with the prefix for our folder destination.

```
s3 = boto3.client('s3', region_name=region)
bucket_config = {'LocationConstraint': region}
s3.create_bucket(Bucket=bucket, CreateBucketConfiguration=bucket_config) #
Create the bucket in our region
s3.put_object(Bucket=bucket, Key=f"{prefix}/") # Create a folder object with the
prefix
```

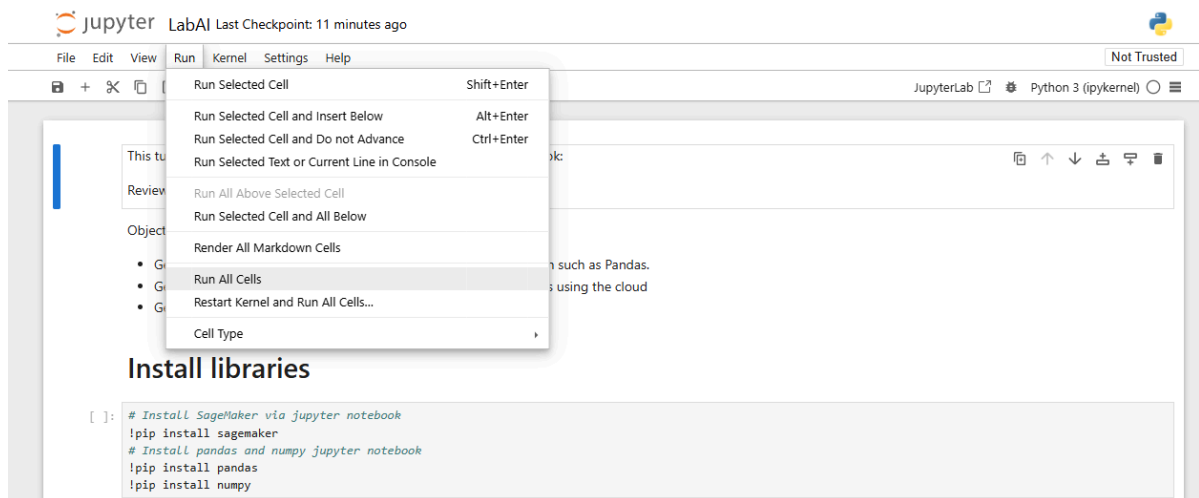
[3] Convert Non-numerical Values (True/False to 1/0)

Because our tuning job can't handle non-numerical values, after reading the dataset and before splitting it into training/testing set, we will traverse the model_data and convert all datas to numeric ones.

```
# Change True/False to 1/0
model_data = model_data.replace({True: 1, False: 0})
```

[4] Run Jupyter Notebook

After making the necessary changes to the notebook, we can execute the notebook by navigating to the **Run** menu and selecting **Run All Cells/ Run Selected Cells**.



Dataset Q&A

Read the dataset into a Pandas data frame and answer the following two questions:

- #1. Which variables in the dataset are categorical? Give at least four variables.
- #2. Which variables in the dataset are numerical? Give at least four variables.

```
After inspecting the dataframe, we can get the following conclusions:
#1: Categorical variables: job, marital, education, contact
#2: Numerical variables: age, duration, nr.employed, euribor3m
```

AI Training

In `LabAI.ipynb`, we will set up a tuning job using Amazon SageMaker. Steps involve installing the necessary libraries, preparing data, and running a hyperparameter tuning job using XGBoost. The final objective is to use SageMaker for training a model on the provided Bank Marketing dataset.

[1] Install Required Libraries

To begin with, we need to install several essential libraries such as SageMaker, Pandas, and Numpy for machine learning and data processing.

1. Install SageMaker:

- SageMaker is used to create and manage training jobs, models in AWS.

2. Install Pandas and Numpy:

- Pandas is used for data manipulation, while Numpy is used for numerical operations.

```
# Install SageMaker via Jupyter Notebook
!pip install sagemaker

# Install Pandas and Numpy via Jupyter Notebook
!pip install pandas
!pip install numpy
```

[2] Prepare SageMaker Session and S3 Bucket

In the next step, we need to set up a SageMaker session, IAM role, and S3 bucket to store the training data.

Workflow

1. Set up SageMaker Session:

- Initialize the SageMaker and IAM clients, specify the region, and get the ARN of the SageMaker role.

2. Create an S3 Bucket:

- Create an S3 bucket to store training/validation/testing data.

3. Download Dataset:

- Download and unzip the Bank Marketing dataset from UCI ML repository.

```
import sagemaker
import boto3
import numpy as np
import pandas as pd

region = 'eu-north-1' # Set our AWS region
smclient = boto3.Session(region_name=region).client("sagemaker")
iam = boto3.client('iam', region_name=region)
sagemaker_role = iam.get_role(RoleName='SageMakerRole')['Role']['Arn']
student_id = "24188516" # Use our student ID
bucket = '24188516-lab8' # Use our student ID for bucket name
prefix = f"sagemaker/{student_id}-hpo-xgboost-dm"

# Create an S3 bucket and folder
s3 = boto3.client('s3', region_name=region)
bucket_config = {'LocationConstraint': region}
```

```
s3.create_bucket(Bucket=bucket, CreateBucketConfiguration=bucket_config) #
Create bucket
s3.put_object(Bucket=bucket, Key=f"{prefix}/") # Create a folder in S3

# Download dataset
!wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-
additional.zip
!unzip -o bank-additional.zip
```

Code Breakdown:

- `sagemaker.Session()`: Initializes a SageMaker session to interact with AWS SageMaker services.
- `boto3.client('s3')`: Creates an S3 client to interact with S3 services.
- `s3.create_bucket()`: Creates an S3 bucket in the specified region.
- `s3.put_object()`: Creates a folder inside the S3 bucket for storing data.
- `!wget` and `!unzip`: Downloads and unzips the dataset to our local folder.

