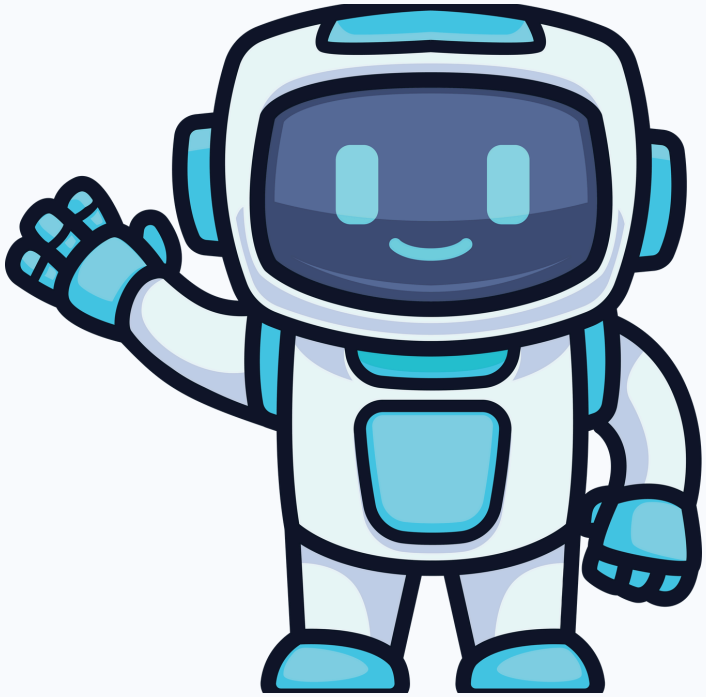
Deploy a NN model to AWS lambda through SAM deployment

LiveAi



Prerequisites

LiveAi

Prerequisites
For this exercise, you should have the following prerequisites:

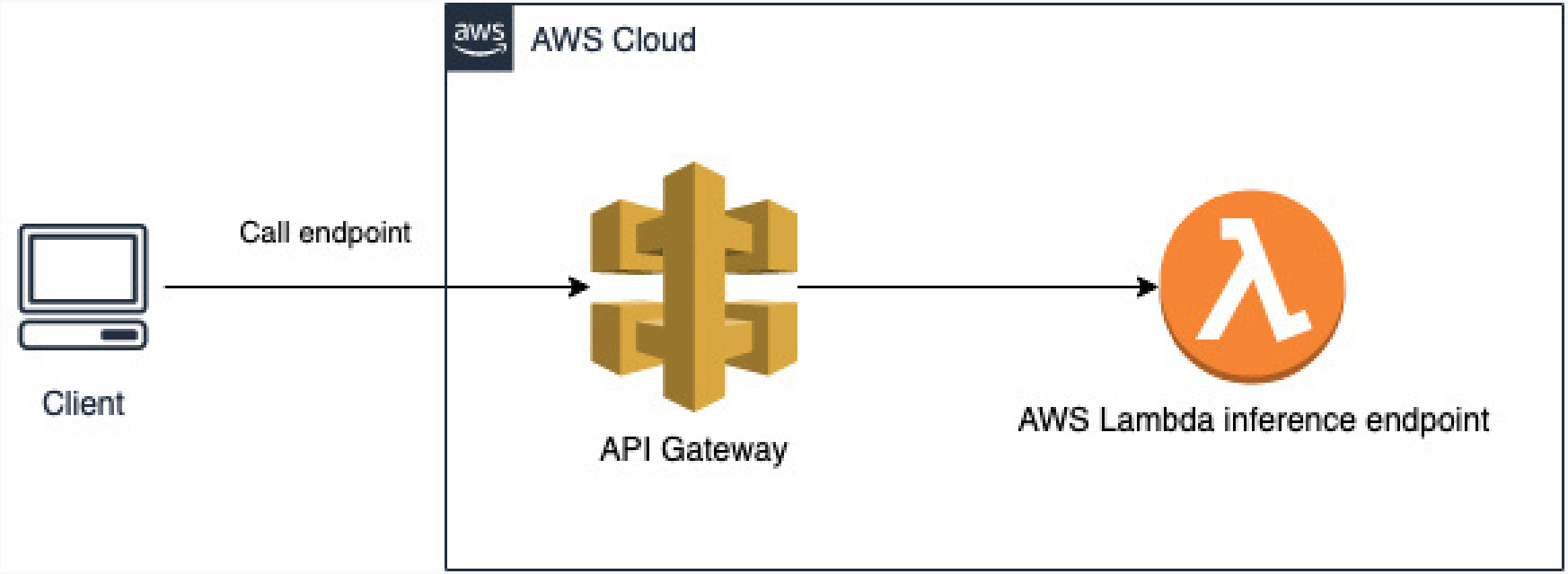
- **An AWS account**
- **The AWS Command Line Interface (AWS CLI) installed and configured to interact with AWS services locally**
- **The AWS Serverless Application Model (AWS SAM) CLI installed**
- **The Docker CLI**

Prerequisites

LiveAi

We use a simple LSTM based language model to classify sentences.

