Build and Deploy Documentation

SightScan | AMOS Project Team 2

INTRODUCTION

This documentation describes how to build and deploy all components of SightScan. Make sure to follow the exact order presented in this documentation to prevent any dependency clashes between the components. The same applies to our advised hardware requirements, especially on the model training side. This documentation relies on any arbitrary Unix based operating system. We assume that the open source SightScan repository is cloned and located in your local home directory, i.e. ~/amos-pj-ws20-21-computer-vision-for-sights directory. Furthermore, being familiar with the technical documentation and especially the software architecture before advancing to the following contents is highly advised. If you simply want to launch selected components locally using Docker, please refer to the README.md files in the individual component folders.

HOW TO: BUILD AND DEPLOY

PostgreSQL Data Warehouse (DWH)

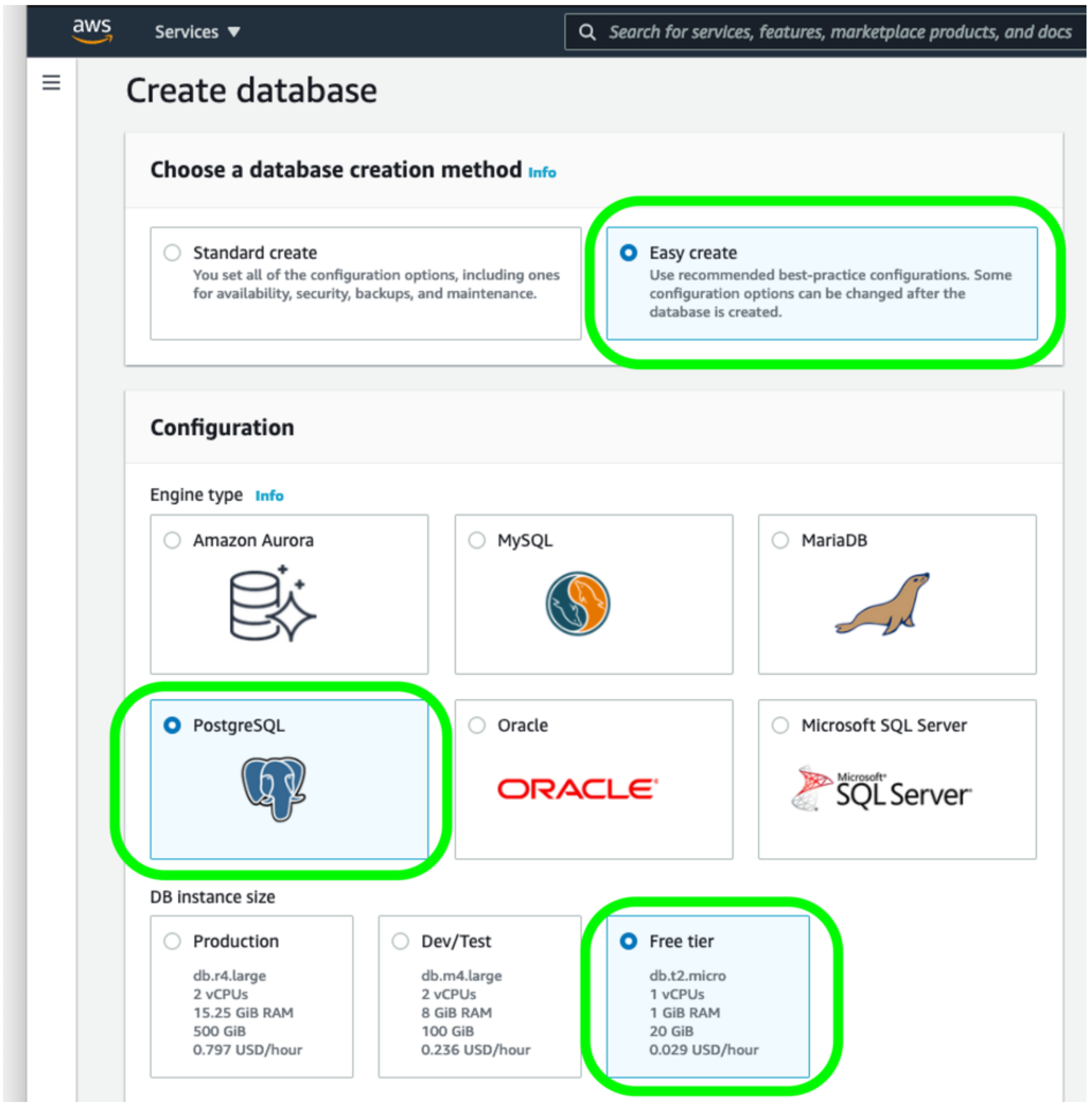
PostgreSQL is the open source, object-relational database SightScan utilizes as its central data warehouse. It is deployed on Amazon RDS (Relational Database Service) with the possibility of using the free tier for testing or development purposes.