We can see that our folder has been created in the S3 bucket.

[Amazon S3](#) > [Buckets](#) > [24188516-lab8](#) > [sagemaker/](#)

sagemaker/


Objects

Properties

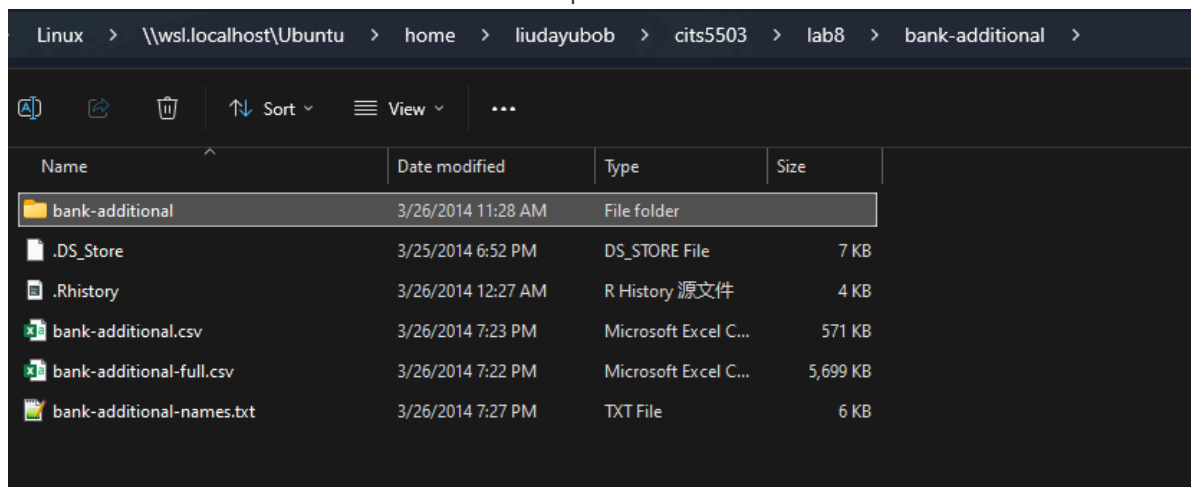
Objects (1) [Info](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all ob

Find objects by prefix

| <input type="checkbox"/> | Name | Type |
|--------------------------|--|--------|
| <input type="checkbox"/> |  24188516-hpo-xgboost-dm/ | Folder |

The dataset has been downloaded and uncompressed.



| Name | Date modified | Type | Size |
|---------------------------|--------------------|----------------------|----------|
| bank-additional | 3/26/2014 11:28 AM | File folder | |
| .DS_Store | 3/25/2014 6:52 PM | DS_STORE File | 7 KB |
| .Rhistory | 3/26/2014 12:27 AM | R History 源文件 | 4 KB |
| bank-additional.csv | 3/26/2014 7:23 PM | Microsoft Excel C... | 571 KB |
| bank-additional-full.csv | 3/26/2014 7:22 PM | Microsoft Excel C... | 5,699 KB |
| bank-additional-names.txt | 3/26/2014 7:27 PM | TXT File | 6 KB |

[3] Data Preparation and Processing

We will prepare the dataset for training by converting categorical data to binary indicators and splitting the data into training, validation, and test sets.

Workflow

1. Load and Process Data:

- Load the dataset into Pandas and create new indicator columns for specific variables.

2. Convert to Dummy Variables:

- Use `pd.get_dummies()` and convert categorical variables into sets of indicators.

3. Remove Unnecessary Columns:

- Drop economic variables and duration from the dataset to avoid bias in future predictions.

4. Fix Non-Numeric Data:

- Replace `True/False` values with `1/0` to avoid non-numeric errors in SageMaker.

5. Split Train/Validation/Test Data:

- Split the data into training set (70%), validation set (20%), and test set (10%).

6. Save Split Datasets as CSV Files:

- Save each split as a CSV file, removing headers and specifying the first column to be the target variable.

