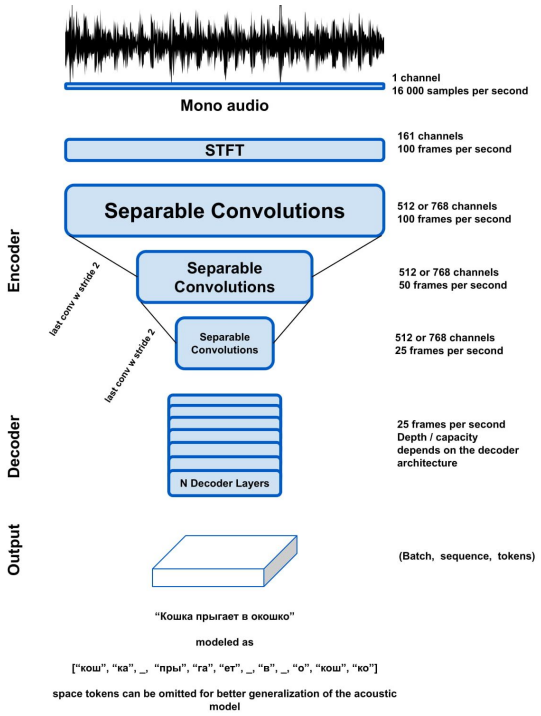
Goal

- Taking voice data as an input -> Using the Speech recognition model to output a text.

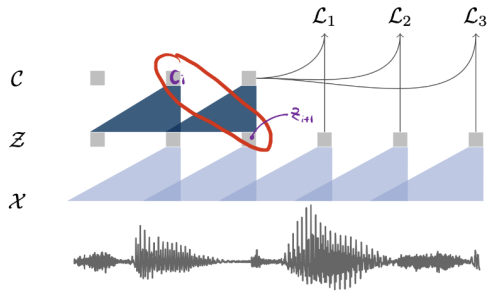
AWS Account: (for your account login details)

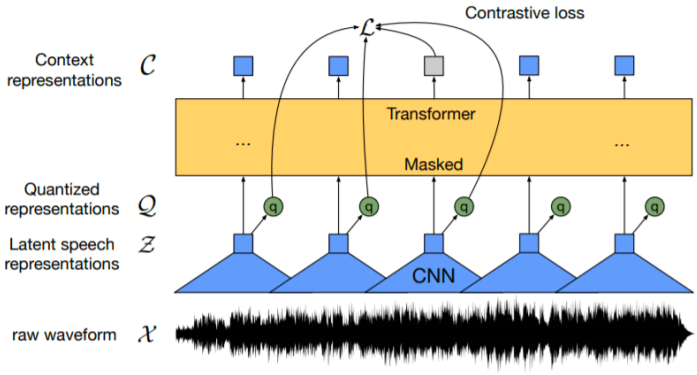
Models to Consider:

- Deep Speech by Mozilla DeepSpeech
  - https://github.com/mozilla/DeepSpeech
  - https://colab.research.google.com/github/tugstugi/dl-colab-notebooks/blob/master/notebooks/MozillaDeepSpeech.ipynb#scrollTo=fZwM0GtmA7mX
- **Silero-Models**



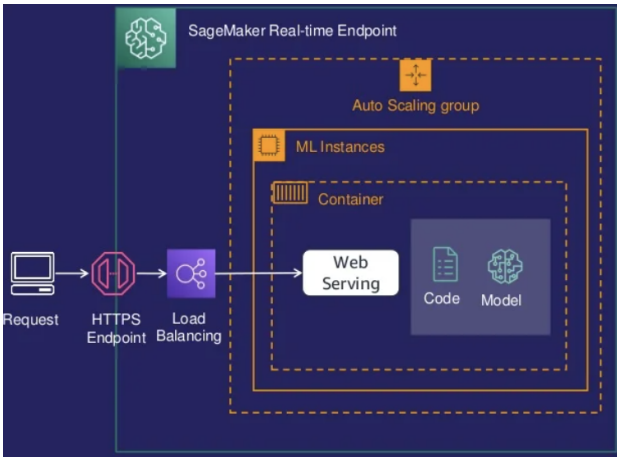

  - https://github.com/snakers4/silero-models#pytorch
  - https://huggingface.co/spaces/pytorch/silero_stt
  - Pre-trained enterprise-grade STT and TTS models.
- **Wav2Vec2(we chose this)**
  - Pretrained/Fine-Tuned 960hrs of Librispeech on 16kHz.
  - Hugging Face/deploy ready
  - Encoder Network/Context Network