We use the AWS SAM CLI to create the serverless endpoint with an Amazon API Gateway. The following diagram illustrates our architecture.



Prerequisites

LiveAi

To implement the solution, complete the following steps:

1. On your local machine, run sam init.
2. Enter 1 for the template source (AWS Quick Start Templates)
3. Enter 1 for the Hello World Example.
4. For the runtime and package type enter N
5. For the python version enter 17 for python3.12.
6. As a package type, enter 2 for image.
7. Disable X-Ray and structured logging by selecting N

Alternatively, use the github repo provided here:

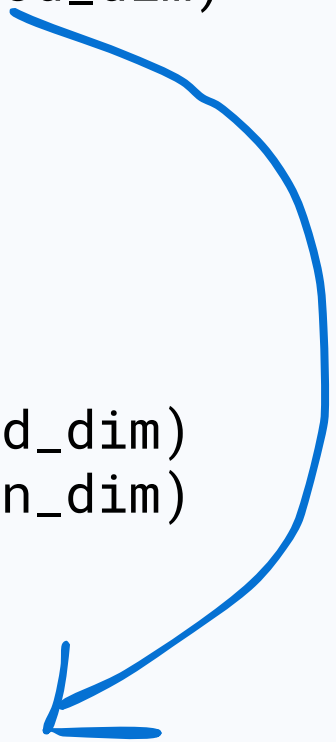
Folder structure

LiveAi

```
sam-test/
├── .aws-sam/
├── events/
├── hello_world/
│   ├── __init__.py
│   ├── app.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── text_classifier_weights.pth
├── tests/
│   ├── __init__.py
├── .gitignore
├── README.md
├── samconfig.toml
└── template.yaml
```

```
    class TextClassifier(nn.Module):
  def __init__(self, vocab_size, embed_dim, hidden_dim):
    super(TextClassifier, self).__init__()
    self.embedding = nn.Embedding(vocab_size, embed_dim)
    self.lstm = nn.LSTM(embed_dim, hidden_dim,
batch_first=True)
    self.fc = nn.Linear(hidden_dim, 2) # Binary
classification

  def forward(self, x):
    x = self.embedding(x) # (batch, seq_len, embed_dim)
    _, (hidden, _) = self.lstm(x) # (batch, hidden_dim)
    out = self.fc(hidden[-1]) # Output layer
    return out
```



nn.Embedding: Converts word indices into dense vectors.
nn.LSTM: Extracts sequential patterns.
nn.Linear: Maps LSTM output to 2-class logits.

Pre-processing and create a toy dataset (Here you should implement ur own code)

```
vocab = sorted(set(...))
word_to_idx = {word: idx for idx, word in enumerate(vocab)}

def encode_sentence(sentence):
#Converts a sentence into a list of integer indices using the
vocabulary.
    return [word_to_idx[word] for word in sentence.split()]
```

Step 1: aws-configure (temporarily in powershell)

LiveAi

```
aws configure list

# list of all the environments
# We will start afresh, so remove all existing credentials.

Remove-Item Env:\AWS_ACCESS_KEY_ID
Remove-Item Env:\AWS_SECRET_ACCESS_KEY
Remove-Item Env:\AWS_SESSION_TOKEN
Remove-Item Env:\AWS_REGION
Remove-Item Env:\AWS_DEFAULT_REGION
Remove-Item "$env:USERPROFILE\.aws\credentials"
Remove-Item "$env:USERPROFILE\.aws\config"
```

Step 1: aws-configure (temporarily in powershell)

LiveAi

```
aws configure list
```

```
# run again this will show empty list
    Name            Value       Type  Location
    ----            -----       ----  --------
   profile          <not set>       None  None
access_key           <not set>       None  None
secret_key           <not set>       None  None
   region          <not set>       None  None
```

Step 1: aws-configure (temporarily in powershell)

LiveAi

```
aws configure list
```

```
# run again this will show empty list
    Name            Value       Type  Location
    ----            -----       ----  --------
   profile          <not set>       None  None
access_key           <not set>       None  None
secret_key           <not set>       None  None
   region          <not set>       None  None
```

aws-configure (temporarily in powershell)

LiveAi

```
aws configure

# configure again

AWS Access Key ID [None]: <Your Key ID>
AWS Secret Access Key [None]: <Your Secrect Access Key>
Default region name [None]: us-east-1
Default output format [None]:
```

Step 2: Modify Lambda Function (For AWS Deployment)

```
def lambda_handler(event, context):
  body = json.loads(event["body"])
```