7. Upload the Datasets to S3:

- Upload the processed datasets to S3 to allow SageMaker to access them during model training.

```
# Load dataset into Pandas
data = pd.read_csv("./bank-additional/bank-additional-full.csv", sep=";")

# Add new indicator columns
data["no_previous_contact"] = np.where(data["pdays"] == 999, 1, 0)
data["not_working"] = np.where(np.in1d(data["job"], ["student", "retired", "unemployed"]), 1, 0)
```

```

# Convert categorical variables to dummy variables
model_data = pd.get_dummies(data)

# Remove unnecessary columns
model_data = model_data.drop(
    ["duration", "emp.var.rate", "cons.price.idx", "cons.conf.idx", "euribor3m",
    "nr.employed"],
    axis=1,
)

# Replace True/False with 1/0
model_data = model_data.replace({True: 1, False: 0})

# Split data into training, validation, and test datasets
train_data, validation_data, test_data = np.split(
    model_data.sample(frac=1, random_state=1729),
    [int(0.7 * len(model_data)), int(0.9 * len(model_data))],
)

# Save datasets as CSV files
pd.concat([train_data["y_yes"], train_data.drop(["y_no", "y_yes"], axis=1)],
axis=1).to_csv(
    "train.csv", index=False, header=False
)
pd.concat([validation_data["y_yes"], validation_data.drop(["y_no", "y_yes"],
axis=1)], axis=1).to_csv(
    "validation.csv", index=False, header=False
)
pd.concat([test_data["y_yes"], test_data.drop(["y_no", "y_yes"], axis=1)],
axis=1).to_csv(
    "test.csv", index=False, header=False
)

# Upload the datasets to S3
boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "train/train.csv")
).upload_file("train.csv")
boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "validation/validation.csv")
).upload_file("validation.csv")

```

Code Breakdown:

- `pd.read_csv()`:
 - Reads the dataset into a Pandas dataframe from the given file.
- `np.where()`:
 - Adds indicator columns based on specific conditions (e.g., checking if a customer was previously contacted).
- `pd.get_dummies()`:

- Converts categorical variables into dummy variables, so we can fit them for training models.
- `model_data.drop()`:
 - Removes unnecessary columns that could introduce bias or noise into the model.
- `model_data.replace()`:
 - Replaces `True/False` values with `1/0` to avoid non-numeric errors during SageMaker training.
- `np.split()`:
 - Splits the dataset into training, validation, and test sets with a specified proportions in a random manner.
- `pd.concat()`:
 - Combines the target column (`y_yes`) with the remaining feature columns and saves them as CSV files for each split.
- `os.path.join()`:
 - Combines the bucket name and the prefix to get the correct path where the files should be uploaded to S3.
- `object.upload_file()`:
 - Uploads the prepared training and validation CSV datasets to the S3 bucket

We can see that the dataset have been created and uploaded to our designated directories in the S3 bucket.

[Amazon S3](#) > [Buckets](#) > [24188516-lab8](#) > [sagemaker/](#) > [24188516-hpo-xgboost-dm/](#) > **train/**

train/

Objects

Properties

Objects (1) [Info](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket.

| <input type="checkbox"/> | Name | Type |
|--------------------------|---------------------------|------|
| <input type="checkbox"/> | train.csv | csv |

[Amazon S3](#) > [Buckets](#) > [24188516-lab8](#) > [sagemaker/](#) > [24188516-hpo-xgboost-dm/](#) > **validation/**

validation/

Objects

Properties

Objects (1) [Info](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket.

| <input type="checkbox"/> | Name | Type |
|--------------------------|--------------------------------|------|
| <input type="checkbox"/> | validation.csv | csv |

[4] Set Up Hyperparameter Tuning Job

Next, we'll configure and launch a hyperparameter tuning job using SageMaker's XGBoost algorithm.

Workflow

1. Configure Hyperparameters:

- Define a set of hyperparameter ranges that will be tuned using SageMaker. These include parameters like `eta` (learning rate), `min_child_weight`, `alpha` and `max_depth`.
- Set the resource limits and specify that the objective is to maximize the area under the curve (AUC) for validation data.

2. Specify Training Job:

- Specify the training algorithm (XGBoost), input data (from S3), and the resource attributes (instance type, count, and storage).
- Set a stopping condition to ensure the training job doesn't run indefinitely and define static hyperparameters like `eval_metric` and `objective`.

3. Launch Hyperparameter Tuning:

- Use the SageMaker APIs to launch the hyperparameter tuning job, train multiple models and return the best one based on the defined metric (`validation:auc`).

```
from time import gmtime, strftime, sleep
from sagemaker.image_uris import retrieve

# Set up a unique tuning job name
tuning_job_name = f"{student_id}-xgboost-tuningjob-01"
print(tuning_job_name)

# Hyperparameter ranges for tuning
tuning_job_config = {
    "ParameterRanges": {
        "ContinuousParameterRanges": [
            {"Name": "eta", "MinValue": "0", "MaxValue": "1"},
            {"Name": "min_child_weight", "MinValue": "1", "MaxValue": "10"},
            {"Name": "alpha", "MinValue": "0", "MaxValue": "2"},
        ],
        "IntegerParameterRanges": [{"Name": "max_depth", "MinValue": "1",
"MaxValue": "10"}],
    },
    "ResourceLimits": {"MaxNumberOfTrainingJobs": 2, "MaxParallelTrainingJobs":
2},
    "Strategy": "Bayesian",
    "HyperParameterTuningJobObjective": {"MetricName": "validation:auc", "Type":
"Maximize"},
}

# Specify XGBoost algorithm for training
training_image = retrieve(framework="xgboost", region=region, version="latest")
s3_input_train = f"s3://{bucket}/{prefix}/train"
s3_input_validation = f"s3://{bucket}/{prefix}/validation/"

training_job_definition = {
    "AlgorithmSpecification": {"TrainingImage": training_image,
"TrainingInputMode": "File"},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {"S3DataSource": {"S3Uri": s3_input_train,
"S3DataType": "S3Prefix"}},
        },
        {
            "ChannelName": "validation",
            "DataSource": {"S3DataSource": {"S3Uri": s3_input_validation,
"S3DataType": "S3Prefix"}},
        },
    ],
    "OutputDataConfig": {"S3OutputPath": f"s3://{bucket}/{prefix}/output"},
}
```

```

    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.m5.xlarge",
    "VolumeSizeInGB": 10},
    "RoleArn": sagemaker_role,
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
    "StaticHyperParameters": {"eval_metric": "auc", "num_round": "1",
    "objective": "binary:logistic"}, }

# Launch the hyperparameter tuning jobsmlclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)