Important Environment Variables

- **PGHOST**: raw host URL the DWH runs on (without http:// or https:// as a prefix)
- **PGPORT**: port the DWH is available from
- **PGDATABASE**: name of the internal database the DWH is setup with
- **PGUSER**: username for accessing the database
- **PGPASSWORD**: password for the specified user

Deployment

1. On Amazon Web Service, navigate to the Amazon RDS dashboard and click "Create database"
2. Choose PostgreSQL as the database engine of choice; optionally, feel free to use the free tier configuration with limited hardware performance


aws
Services ▼
Search for services, features, marketplace products, and docs

Create database

Choose a database creation method Info

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

[✓ Easy create]
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Configuration

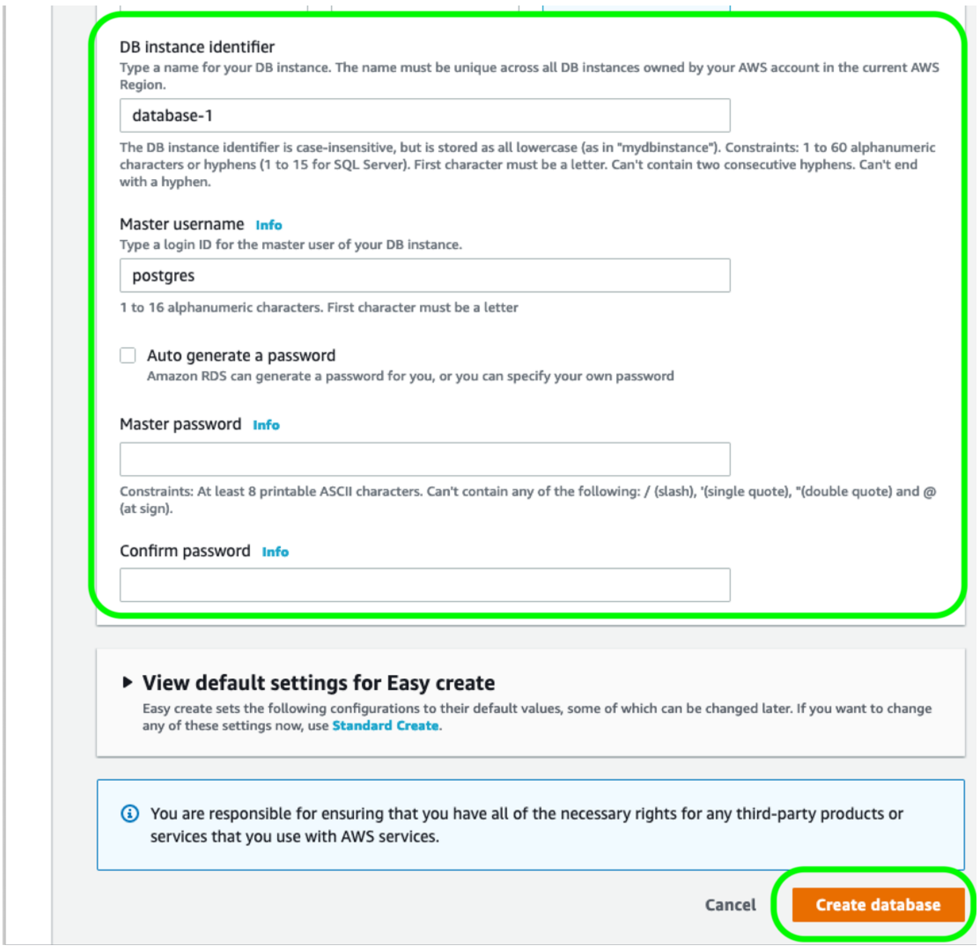
Engine type Info

Amazon Aurora | MySQL | MariaDB | [✓ PostgreSQL] | Oracle | Microsoft SQL Server

DB instance size

Production	Dev/Test	[✓ Free tier]
db.r4.large	db.m4.large	db.t2.micro
2 vCPUs	2 vCPUs	1 vCPUs
15.25 GiB RAM	8 GiB RAM	1 GiB RAM
500 GiB	100 GiB	20 GiB
0.797 USD/hour	0.236 USD/hour	0.029 USD/hour

3. Specify a database name, username, and a password for accessing the database; these parameters constitute the PGDATABASE, PGUSER, and PGPASSWORD parameters



DB instance identifier
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

database-1

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Master username Info
Type a login ID for the master user of your DB instance.

postgres

1 to 16 alphanumeric characters. First character must be a letter

☐ Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password Info

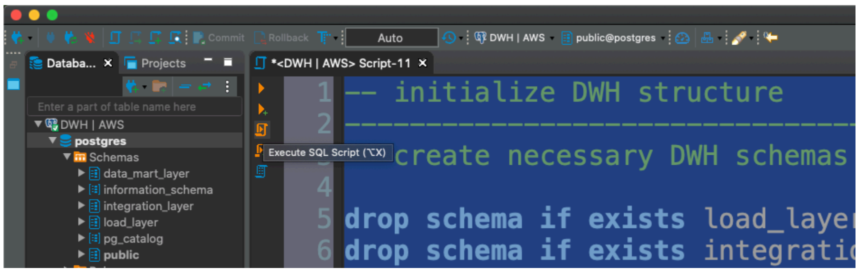
Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password Info

▸ **View default settings for Easy create**
Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard Create**.

ⓘ You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

4. Confirm your input and create the database instance
5. In the settings, make sure incoming traffic from anywhere (0.0.0.0/0 and ::/0)
6. Execute the database_init.sql file on the newly created database instance to initialize its schema and automation mechanisms – either directly in the Amazon RDS dashboard or by using an external tool of choice, e.g. DBeaver, as can be seen below:



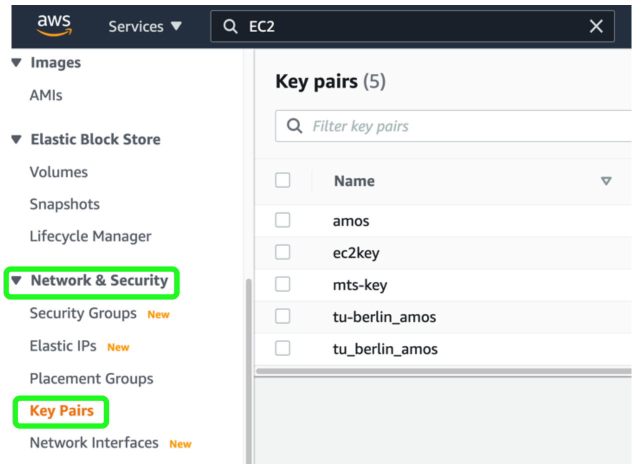
```
-- initialize DWH structure
------------------------------------
create necessary DWH schemas

drop schema if exists load_laye
drop schema if exists integratio
```

Global SSH Key for Computing Instance Access

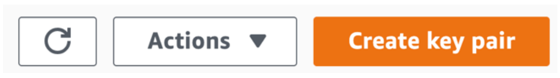
Since SightScan heavily relies on distributed computing instances, multiple instances must be managed. It uses Amazon Web Services (AWS) as the cloud platform of choice. In order to avoid juggling with SSH keys, we highly advise to generate a global SSH key for accessing all EC2 instances collectively. This way, key exchanges are easier on AWS, as only a single key reference needs to be replaced instead of taking care of every computing instance individually – an especially huge manual effort for large production environments. If you do not want to do this, repeat this section for every component you want to deploy, and provide the respective path to the individual SSH key in each SSH command.

1. Visit the EC2 service page on AWS and navigate to the “Key Pairs” field under “Under Network & Security”

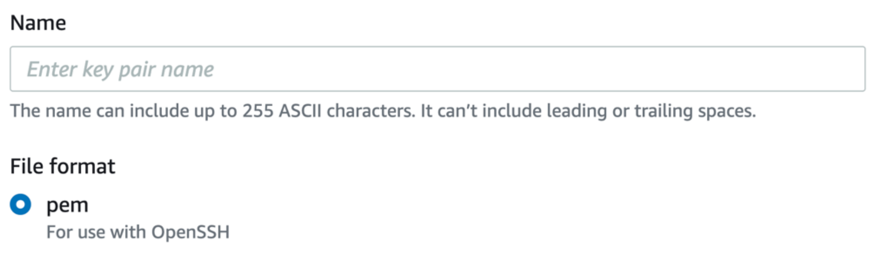


amos
ec2key
mts-key
tu-berlin_amos
tu_berlin_amos

2. Click “Create key pair”



3. Create a new SSH key pair in the .pem format and using your selected name of choice; in this documentation, our global, private SSH key is persisted in the file **ec2key.pem**


Name

Enter key pair name

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

File format

[✓ pem]
For use with OpenSSH

4. Download the private SSH key so that it resides in the local Download folder of your home directory

Image Crawler (IC)

The crawler is responsible for retrieving sight images for a given city and inserting them into the data warehouse subsequently.

Important Environment Variables

- **IC_URL:** URL of the EC2 instance of the image crawler
- For more: see **DOS** component

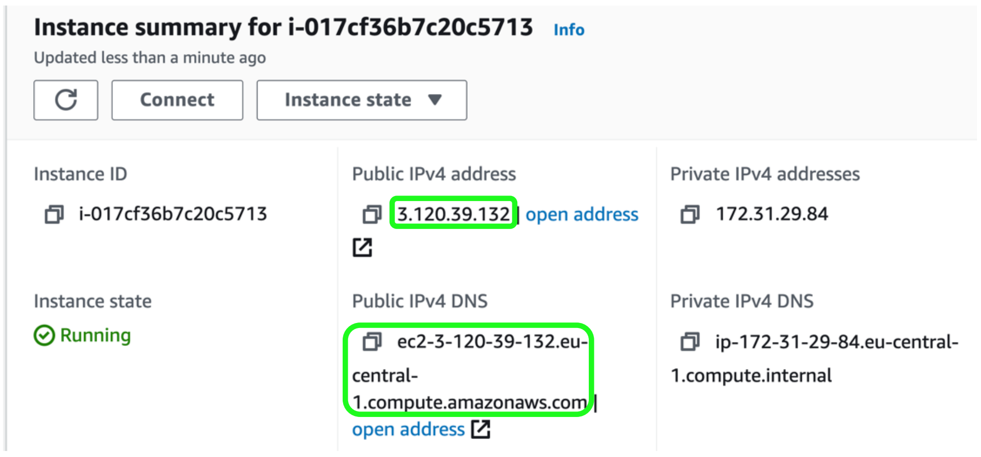
Additional Command Line Arguments

- **skip**: skips keywords if a downloaded directory already exists - this is useful for caching already downloaded items or forcing re-downloads; default: true
- **threads**: number of threads available for downloading; default: 4
- **google**: whether to download from google.com; default: true
- **full**: whether to download images in full resolution; default: false
  - Note: enabling full resolution downloads often substantially improves model performance, but comes with the expense of increased memory footprints in the data warehouse

- **face**: whether to use the included face search mode; default: false
- **no_gui**: whether to load images in headless mode; default: auto
    - Note: When this parameter is set to auto instead of a Boolean value, it is internally overwritten with false if the full argument is true, otherwise it is set to false
- **limit**: maximum images to download per identified sight; default: 0
    - Note: setting this parameter to 0 means unbounded – this could theoretically lead to an unlimited number of persisted images in the data warehouse and should thus be avoided
- **no_driver**: whether a dedicated image crawling driver should be used; default: false
- **location**: name of the city to crawl sight images for
- **sights_limit**: maximum sights to support for a given city

Deployment

1. Launch an EC2 instance for the IC component with the following basic requirements:
    - **Operating system**: up-to-date Linux Ubuntu
    - **Computing power**: at least two vCPUs
    - **Random access memory**: at least 8 GB
    - **Disk memory**: enough memory to store the downloaded sight images for a given city; we recommend 5 GB memory for small and at least 40GB for large deep learning models
    - **Security group**: allowing traffic from 0.0.0.0/0 and ::/0 origins for port 22 (SSH port), or preferably merely from the default AWS virtual private cloud (VPC) IP range
    - **Docker daemon**: configured and running
2. Prepare the folder structures on the EC2 instance to deploy your code into:
    a. Notice the target values for **IC_IP** (public IPv4 IP address) and **IC_URL** (public IPv4 DNS) on the main page of your launched instance.


