Author: Madhurima Rawat

Using Cloud Functions for Serverless Computing

This experiment focuses on the deployment, execution, and testing of an AWS Lambda function using AWS CLI and LocalStack. The Lambda runtime image is pulled from Amazon ECR and executed locally using Docker, simulating a serverless environment. The process includes function invocation and result validation.

This document provides a comprehensive breakdown of all commands, inputs, outputs, and their explanations, ensuring a clear understanding of each step in the workflow.

1. Start LocalStack and Docker

```
localstack start
```

- Ensures that LocalStack is running before executing AWS CLI commands.

2. Creating a VPC

Command:

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --endpoint-
url=http://localhost:4566
```

Output:

```
-------------------------------------------------------------
|                        CreateVpc                          |
+-----------------------------------------------------------+
||                           Vpc                           ||
|+-------------------------+-------------------------------+|
||  CidrBlock              |  10.0.0.0/16                  ||
||  DhcpOptionsId          |  default                      ||
||  InstanceTenancy        |  default                      ||
||  OwnerId                |  000000000000                 ||
||  State                  |  pending                      ||
||  VpcId                  |  vpc-66375b0cbe498b519        ||
|+-------------------------+-------------------------------+|
|||               CidrBlockAssociationSet                |||
```

```
||+----------------+-------------------------------------+||
|||  AssociationId |  vpc-cidr-assoc-0c82d2b7eaae6089a  |||
|||  CidrBlock     |  10.0.0.0/16                       |||
||+----------------+-------------------------------------+||
||||                  CidrBlockState                   ||||
|||+------------------+-------------------------------+|||
||||  State            |  associated                  ||||
|||+------------------+-------------------------------+|||
```

Explanation:

- `aws ec2 create-vpc` → Creates a new Virtual Private Cloud (VPC).
- `--cidr-block 10.0.0.0/16` → Defines the IP address range for the VPC.
- `--endpoint-url=http://localhost:4566` → Uses **LocalStack** instead of AWS.
- **State:** `"pending"` → The VPC is being created.
- **VpcId:** `"vpc-66375b0cbe498b519"` → Unique identifier for the VPC.

3. Creating a Subnet

Command:

```
aws ec2 create-subnet --vpc-id vpc-66375b0cbe498b519 --cidr-block
10.0.1.0/24 --endpoint-url=http://localhost:4566
```

Output:

```
-----------------------------------------------------------------------------------------
|                                          CreateSubnet                                 |
+---------------------------------------------------------------------------------------+
||                                            Subnet                                   ||
|+-----------------------------+-------------------------------------------------------+
||  AssignIpv6AddressOnCreation |  False                                                ||
||  AvailabilityZone            |  us-east-1b                                           ||
||  AvailabilityZoneId          |  use1-az1                                             ||
||  AvailableIpAddressCount     |  251                                                  ||
||  CidrBlock                   |  10.0.1.0/24                                          ||
||  DefaultForAz                |  False                                                ||
||  Ipv6Native                  |  False                                                ||
||  MapPublicIpOnLaunch         |  False                                                ||
||  OwnerId                     |  000000000000                                         ||
||  State                       |  pending                                              ||
||  SubnetArn                   |  arn:aws:ec2:us-east-1:000000000000:subnet/subnet-3f40f7c6e3
||  SubnetId                    |  subnet-3f40f7c6e3a26040f                             ||
```

```
||  VpcId                       |  vpc-66375b0cbe498b519
|+-------------------------------+-------------------------------------------------------------
```

Explanation:

- `aws ec2 create-subnet` → Creates a new subnet within a VPC.
- `--vpc-id vpc-66375b0cbe498b519` → Specifies the VPC in which the subnet is created.
- `--cidr-block 10.0.1.0/24` → Defines the subnet's IP range.
- **SubnetId:** `"subnet-3f40f7c6e3a26040f"` → Unique identifier for the subnet.
- **State:** `"pending"` → The subnet is being created.

4. Attempting to Create a Load Balancer

Command:

```
aws elbv2 create-load-balancer --name my-load-balancer --subnets
subnet-3f40f7c6e3a26040f --security-groups default --type application
--endpoint-url=http://localhost:4566
```

Output:

```
An error occurred (InternalFailure) when calling the CreateLoadBalancer
operation: API for service 'elbv2' not yet implemented or pro feature -
please check https://docs.localstack.cloud/references/coverage/
for further information
```

Explanation:

- `aws elbv2 create-load-balancer` → Attempts to create an **Application Load Balancer**.
- `--subnets subnet-3f40f7c6e3a26040f` → Specifies the subnet.
- **Error:** LocalStack does not support `elbv2` without a pro subscription.

5. Running Docker Containers

Command:

```
docker run -d --name backend1 nginx
```

```
docker run -d --name backend2 nginx
```

Output:

```
63d7914e78bb0791288824fffdcf46d77bd92b4f895de4a403c88012dc87b601
e565ff4cf0081c7965a7634fb1de6018e88d927dd80ca179822aa7a33994f87c
```

Explanation:

- `docker run -d --name backend1 nginx` → Starts an **nginx** container in detached mode (`-d`).
- **Container ID:** `"63d7914e78bb..."` → Unique ID of the container.

6. Checking Running Docker Containers

Command:

```
docker ps
```

Output:

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS
e565ff4cf008   nginx                  "/docker-entrypoint.…"   5 minutes ago    Up 5 minutes
63d7914e78bb   nginx                  "/docker-entrypoint.…"   5 minutes ago    Up 5 minutes
34f41149b6a9   localstack/localstack  "docker-entrypoint.sh"   15 minutes ago   Up 15 minutes
```

Explanation:

- **`docker ps`** → Lists active containers.
- `"backend1"` and `"backend2"` → Both are running.