```

Code Breakdown:

1. `tuning_job_config = {}`:

- `ParameterRanges`: Specifies the range of hyperparameters to be optimized.
- `ResourceLimits`: Specifies the number of training jobs to 2, both for maximum jobs and parallel jobs.
- `Strategy`: Specifies that SageMaker will use Bayesian optimization to explore the hyperparameter space.
- `HyperParameterTuningJobObjective`: Specifies the objective of our tuning job, to maximize the area under the ROC curve (AUC).

2. `retrieve()`:

- Retrieves the latest version of the XGBoost image from SageMaker to ensure most accurate training.

3. `training_job_definition = {}`:

- `AlgorithmSpecification`: Specifies the XGBoost algorithm and the input mode (File-based input).
- `InputDataConfig`: Specifies the input paths where the training and validation data are stored.
- `OutputDataConfig`: Specifies the output paths where the training outputs will be saved.
- `ResourceConfig`: Specifies the compute resources for the training job, including the instance type, instance count, and storage.
- `StoppingCondition`: Specifies the maximum run time allowed of the training job to 12 hours (43,200 seconds).
- `StaticHyperParameters`: Specifies fixed hyperparameters that are not tuned, such as `objective` (binary classification) and `eval_metric` (AUC).

4. `create_hyper_parameter_tuning_job()`:

- Launches the tuning job using the previous defined configuration.

[5] Monitor Hyperparameter Tuning Job

After launching the hyperparameter tuning job, we can monitor its progress in the AWS console.

Amazon SageMaker > Training jobs

Training jobs [Info](#)

Actions ▾

Create training job

Q

Search training jobs

<

1

...

>

| | Name ▾ | Creation time ▾ | Duration | Job status ▾ | Warm pool status | Time left |
|-------------|--|------------------------|-----------|------------------------|------------------------|-----------|
| <div></div> | 24188516-xgboost-tuningjob-03-002-c7134e6f | 10/14/2024, 4:11:40 PM | 2 minutes | <div>✔ Completed</div> | <div>✔ Available</div> | - |
| <div></div> | 24188516-xgboost-tuningjob-03-001-8741f905 | 10/14/2024, 4:11:38 PM | 2 minutes | <div>✔ Completed</div> | <div>✔ Available</div> | - |

Lab 9

AWS Comprehend

In this task, we will leverage AWS Comprehend to analyze text for **language detection**, **sentiment detection**, **entity detection**, **key phrase detection** and **syntax detection**.

[1] Client Setup & Language Detection

We'll start by using AWS Comprehend's `detect_dominant_language` method to identify the language in given texts and display the confidence level from the prediction.

Workflow

1. Set Up AWS Comprehend Client:

- Initialize the AWS Comprehend client using `boto3` with a specific region (`ap-southeast-2` in this case).

2. Detect Dominant Language:

- Use `client.detect_dominant_language` to detect the language for each piece of text.
- Extract the first probable language from the response.

3. Map Language Codes to Language Names:

- Use a dictionary to map language codes (such as `'en'`, `'fr'`, `'es'`, `'it'`) to their corresponding language names (English, French, Spanish, Italian).

4. Extract Confidence and Print Results:

- Extract the confidence level in target language and print the language name along with its confidence level.

```
import boto3

# AWS Comprehend client
REGION = "ap-southeast-2" # Specify the AWS region
client = boto3.client('comprehend', region_name=REGION)

def detect_language(text):
    # Detect the dominant language in the provided text
    response = client.detect_dominant_language(Text=text)
    lang = response['Languages'][0] # We only use the first detected language

    # Language code for mapping
    language_map = {
        'en': 'English',
        'es': 'Spanish',
        'fr': 'French',
        'it': 'Italian',
    }

    # Convert results to message
    lang_code = lang['LanguageCode']
    confidence = round(lang['Score'] * 100, 2)
```

```

    language_name = language_map.get(lang_code, lang_code) # Get the language
    name from the map
    print(f"{language_name} detected with {confidence}% confidence")

# Test with various texts in different languages
texts = [
    "The French Revolution was a period of social and political upheaval in
    France.",
    "El Quijote es la obra más conocida de Miguel de Cervantes Saavedra.",
    "Moi je n'étais rien Et voilà qu'aujourd'hui Je suis le gardien Du sommeil de
    ses nuits.",
    "L'amor che move il sole e l'altre stelle."
]

# Loop through the texts and detect the language
for text in texts:
    detect_language(text)