Instance summary for i-017cf36b7c20c5713

Updated less than a minute ago

Instance ID: i-017cf36b7c20c5713

Public IPv4 address: 3.120.39.132 | open address

Private IPv4 addresses: 172.31.29.84

Instance state: Running

Public IPv4 DNS: ec2-3-120-39-132.eu-central-1.compute.amazonaws.com | open address

Private IPv4 DNS: ip-172-31-29-84.eu-central-1.compute.internal

b. Connect to the instance via SSH through your terminal of choice:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<IC_IP>
```

c. On the instance, create a dedicated crawler folder to store the artifacts:

```
sudo mkdir crawler
sudo chmod 777 crawler
exit
```

3. From your local computer, upload the cache-free, clean code to the EC2 instance of the crawler:

```
sudo scp -i ~/Downloads/ec2key.pem -r ~/amos-pj-ws20-21-
computer-vision-for-sights/amos/crawler
ubuntu@<IC_URL>:~/crawler/
```

4. Reconnect to the instance via SSH and build its Docker image:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<IC_IP>
cd crawler/crawler
sudo docker build . -t crawler
exit
```

5. That's it – now, executions of the constructed Docker image can be triggered from a remote DOS instance through SSH.

Model Training Service (MTS)

The model training service is responsible for training Yolov5 deep learning models for computer vision.

Important Environment Variables

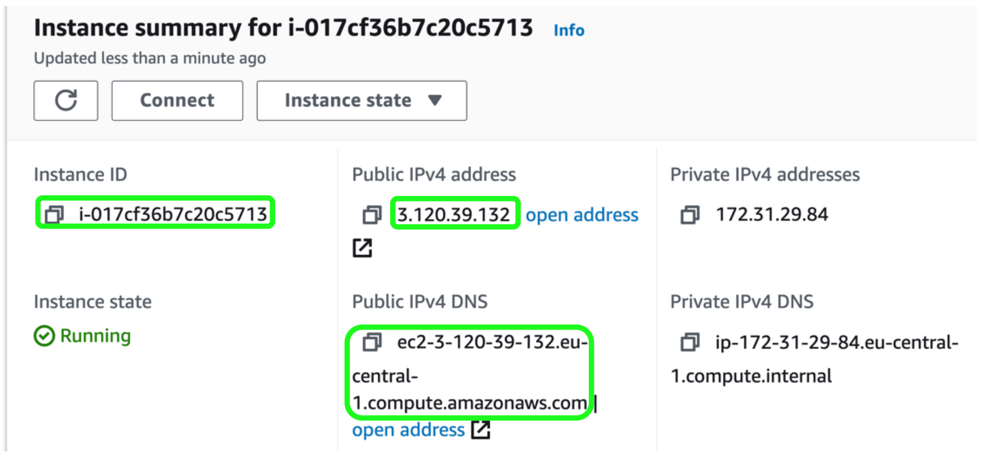
- **MTS_EC2_INSTANCE_ID**: id assigned to the EC2 instance of the MTS
- For more: see **DMR** component

Deployment

1. Launch an EC2 instance for the MTS component with the following basic requirements:

- **Amazon Machine Image (AMI)**: deep learning-specialized AMI with
    - **Operating system**: up-to-date Linux Ubuntu
    - **Docker and Nvidia Docker pre-installed**
    - **CUDA and PyTorch optimization**
    - Recommendation: Deep Learning Base AMI (Ubuntu 18.04) Version 32.0 (or higher)
- **Computing power**: at least four vCPUs, at least Nvidia Volta GPU
    - Note: older Nvidia Tesla GPUs like the K80 are not supported by our Yolov5 setup
- **Random access memory**: at least 16 GB
- **Disk memory**: persistent EBS block storage and enough memory to store the downloaded sight images for a given city, but at least 85 GB
- **Security group**: allowing traffic from 0.0.0.0/0 and ::/0 origins for port 22 (SSH port), or preferably merely from the default AWS virtual private cloud (VPC) IP range
- Our EC2 instance recommendation: p3.xlarge type for most cost-efficient compatibility

2. Prepare the folder structures on the EC2 instance to deploy your code into:
   a. Notice the target values for **MTS_IP** (public IPv4 IP address) and **MTS_URL** (public IPv4 DNS), and the environment variable **MTS_EC2_INSTANCE_ID** (i-xxxxxxxxxxxxx) on the main page of your launched instance.