7. Running a Load Balancer with Docker

Command:

```
docker run -d --name load-balancer -p 8080:80 -v "C:/Users/rawat/Documents
/8 SEMESTER/Cloud Computing/Lab/Experiment 6/Codes/nginx.conf:/etc/nginx/
```

```
nginx.conf" nginx
```

Output:

```
a9fa9f96d302eacadba15bb150052e4dd4880a37d38e1f5811603296bac28884
```

Explanation:

- **Fixes volume path issue:** `"C:/Users/rawat/Documents/..."` is now an absolute path.
- **Mounts** `nginx.conf` into the container.
- **Container ID:** `"a9fa9f96d30..."` .

8. Checking All Docker Containers (Including Stopped Ones)

Command:

```
docker ps -a
```

Output:

```
CONTAINER ID   IMAGE                  STATUS
PORTS
NAMES
a9fa9f96d302   nginx
Exited (1) About a minute ago
load-balancer
e565ff4cf008   nginx
Up 6 minutes                   80/tcp
backend2
63d7914e78bb   nginx
Up 6 minutes                   80/tcp
backend1
34f41149b6a9   localstack/localstack
Up 17 minutes (healthy)        127.0.0.1:4566->4566
/tcp, 5678/tcp
localstack-main
```

Explanation:

- `load-balancer` **exited** → Configuration error.

- `"backend1"` and `"backend2"` → Still running.

9. Check Load Balancer Connection

```
docker exec -it load-balancer curl -I http://backend1
docker exec -it load-balancer curl -I http://backend2
```

Command Explanation:

- `docker exec -it load-balancer` → Runs a command inside the **load-balancer** container.
- `curl -I http://backend1` → Sends an HTTP `HEAD` request to `backend1` to check connectivity.
- `curl -I http://backend2` → Sends an HTTP `HEAD` request to `backend2` to check connectivity.

Expected Output:

```
HTTP/1.1 200 OK
Server: nginx/1.27.4
Date: Thu, 06 Mar 2025 05:30:04 GMT
Content-Type: text/html
Content-Length: 615
Last-Modified: Wed, 05 Feb 2025 11:06:32 GMT
Connection: keep-alive
ETag: "67a34638-267"
Accept-Ranges: bytes
```

Output Breakdown:

- `HTTP/1.1 200 OK` → The request was successful, and the server is responding.
- `Server: nginx/1.27.4` → The backend server is running **NGINX version 1.27.4**.
- `Date: Thu, 06 Mar 2025 05:30:04 GMT` → The timestamp when the response was generated.
- `Content-Type: text/html` → The response content is an **HTML page**.
- `Content-Length: 615` → The size of the response body is **615 bytes**.
- `Last-Modified: Wed, 05 Feb 2025 11:06:32 GMT` → The last modification timestamp of the content.
- `Connection: keep-alive` → The connection is persistent for multiple requests.
- `ETag: "67a34638-267"` → A unique identifier for the content version.
- `Accept-Ranges: bytes` → Supports **partial content requests**.

Issue: The expected response was not shown correctly during execution.

10. Build the Flask App Image

```
docker build -t my-flask-app .
```

Command Explanation:

- `docker build` → Builds a new Docker image.
- `-t my-flask-app` → Assigns the name **my-flask-app** to the built image.
- `.` → Uses the **current directory** as the build context.

Build Process Output:

```
[+] Building 18.4s (9/9) FINISHED                                docker:desktop-linux
 => [internal] load build definition from Dockerfile                              0.0s
 => => transferring dockerfile: 332B                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                0.0s
 => [internal] load .dockerignore                                                 0.0s
 => => transferring context: 2B                                                   0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim                                  0.0s
 => [internal] load build context                                                 0.0s
 => => transferring context: 1.35kB                                               0.0s
 => CACHED [2/4] WORKDIR /app                                                     0.0s
 => [3/4] COPY . .                                                                0.1s
 => [4/4] RUN pip install --no-cache-dir flask requests                          18.0s
 => exporting to image                                                            0.2s
 => => exporting layers                                                           0.1s
 => => writing image sha256:6e4cbd50721df2af76ffcc7bcf3ae2550a3b50008a9cb4212     0.0s
 => => naming to docker.io/library/my-flask-app                                   0.0s
```

Output Breakdown:

- `[+] Building 18.4s (9/9) FINISHED` → The image build process took **18.4 seconds** and completed successfully.
- `load build definition from Dockerfile` → Reads the **Dockerfile** in the current directory.
- `load metadata for docker.io/library/python:3.9-slim` → Retrieves the **Python 3.9 slim image** from Docker Hub.
- `WORKDIR /app` → Sets `/app` as the **working directory** inside the container.
- `COPY . .` → Copies all files from the local directory into the container.
- `RUN pip install --no-cache-dir flask requests` → Installs **Flask** and **Requests** without caching.
- `exporting to image` → Saves the built image.
- `naming to docker.io/library/my-flask-app` → The final image is tagged as `my-flask-app`.