```

Code Breakdown

- `client = boto3.client()`:
 - Creates an AWS Comprehend client in the specified region (`ap-southeast-2`), allowing us to make requests to the AWS Comprehend service.
- `response = client.detect_dominant_language()`:
 - Uses the `detect_dominant_language` API to detect the dominant language in the given text.
- `lang = response['Languages'][0]`:
 - Extracts the first (most confident) language from the list of detected languages.
- `language_map = {}`:
 - Defines a dictionary that maps language codes to language names. It prints the corresponding language name (e.g., `'en'` -> `'English'`).
- `lang['Score']`:
 - Extracts the confidence score, convert it to a percentage, and rounds it to two decimal places.

```

liudayubob@Dayu:~/cits5503/lab9$ python3 detectlanguage.py
English detected with 99.84% confidence
Spanish detected with 99.92% confidence
French detected with 99.88% confidence
Italian detected with 99.65% confidence

```

[2] Sentiment Detection

We will detect the sentiment of a given text, to determine if a text expresses positive, negative, neutral, or mixed emotion.

```
def detect_sentiment(text, language_code='en'):
    response = client.detect_sentiment(Text=text, LanguageCode=language_code)
    sentiment = response['Sentiment']
    sentiment_scores = response['SentimentScore']

    print(f"Sentiment: {sentiment} with scores: {sentiment_scores}")

# Test sentiment detection with the same texts
for text in texts:
    detect_sentiment(text)
```

Code Breakdown:

1. `client.detect_sentiment()`:
 - Uses AWS Comprehend to detect the sentiment of the input text.
2. `response['Sentiment']`:
 - Extracts the detected sentiment (Positive, Negative, Neutral, or Mixed).
3. `response['SentimentScore']`:
 - Extracts the confidence scores for each sentiment type.

```
-----
Sentiment: NEUTRAL with scores: {'Positive': 0.0001116976491175592, 'Negative': 0.0002002113760681823, 'Neutral': 0.9996
846914291382, 'Mixed': 3.430022843531333e-06}
Sentiment: NEUTRAL with scores: {'Positive': 0.018150612711906433, 'Negative': 6.033524914528243e-05, 'Neutral': 0.98172
80769348145, 'Mixed': 6.095129356253892e-05}
Sentiment: POSITIVE with scores: {'Positive': 0.9699118137359619, 'Negative': 0.011562912724912167, 'Neutral': 0.0031254
67337667942, 'Mixed': 0.015399825759232044}
Sentiment: POSITIVE with scores: {'Positive': 0.99847012758255, 'Negative': 0.0003379362460691482, 'Neutral': 0.00115718
56448426843, 'Mixed': 3.477179416222498e-05}
```

[3] Entity Detection

Answer: Entities are some key element or item classes with different types, that helps to identify and classify an object.

```
def detect_entities(text, language_code='en'):
    response = client.detect_entities(Text=text, LanguageCode=language_code)
    entities = response['Entities']

    for entity in entities:
        print(f"Entity: {entity['Text']}, Type: {entity['Type']} with
{round(entity['Score']*100, 2)}% confidence")

# Test entity detection
for text in texts:
    detect_entities(text)
```

Code Breakdown:

1. `client.detect_entities()`:
 - Detects entities from the input text.
2. `response['Entities']`:
 - Extracts the detected entities and their types.

```
-----
Entity: French Revolution, Type: EVENT with 98.6% confidence
Entity: France, Type: LOCATION with 98.98% confidence
Entity: 1789, Type: DATE with 99.8% confidence
Entity: 1799, Type: DATE with 99.87% confidence
Entity: El Quijote, Type: TITLE with 96.96% confidence
Entity: Miguel de Cervantes Saavedra, Type: PERSON with 99.93% confidence
Entity: primera parte, Type: QUANTITY with 86.64% confidence
Entity: El ingenioso hidalgo don Quijote de la Mancha, Type: TITLE with 87.85% confidence
Entity: 1605, Type: DATE with 79.77% confidence
Entity: una de, Type: QUANTITY with 59.71% confidence
Entity: española, Type: OTHER with 98.65% confidence
Entity: una de las más, Type: QUANTITY with 63.2% confidence
Entity: 1615, Type: DATE with 98.81% confidence
Entity: segunda parte, Type: QUANTITY with 88.87% confidence
Entity: Quijote de Cervantes, Type: TITLE with 74.33% confidence
Entity: El ingenioso caballero don Quijote de la Mancha, Type: TITLE with 91.13% confidence
Entity: qu'aujourd'hui, Type: DATE with 65.34% confidence
Entity: Tout ce qu'il, Type: QUANTITY with 68.71% confidence
-----
```

[4] Key Phrase Detection

Answer: Key phrases are sub-groups of words that are extracted from the whole sentence, which serves as classifiers and provide important concepts to the text.

```
def detect_key_phrases(text, language_code='en'):
    response = client.detect_key_phrases(Text=text, LanguageCode=language_code)
    key_phrases = response['KeyPhrases']

    for phrase in key_phrases:
        print(f"Key Phrase: {phrase['Text']} with {round(phrase['Score']*100,
2)}% confidence")

# Test key phrase detection
for text in texts:
    detect_key_phrases(text)
```

Code Breakdown:

1. `client.detect_key_phrases()`: Identifies key phrases within the text.
2. `response['KeyPhrases']`: Extracts the list of key phrases detected.

```

-----
Key Phrase: The French Revolution with 100.0% confidence
Key Phrase: a period with 100.0% confidence
Key Phrase: social and political upheaval with 100.0% confidence
Key Phrase: France with 99.99% confidence
Key Phrase: its colonies with 100.0% confidence
Key Phrase: 1789 with 100.0% confidence
Key Phrase: 1799 with 99.99% confidence
Key Phrase: El Quijote with 99.95% confidence
Key Phrase: la obra with 100.0% confidence
Key Phrase: más conocida with 99.88% confidence
Key Phrase: Miguel de Cervantes Saavedra with 100.0% confidence
Key Phrase: su primera parte with 99.99% confidence
Key Phrase: el título with 100.0% confidence
Key Phrase: El ingenioso hidalgo don Quijote de la Mancha with 95.61% confidence
Key Phrase: comienzos with 99.98% confidence
Key Phrase: 1605 with 99.62% confidence
Key Phrase: una with 98.69% confidence
Key Phrase: las obras with 100.0% confidence
Key Phrase: más destacadas with 99.99% confidence

```

[5] Syntax Detection

Answer: Syntax assigns a token to each letter called parts of speech (POS), such as nouns, verbs, adjectives, etc.

```

def detect_syntax(text, language_code='en'):
    response = client.detect_syntax(Text=text, LanguageCode=language_code)
    syntax_tokens = response['SyntaxTokens']

    for token in syntax_tokens:
        print(f"word: {token['Text']}, POS: {token['PartOfSpeech']['Tag']} with
{round(token['PartOfSpeech']['Score']*100, 2)}% Confidence")

# Test syntax detection
for text in texts:
    detect_syntax(text)

```

Code Breakdown:

1. `client.detect_syntax()`:
 - Analyzes the text for syntactical elements like nouns, verbs, etc.
2. `response['SyntaxTokens']`:
 - Extracts each word and its corresponding part of speech.


```
-----  
Word: The, POS: DET with 100.0% Confidence  
Word: French, POS: PROPN with 100.0% Confidence  
Word: Revolution, POS: PROPN with 100.0% Confidence  
Word: was, POS: VERB with 100.0% Confidence  
Word: a, POS: DET with 100.0% Confidence  
Word: period, POS: NOUN with 100.0% Confidence  
Word: of, POS: ADP with 100.0% Confidence  
Word: social, POS: ADJ with 100.0% Confidence  
Word: and, POS: CCONJ with 100.0% Confidence  
Word: political, POS: ADJ with 100.0% Confidence  
Word: upheaval, POS: NOUN with 100.0% Confidence  
Word: in, POS: ADP with 100.0% Confidence  
Word: France, POS: PROPN with 100.0% Confidence  
Word: and, POS: CCONJ with 100.0% Confidence  
Word: its, POS: PRON with 100.0% Confidence  
Word: colonies, POS: NOUN with 100.0% Confidence  
Word: beginning, POS: VERB with 100.0% Confidence  
Word: in, POS: ADP with 100.0% Confidence  
Word: 1789, POS: NUM with 100.0% Confidence  
Word: and, POS: CCONJ with 100.0% Confidence  
Word: ending, POS: VERB with 100.0% Confidence  
Word: in, POS: ADP with 100.0% Confidence  
Word: 1799, POS: NUM with 100.0% Confidence  
Word: ., POS: PUNCT with 100.0% Confidence  
Word: El, POS: DET with 100.0% Confidence  
Word: Quijote, POS: PROPN with 100.0% Confidence  
Word: es, POS: VERB with 100.0% Confidence  
Word: la, POS: DET with 100.0% Confidence  
Word: obra, POS: NOUN with 100.0% Confidence  
Word: más, POS: ADV with 100.0% Confidence  
Word: conocida, POS: ADJ with 69.04% Confidence
```

AWS Rekognition

In this task, we will leverage AWS Rekognition to analyze image for **Label detection**, **Moderation detection**, **Facial Analysis** and **Text Extraction**.