Instance summary for i-017cf36b7c20c5713 Info
Updated less than a minute ago

Instance ID	Public IPv4 address	Private IPv4 addresses
i-017cf36b7c20c5713	3.120.39.132 open address	172.31.29.84
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	ec2-3-120-39-132.eu-central-1.compute.amazonaws.com open address	ip-172-31-29-84.eu-central-1.compute.internal

   b. Connect to the instance via SSH through your terminal of choice:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<MTS_IP>
```

   c. Make sure the AWS CLI is installed to enable dynamic remote calls:

```
sudo apt-get update
sudo apt-get install awscli
```

   d. On the instance, create a dedicated crawler folder to store the artifacts:

```
sudo mkdir mts
sudo chmod 777 mts
exit
```

3. From your local computer, upload the cache-free, clean code to the EC2 instance of the MTS:

```
sudo scp -i ~/Downloads/ec2key.pem -r ~/amos-pj-ws20-21-computer-vision-for-sights/amos/mts ubuntu@<MTS_URL>:~/mts/
```

4. Upload the orchestrating bash script to the EC2 instance of the MTS:

```
sudo scp -i ~/Downloads/ec2key.pem ~/amos-pj-ws20-21-computer-vision-for-sights/amos/mts.sh ubuntu@<MTS_URL>:~/mts/
```

5. *Reconnect to the instance via SSH and make the bash script executable:*

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<MTS_IP>
cd mts
command chmod +x mts.sh
```

6. Build the Docker image for the MTS:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<MTS_IP>
cd mts/yolov5
sudo docker build . -t mts
exit
```

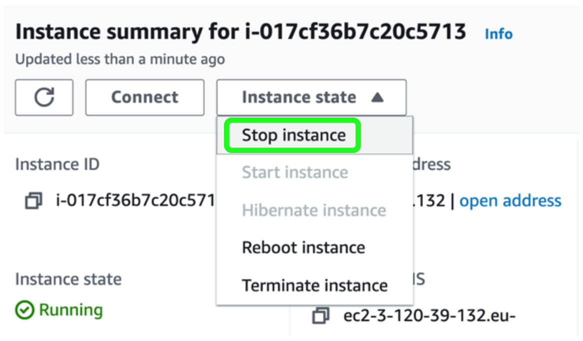
7. Stop the EC2 instance of the MTS

   Note 1: This step is crucial since the EC2 instance of the MTS is dynamically booted and shut down for training to save costs – GPU instances are expensive. By stopping the instance initially, we allow it

   to be booted by dedicated city training jobs and minimize financial expenditures.

   Note 2: Terminating the instance deallocates the entire instance forever and should not be

   performed at all – stopping it merely shuts it down and retains the configurations.



8. That's it – now, city training jobs can be triggered from remote services, which subsequently boots the MTS instance, performs training, and finally shuts the instance down.

Image Labelling Service (ILS)

The image labelling service communicates with the Google Vision API in order to provide labels for cities that have been lately added to SightScan.

Important Environment Variables

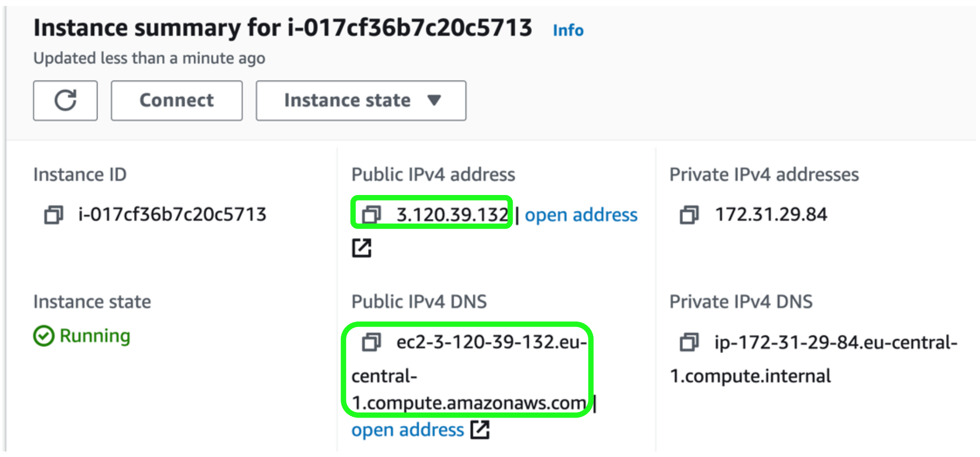
- **ILS_PORT**: port the ILS runs on
  - Note: for the sake of simplicity, picking the container-internal port (8001) is advised
- **MAX_GOOGLE_VISION_CALLS_PER_NEW_CITY**: maximum number of requests sent to the Google Vision API for a newly supported city to restrict costs
- **PGHOST, PGPORT, PGDATABASE, PGUSER,** and **PGPASSWORD** (database parameters from above)

Deployment

1. Launch an EC2 instance for the ILS component with the following basic requirements:
- **Operating system**: up-to-date Linux Ubuntu
- **Computing power**: at least two vCPU
- **Random access memory**: at least 4 GB
- **Docker daemon**: configured and running
- **Security group**: allowing traffic from 0.0.0.0/0 and ::/0 origins for port 22 (SSH port) and TCP traffic on **<ILS_PORT>**, or preferably merely from the default AWS virtual private cloud (VPC) IP range

2. Prepare the folder structures on the EC2 instance to deploy your code into:

   a. Notice the target values for **ILS_IP** (public IPv4 IP address) and **ILS_URL** (public IPv4 DNS) on the main page of your launched instance.


Instance summary for i-017cf36b7c20c5713 Info
Updated less than a minute ago
Connect
Instance state

Instance ID: i-017cf36b7c20c5713
Public IPv4 address: 3.120.39.132 | open address
Private IPv4 addresses: 172.31.29.84
Instance state: Running
Public IPv4 DNS: ec2-3-120-39-132.eu-central-1.compute.amazonaws.com | open address
Private IPv4 DNS: ip-172-31-29-84.eu-central-1.compute.internal

b. Connect to the instance via SSH through your terminal of choice:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<ILS_IP>
```

c. On the instance, create a dedicated ILS folder to store the artifacts:

```
sudo mkdir ils
sudo chmod 777 ils
exit
```