Designed to run in AWS Lambda environment (serverless function).
Input: An event with JSON body containing a "sentence".

```
file_path = "model/text_classifier_weights.pth" # Change this
to your file path

  cuda_available = torch.cuda.is_available()
  device = torch.device("cpu")

  try:

   .....
```

```
 try:

 model = TextClassifier(vocab_size, embed_dim, hidden_dim) #
Instantiate the model
 model.load_state_dict(torch.load(file_path,
map_location=torch.device("cpu")))
 model.eval()
# Preprocess and encode the sentence. Converts input string
into tensor for inference.
    encoded_sentence =
torch.tensor(encode_sentence(sentence), dtype=torch.long)
# Add batch dimension
    input_tensor = encoded_sentence.clone().detach()
    output = model(input_tensor)
    predicted_label = torch.argmax(output).item()

#if the model predicts the class 1 response will be
{"statusCode": 200, "body": "The prediction is class 1!"}
```

Step3: DockerFile

LiveAi

```
# Pull the base image with python 3.12 as a
runtime for your Lambda

FROM public.ecr.aws/lambda/python:3.12
```

Purpose: Starts from an official AWS Lambda base image with Python 3.12 installed.

This image is optimized to run Python functions as AWS Lambda expects.

```
# Copy the earlier created requirements.txt file
to the container
COPY requirements.txt ./
```

Purpose: Adds your requirements.txt (which lists Python dependencies) into the Docker image.

DockerFile explaination

LiveAi

```
# Install the python requirements from requirements.txt
RUN python3.12 -m pip install -r requirements.txt
```

Purpose: Installs all dependencies inside the container using pip.

```
RUN mkdir model
```

Purpose: Creates a model directory inside the container to store model files or other assets.

```
# Copy the earlier created app.py file to the container
COPY app.py ./
COPY text_classifier_weights.pth ./model
```

Purpose: Adds your main Python script (app.py) and the pre-trained model file (text_classifier_weights.pth) into the image.

DockerFile explaination

LiveAi

```
# Set the CMD to your handler
CMD ["app.lambda_handler"]
```

Purpose: Tells AWS Lambda which function to invoke.

Step 5: build and deploy app

LiveAi

sam build

```
Building codeuri: C:\Users\mtann\Documents\Deployment\sam-
test runtime: None architecture: x86_64 functions:
pytorchEndpoint
Building image for pytorchEndpoint function
Setting DockerBuildArgs for pytorchEndpoint function
Step 1/7 : FROM public.ecr.aws/lambda/python:3.12
 ---> 71ade3c7678a
Step 2/7 : COPY requirements.txt ./
 ---> Using cache
 ---> 2bf4920fcd6a
Step 3/7 : RUN python3.12 -m pip install -r requirements.txt
 ---> Using cache
 ---> 5305e7df0261
```

build and deploy app

LiveAi

```
Step 4/7 : RUN mkdir model
 ---> Using cache
 ---> b74abe0a5208
Step 5/7 : COPY app.py ./
 ---> Using cache
 ---> d9e5b1a2b3d5
Step 6/7 : COPY text_classifier_weights.pth ./model
 ---> Using cache
 ---> d27a0511cc17
Step 7/7 : CMD ["app.lambda_handler"]
 ---> Using cache
 ---> a1c492eeda6f
Successfully built a1c492eeda6f
Successfully tagged pytorchendpoint:python3.12-v1
```

build and deploy app

LiveAi

sam deploy --guided

```
Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-test]:
        AWS Region [us-east-1]:
        #Shows you resources changes to be deployed and
require a 'Y' to initiate deploy
        Confirm changes before deploy [Y/n]: Y
        #SAM needs permission to be able to create roles to
connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]: Y
```

build and deploy app

LiveAi

```
#Preserves the state of previously provisioned resources when
an operation fails
        Disable rollback [Y/n]: n
        pytorchEndpoint has no authentication. Is this okay?
[y/N]: y
        Save arguments to configuration file [Y/n]:
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]:

        Looking for resources needed for deployment:

        Managed S3 bucket: aws-sam-cli-managed-default-
samclisourcebucket-xjrnuspzi6v9
        A different default S3 bucket can be set in
samconfig.toml and auto resolution of buckets turned off by
setting resolve_s3=False
File with same data already exists at sam-
test/e4d72c74316337c1fec3d1bfd64a0dda.template, skipping
upload
```

build and deploy app

LiveAi

```
Parameter "stack_name=sam-test" in
[default.deploy.parameters] is defined as a global parameter
[default.global.parameters].
        This parameter will be only saved under
[default.global.parameters] in
C:\Users\mtann\Documents\Deployment\sam-test\samconfig.toml.

        Saved arguments to config file
        Running 'sam deploy' for future deployments will use
the parameters saved above.
        The above parameters can be changed by modifying
samconfig.toml
        Learn more about samconfig.toml syntax at
        https://docs.aws.amazon.com/serverless-application-
model/latest/developerguide/serverless-sam-cli-config.html
```

👉 Subscribe to our YouTube channel: **@liveaiAIClub**

🚀 Let's learn, build, and grow—together!