[1] Set up instances and Upload Images

Workflow

1. **Create the Bucket/Rekognition Client:** First, we create an S3 bucket and rekognition client using `boto3` in the region `eu-north-1`. The bucket has a unique bucket name that follows the format `24188516-1ab9`
2. **Upload Images:** After the bucket is created, we upload the four images (`urban.jpg`, `beach.jpg`, `faces.jpg`, `text.jpg`) to this S3 bucket for AWS Rekognition to analyze.

```
import boto3  
  
# Constants  
REGION = "ap-southeast-2"
```

```

STUDENT_ID = "24188516"
BUCKET_NAME = f"{STUDENT_ID}-lab9"

# Initialize S3 client & Rekognition client
s3 = boto3.client('s3', region_name=REGION)
rekognition = boto3.client('rekognition', region_name=REGION)

# List of images to upload
images = ['urban.jpg', 'beach.jpg', 'faces.jpg', 'text.jpg']

def upload_images():
    # Create the S3 bucket
    bucket_config = {'LocationConstraint': REGION}
    s3.create_bucket(Bucket=BUCKET_NAME, CreateBucketConfiguration=bucket_config)

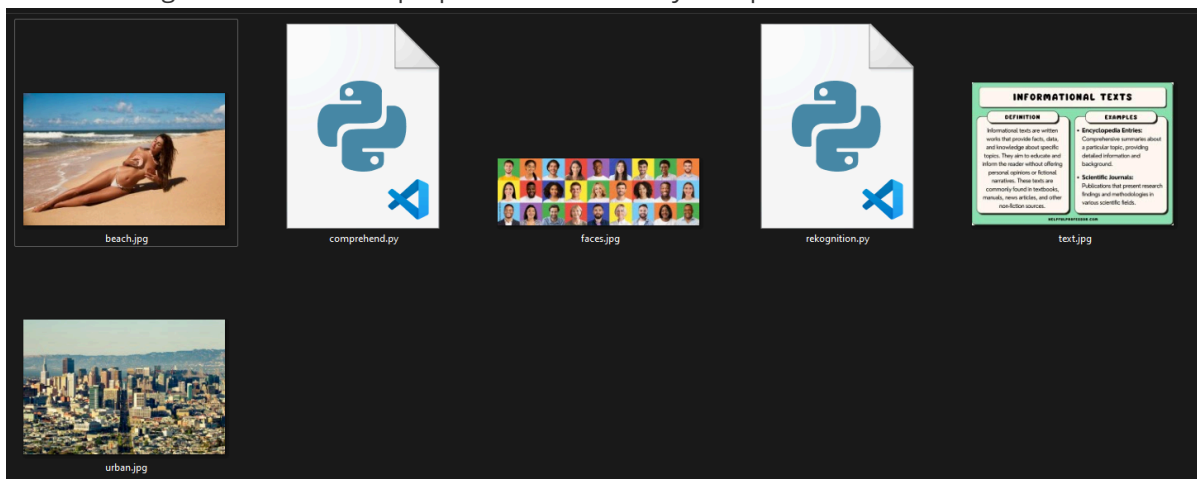
    # Upload images to the bucket
    for image in images:
        s3.upload_file(image, BUCKET_NAME, image)
    print(f"Images uploaded to {BUCKET_NAME}")

```

Code Breakdown:

1. `boto3.client()`:
 - Initializes the client to interact with AWS S3 service or AWS Rekognition service.
2. `create_bucket()`:
 - Creates an S3 bucket in the specified region, using the student's ID as part of the bucket name.
3. `upload_file()`:
 - Uploads the specified images to the bucket.

The four images files have been prepared and are ready for upload.



After execution, we can inspect the S3 bucket interface to verify successful uploads.





[Amazon S3](#) > [Buckets](#) > 24188516-lab9

24188516-lab9 [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

Objects (4) [Info](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For other

| <input type="checkbox"/> | Name | Type |
|--------------------------|---|------|
| <input type="checkbox"/> |  beach.jpg | jpg |
| <input type="checkbox"/> |  faces.jpg | jpg |
| <input type="checkbox"/> |  text.jpg | jpg |
| <input type="checkbox"/> |  urban.jpg | jpg |

[2] Test AWS Rekognition

Workflow

1. **Label Recognition:** Recognize objects, scenes, or actions from the uploaded images.
2. **Image Moderation:** Check the images for explicit or inappropriate content.
3. **Facial Analysis:** Analyze facial attributes in the images, such as emotions, gender, and age.
4. **Text Extraction:** Extract and analyze text from the image containing text.

```
def label_recognition(image):
    response = rekognition.detect_labels(
        Image={'S3Object': {'Bucket': BUCKET_NAME, 'Name': image}},
        MaxLabels=10,
        MinConfidence=70
    )
    print(f"Labels detected in {image}:")
    for label in response['Labels']:
        print(f"  {label['Name']}: {round(label['Confidence'], 2)}% confidence")

def image_moderation(image):
    response = rekognition.detect_moderation_labels(
        Image={'S3Object': {'Bucket': BUCKET_NAME, 'Name': image}},
        MinConfidence=70
    )
    print(f"Moderation labels detected in {image}:")
    for label in response['ModerationLabels']:
        print(f"  {label['Name']}: {round(label['Confidence'], 2)}% confidence")

def facial_analysis(image):
    response = rekognition.detect_faces(
        Image={'S3Object': {'Bucket': BUCKET_NAME, 'Name': image}},
```

```

        Attributes=['ALL']
    )
    print(f"Facial analysis for {image}:")
    for face in response['FaceDetails']:
        print(f"  Age range: {face['AgeRange']['Low']} - {face['AgeRange']
['High']}")
        print(f"  Emotions: {' '.join([emotion['Type'] for emotion in
face['Emotions'] if emotion['Confidence'] > 50])}")

def text_extraction(image):
    response = rekognition.detect_text(
        Image={'S3Object': {'Bucket': BUCKET_NAME, 'Name': image}}
    )
    print(f"Text detected in {image}:")
    for text in response['TextDetections']:
        print(f"  {text['DetectedText']} (Confidence: {round(text['Confidence'],
2)})%")

# Run the analyses on each image
def run_analyses():
    for image in images:
        label_recognition(image)
        image_moderation(image)
        if image == 'faces.jpg':
            facial_analysis(image)
        if image == 'text.jpg':
            text_extraction(image)

if __name__ == "__main__":
    upload_images()
    run_analyses()

```

Code Breakdown:

1. `rekognition.detect_labels()`:
 - Detects labels (objects, concepts, and actions) in the image with confidence levels.
2. `rekognition.detect_moderation_labels()`:
 - Detects inappropriate content, `Non-explicit Nudity` is detected in `beach.jpg` for example
3. `rekognition.detect_faces()`:
 - Analyzes face details such as age ranges and emotions(run only on `faces.jpg`).
4. `rekognition.detect_text()`:
 - OCR scan and extract text from images that contain written content (run only on `text.jpg`).