3. From your local computer, upload the cache-free, clean code to the EC2 instance of the ILS:

```
sudo scp -i ~/Downloads/ec2key.pem -r ~/amos-pj-ws20-21-computer-vision-for-sights/amos/image_labelling_service ubuntu@<ILS_URL>:~/ils/
```

4. Reconnect to the instance via SSH and build its Docker image:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<ILS_IP>
cd ils/image_labelling_service
sudo docker build . -t ils
```

5. Launch the ILS on its EC2 instance via Docker

```
sudo docker run -d
-e PGHOST=<PGHOST>
-e PGDATABASE=<PGDATABASE>
-e PGUSER=<PGUSER>
-e PGPORT=<PGPORT>
-e PGPASSWORD=<PGPASSWORD>
-e MAX_GOOGLE_VISION_CALLS_PER_NEW_CITY=
<MAX_GOOGLE_VISION_CALLS_PER_NEW_CITY>
-p <ILS_PORT>:8001
-it ils
```

6. Exit the instance:

```
exit
```

Data Mart Refresher (DMR)

The data mart refresher component refreshes data warehouse contents asynchronously to expose both labelled images and trained models in dedicated data marts for faster queries. During refreshes, it identifies new cities to be labelled and triggers labelling processes. Furthermore, it detects whenever new city models are trainable, and notifies the MTS subsequently.

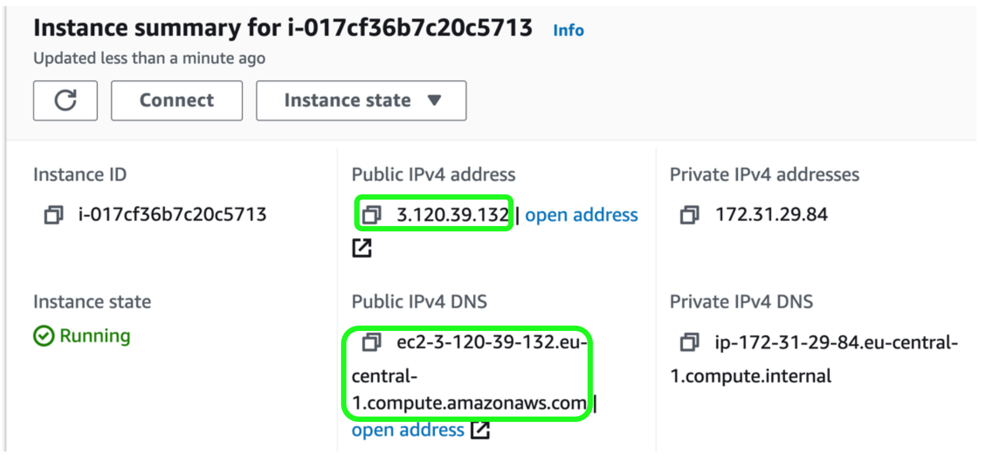
Important Environment Variables

- **AWS_ACCESS_KEY_ID**: Amazon Web Services key ID for the IAM role under which the MTS should be triggered (needs Full EC2 permission policy)
- **AWS_ACCESS_KEY**: corresponding key for the given key ID
- **AWS_REGION**: region of the AWS EC2 instances; generally: eu-central-1
- **DATA_MART_REFRESH_DATA_MARTS_EVERY_SECONDS**: delay between resource-intensive data mart refreshes
  - Note: typically, refreshing every hour (setting this value to 3600) is more than enough since users usually not count the seconds until a new model is available
- **DATA_MART_ENABLE_MODEL_TRAINING_EVERY_SECONDS**: delay between checking if a city is available for training and potentially triggering the MTS
  - Note: with the same reasoning as before, we advise setting this value to 3600 or higher; training usually takes more than an hour anyway
- **DATA_MART_ENABLE_LABELLING_REQUESTS_EVERY_SECONDS**: delay between emitting a batch of labelling requests to the ILS component
  - Note: the same reasoning as with the two previously mentioned parameters applies here
- **ILS_PUBLIC_ENDPOINT_URL**: public URL to the ILS service, including port and http prefix
- **MTS_EC2_INSTANCE_ID**: cf. above

- **MTS_EPOCHS**: number of epochs the MTS should train for a new city
- **MIN_LABELLED_IMAGES_NEEDED_FOR_TRAINING**: minimum number of required labelled images to trigger a city training job in the MTS component
- **MIN_IMAGE_NUMBER_PER_LABEL**: minimum number of required available images per label
  - Note: setting this parameter too high leaves more data points out of the training points, setting it too low may decrease the model performance due to highly poor class performance; consider the trade-off between data volume and data quality in your training process
- **PGHOST, PGPORT, PGDATABASE, PGUSER,** and **PGPASSWORD** (database parameters from above)

Deployment

1. Launch an EC2 instance for the DMR with the following basic requirements:
   - **Operating system**: up-to-date Linux Ubuntu
   - **Computing power**: at least one vCPU
   - **Random access memory**: at least 4 GB
   - **Docker daemon**: configured and running
2. Prepare the folder structures on the EC2 instance to deploy your code into:
   a. Notice the target values for **DMR_IP** (public IPv4 IP address) and **DMR_URL** (public IPv4 DNS) on the main page of your launched instance.


Instance summary for i-017cf36b7c20c5713

Updated less than a minute ago

Instance ID	Public IPv4 address	Private IPv4 addresses
i-017cf36b7c20c5713	3.120.39.132	open address	172.31.29.84
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	ec2-3-120-39-132.eu-central-1.compute.amazonaws.com	open address	ip-172-31-29-84.eu-central-1.compute.internal

   b. Connect to the instance via SSH through your terminal of choice:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<DMR_IP>
```

   c. On the instance, create a dedicated data mart refresher folder to store the artifacts:

```
sudo mkdir dmr
sudo chmod 777 dmr
exit
```