[3] Result Analysis

In `urban.jpg`, we can see multiple urban and city-related features, highlighting key elements of the image such as buildings, cityscapes, and outdoor settings

Labels:

- **Architecture:** 100.0% confidence
- **Building:** 100.0% confidence
- **Cityscape:** 100.0% confidence

```
Labels detected in urban.jpg:  
Architecture: 100.0% confidence  
Building: 100.0% confidence  
Cityscape: 100.0% confidence  
Urban: 100.0% confidence  
City: 100.0% confidence  
Metropolis: 99.75% confidence  
Outdoors: 93.12% confidence  
Neighborhood: 80.61% confidence  
Tower: 72.03% confidence
```

In `beach.jpg`, we can see various elements of the beach scene along with some moderation labels related to nudity and swimwear.

Labels:

- **Sunbathing:** 99.03% confidence
- **Adult:** 98.95% confidence
- **Female:** 98.95% confidence

Moderation Labels:

- **Non-Explicit Nudity of Intimate parts and Kissing:** 95.71% confidence
- **Non-Explicit Nudity:** 95.71% confidence
- **Implied Nudity:** 95.71% confidence

```
Labels detected in beach.jpg:
  Sunbathing: 99.03% confidence
  Adult: 98.95% confidence
  Female: 98.95% confidence
  Person: 98.95% confidence
  Woman: 98.95% confidence
  Beach: 82.96% confidence
  Outdoors: 82.96% confidence
  Sea: 82.96% confidence
  Shoreline: 82.96% confidence
  Water: 82.96% confidence
Moderation labels detected in beach.jpg:
  Non-Explicit Nudity of Intimate parts and Kissing: 95.71% confidence
  Non-Explicit Nudity: 95.71% confidence
  Implied Nudity: 95.71% confidence
  Swimwear or Underwear: 90.3% confidence
  Female Swimwear or Underwear: 90.3% confidence
  Partially Exposed Female Breast: 89.76% confidence
```

In `faces.jpg`, we can see various human features and emotions, labeling individuals' age ranges and facial expressions.

Labels:

- **Head:** 99.93% confidence
- **Person:** 99.93% confidence
- **Face:** 99.9% confidence

Facial Analysis:

- Age ranges vary from **20 to 26, 21 to 27, 23 to 29, 51 to 57**.
- Emotion detected is consistently **Happy** among all detected faces

```
Labels detected in faces.jpg:
  Head: 99.93% confidence
  Person: 99.93% confidence
  Face: 99.9% confidence
  Adult: 98.25% confidence
  Male: 98.25% confidence
  Man: 98.25% confidence
  Collage: 95.98% confidence
  Suit: 93.41% confidence
  Happy: 89.14% confidence
  Woman: 84.55% confidence
Moderation labels detected in faces.jpg:
Facial analysis for faces.jpg:
  Age range: 20 - 26
  Emotions: HAPPY
  Age range: 21 - 27
  Emotions: HAPPY
  Age range: 23 - 29
  Emotions: HAPPY
  Age range: 19 - 25
  Emotions: HAPPY
  Age range: 21 - 27
  Emotions: HAPPY
  Age range: 51 - 57
  Emotions: HAPPY
  Age range: 26 - 34
  Emotions: HAPPY
  Age range: 19 - 25
  Emotions: HAPPY
  Age range: 18 - 22
  Emotions: HAPPY
  Age range: 51 - 57
  Emotions: HAPPY
```

In `text.jpg`, we can see multiple text-related tags, as well as OCR results from the image.

Labels:

- **Page:** 99.99% confidence
- **Text:** 99.99% confidence
- **Menu:** 72.83% confidence

Texts Detection:

- **INFORMATIONAL TEXTS:** 99.82% confidence
- **DEFINITION:** 99.69% confidence
- **EXAMPLES:** 99.71% confidence

Labels detected in text.jpg:

Page: 99.99% confidence

Text: 99.99% confidence

Menu: 72.83% confidence

Moderation labels detected in text.jpg:

Text detected in text.jpg:

INFORMATIONAL TEXTS (Confidence: 99.82%)

DEFINITION (Confidence: 99.69%)

EXAMPLES (Confidence: 99.71%)

Informational texts are written (Confidence: 99.94%)

Encyclopedia Entries: (Confidence: 99.93%)

works that provide facts, data, (Confidence: 99.98%)

Comprehensive summaries about (Confidence: 99.81%)

and knowledge about specific (Confidence: 99.91%)

a particular topic, providing (Confidence: 99.33%)

detailed information and (Confidence: 99.92%)

topics. They aim to educate and (Confidence: 99.95%)

inform the reader without offering (Confidence: 99.96%)

background. (Confidence: 100.0%)

personal opinions or fictional (Confidence: 99.87%)

Scientific Journals: (Confidence: 99.79%)

narratives. These texts are (Confidence: 99.9%)

Publications that present research (Confidence: 99.95%)

commonly found in textbooks, (Confidence: 100.0%)