3. From your local computer, upload the cache-free, clean code to the EC2 instance of the DMR:

```
sudo scp -i ~/Downloads/ec2key.pem -r ~/amos-pj-ws20-21-computer-vision-for-sights/amos/data_mart_refresher ubuntu@<DMR_URL>:~/dmr/
```

4. Reconnect to the instance via SSH and build its Docker image:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<DMR_IP>
cd dmr/data_mart_refresher
sudo docker build . -t dmr
```

5. Launch the DMR on its EC2 instance via Docker:

```
sudo docker run -d
-e ILS_PUBLIC_ENDPOINT_URL=<ILS_PUBLIC_ENDPOINT_URL>
-e MTS_EC2_INSTANCE_ID=<MTS_EC2_INSTANCE_ID>
-e AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
-e AWS_ACCESS_KEY=<AWS_ACCESS_KEY>
-e AWS_REGION=<AWS_REGION>
-e MTS_EPOCHS=<MTS_EPOCHS>
-e DATA_MART_REFRESH_DATA_MARTS_EVERY_SECONDS=
<DATA_MART_REFRESH_DATA_MARTS_EVERY_SECONDS>
-e DATA_MART_ENABLE_MODEL_TRAINING_EVERY_SECONDS=
<DATA_MART_ENABLE_MODEL_TRAINING_EVERY_SECONDS>
-e DATA_MART_ENABLE_LABELLING_REQUESTS_EVERY_SECONDS=
<DATA_MART_ENABLE_LABELLING_REQUESTS_EVERY_SECONDS>
-e PGHOST=<PGHOST>
-e PGDATABASE=<PGDATABASE>
-e PGUSER=<PGUSER>
-e PGPORT=<PGPORT>
-e PGPASSWORD=<PGPASSWORD>
-e MIN_LABELLED_IMAGES_NEEDED_FOR_TRAINING=
<MIN_LABELLED_IMAGES_NEEDED_FOR_TRAINING>
-e MIN_IMAGE_NUMBER_PER_LABEL=<MIN_IMAGE_NUMBER_PER_LABEL>
-it dmr
```

6. Exit the instance:

```
exit
```

Django Orchestration Service (DOS)

The Django orchestration service primarily functions as an abstraction layer between the client and the backend microservice ensemble. It persists client data in the data warehouse, exposes data from the latter to the client, and triggers the image crawler if needed.

Important Environment Variables

- **DOS_PORT**: the port that exposes the DOS endpoints for access by clients
- **IC_URL:** URL of the EC2 instance of the image crawler (cf. above)
- **MAPS_KEY**: Google Maps key passed to the triggered image crawler for sight name retrieval
- **MAX_IMAGES_PER_SIGHT**: maximum number of images the triggered image crawler should retrieve per sight
    - Note: under the hood, this parameter sets the limit command line argument of the image crawler
- **MAX_SIGHTS_PER_CITY**: maximum number of sights the triggered image crawler should retrieve for the city

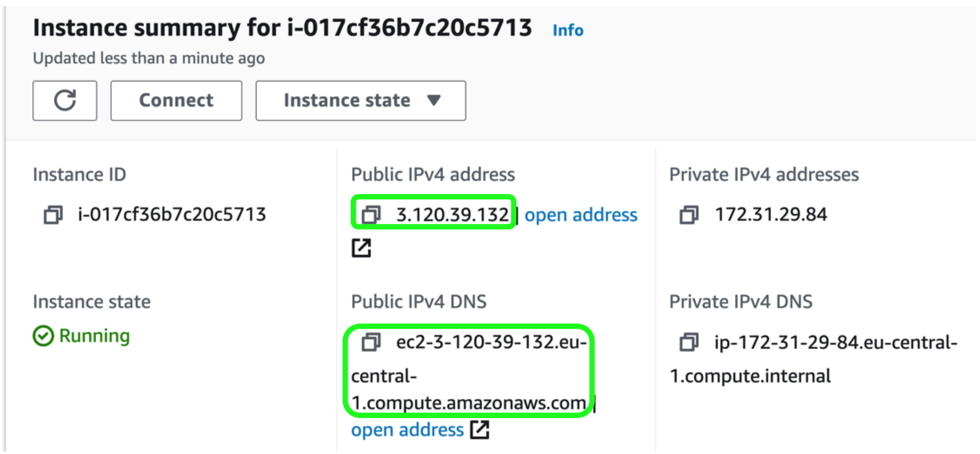
    Note: this sets the command line argument sights_limit of the triggered image crawler
- **PGHOST, PGPORT, PGDATABASE, PGUSER,** and **PGPASSWORD** (database parameters from above)

Deployment

1. Launch an EC2 instance for the DOS with the following basic requirements:
    - **Operating system**: up-to-date Linux Ubuntu
    - **Computing power**: at least one vCPU
    - **Random access memory**: at least 4 GB
    - **Docker daemon**: configured and running

- **Security group**: allowing traffic from 0.0.0.0/0 and ::/0 origins for port 22 (SSH port and TCP traffic on port **<DOS_PORT>**

2. Prepare the folder structures on the EC2 instance to deploy your code into:
   a. Notice the target values for **DOS_IP** (public IPv4 IP address) and **DOS_URL** (public IPv4 DNS) on the main page of your launched instance.



Instance summary for i-017cf36b7c20c5713 Info
Updated less than a minute ago

Instance ID	Public IPv4 address	Private IPv4 addresses
i-017cf36b7c20c5713	3.120.39.132 | open address	172.31.29.84
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	ec2-3-120-39-132.eu-central-1.compute.amazonaws.com | open address	ip-172-31-29-84.eu-central-1.compute.internal

   b. Connect to the instance via SSH through your terminal of choice:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<DOS_IP>
```

   c. On the instance, create a dedicated Django orchestrator folder to store the artifacts:

```
sudo mkdir dos
sudo chmod 777 dos
exit
```

3. From your local computer, upload the cache-free, clean code to the EC2 instance of the DOS:

```
sudo scp -i ~/Downloads/ec2key.pem -r ~/amos-pj-ws20-21-computer-vision-for-sights/amos/django_orchestrator ubuntu@<DOS_URL>:~/dos/
```

4. Reconnect to the instance via SSH and build its Docker image:

```
sudo ssh -i ~/Downloads/ec2key.pem ubuntu@<DMR_IP>
cd dmr/data_mart_refresher
sudo docker build . -t dmr
```

5. Launch the DOS on its EC2 instance via Docker:

```
sudo docker run -d
-e PGHOST=<PGHOST>
-e PGDATABASE=<PGDATABASE>
-e PGUSER=<PGUSER>
-e PGPORT=<PGPORT>
-e PGPASSWORD=<PGPASSWORD>
-e IC_URL=<IC_URL>
-e MAX_SIGHTS_PER_CITY=<MAX_SIGHTS_PER_CITY>
-e MAX_IMAGES_PER_SIGHT=<MAX_IMAGES_PER_SIGHT>
-e MAPS_KEY=<MAPS_KEY>
-p <DOS_PORT>:8002 -it dos
```

6. Exit the instance:

```
exit
```

Graphical User Interface

The graphical user interface is controlled by the user – alongside communication with the DOS component, it performs typical SightScan functionalities, e.g. detecting a city's sights in a real-time webcam stream.

Important Environment Variables

- **API_ENDPOINT_URL**: full base URL of the DOS component
  - **Note**: this parameter needs to follow the format http://**<DOS_URL>**:**<DOS_PORT>**

Deployment

1. Navigate into the folder containing the GUI

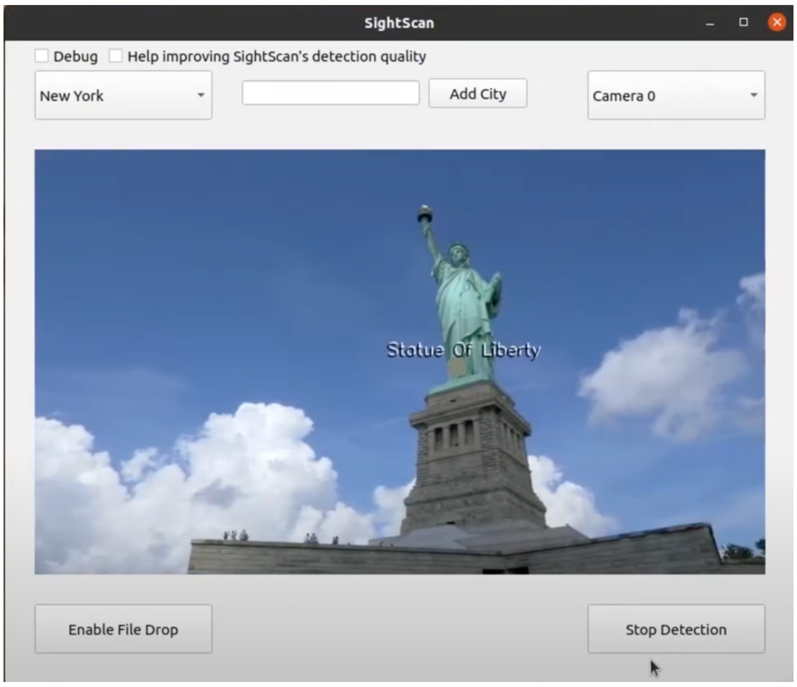
```
cd ~/amos-pj-ws20-21-computer-vision-for-sights/amos/gui
```

2. Create a file with name .env, containing only a single line:

```
API_ENDPOINT_URL= http://<DOS_URL>:<DOS_PORT>
```

3. Make sure SightScan's GUI runs as expected on the client's local hardware by double-click the main.py file to launch it; alternatively, you may also double-click on the bash script located in the same folder

   Note: the GUI has been tested on Windows 10, many Linux versions, and macOS



Large-Scale Production Environments

If you desire to deploy SightScan as a large-scale application, such as rolling it out for an entire country, you obviously need load balancers and many more computing resources than just one EC2 instance per component. In that case, since all components are dockerized, they can be easily deployed through Kubernetes via the Amazon Elastic Kubernetes Service (EKS) or EC2 Auto Scaling.

Final Words

Congratulations – at this point, the SightScan ecosystem is successfully deployed and ready to unleash its entire capabilities upon its users. Enjoy! ☺