- 

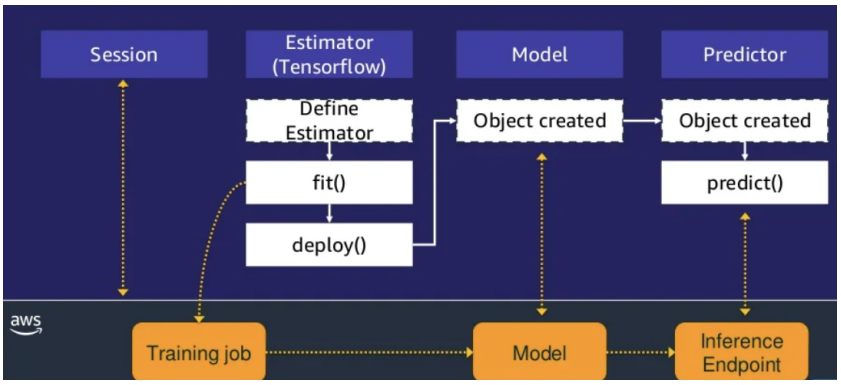
- 


AWS service

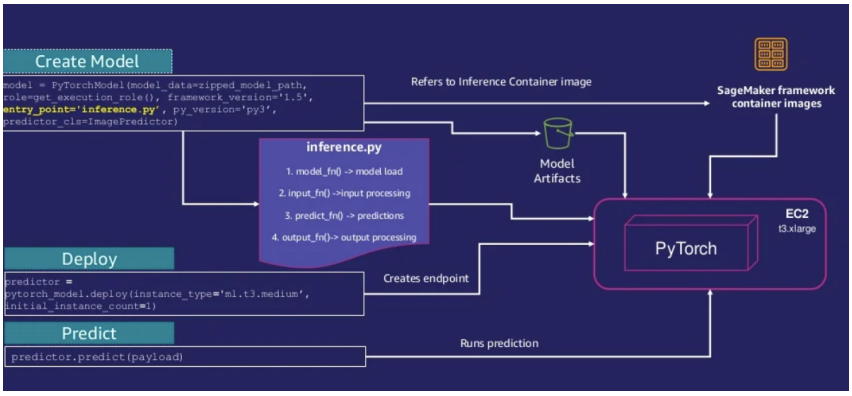
- EC2(Notebook Instances)
  - Computing service that enables applications to run on AWS
  - Web-based service that supports scalable computing power on Amazon Web services
  - AWS refers to servers as EC2 instances.
    - Can spin up(boot up) virtually limitless VMs in the cloud, instead of connecting own computing hardware
    - Could adjust computing capacity(Scaling EC2 Instances)
    - Could choose operating systems (Linux, Windows, Mac, Custom OS)
    - Auto Scaling Groups
      - Flexible hardware adjustments(CPU,RAM,and Storage) depends on the demand
    - Do not share memory between instances
    - Only Keep data when they are running
      - Elastic Block Store to make instances more durable
      - Backup EC2 data to S3
- ECS
  - AWS services that orchestrates Docker containers
  - Elastic Container Services
  - Allows you to focus on building applications than maintaining the infrastructure on which it will run

    - Resources are scalable like EC2
        - Clusters are on-demand, not like EC2's scaling
    - Supports Docker Containers
    - ECS anywhere
        - Run, secure, and scale Docker containers on customer-managed infrastructure
        - AWS engineers manage the backend.
- S3
    - Scalable cloud storage services for object storage
    - The object comprises data, metadata, and its assigned key.
    - Store, retrieve, and back up data from anywhere, anytime
    - Allows to store archived data for years at a lower cost than EBS and EFS
- Amazon Lambda
    - Serverless computing on Amazon Web Services
    - Event-driven service uses serverless architecture to run applications.
        - Serverless, no need to manage runtimes, servers, or clusters
    - Lambda function
        - Code you run in AWS lambda
            - Could be created, called, and managed with AWS management console, AWS cloud formation, AWS serverless application Models, AWS SDKs, and AWS Command Line interface
        - Usually a containers
            - Configuration information that relates to that function
            - Function name/description, resource requirements, and entry point
- Deployment Baseline(Single Model)
    - 

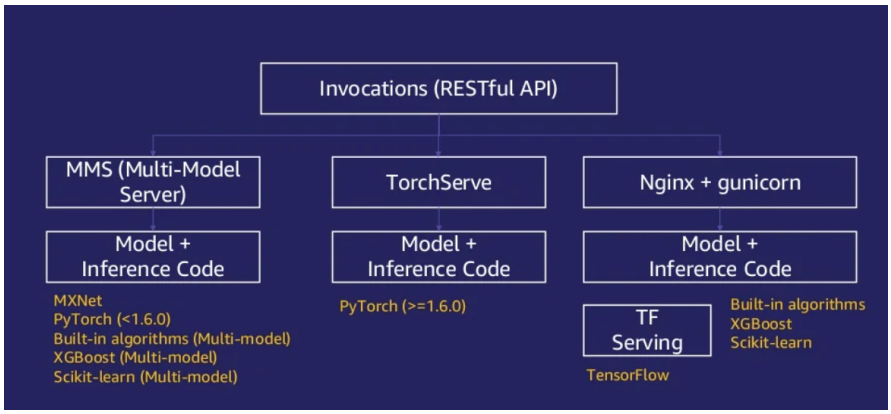
    - Save all artifacts that are required for inference on web server
    - Instant Response for Max 6MB payload

    - 60 Seconds Time Out, Auto Scaling
- End Point Generations Steps for pretrained model (AWS Command Line Interface)
    - Model Generation
        - 
```
aws sagemaker create-model
  --model-name model1
  --primary-container '{"Image": "123.dkr.ecr.amazonaws.com/algo",
                        "ModelDataUrl": "s3://bkt/model1.tar.gz"}'
  --execution-role-arn arn:aws:iam::123:role/me
```
    - Endpoint Configuration
        - 
```
aws sagemaker create-endpoint-config
  --endpoint-config-name model1-config
  --production-variants '{"InitialInstanceCount": 2,
                          "InstanceType": "ml.m4.xlarge",
                          "InitialVariantWeight": 1,
                          "ModelName": "model1",
                          "VariantName": "AllTraffic"}'
```
    - End Point
        - 
```
aws sagemaker create-endpoint
  --endpoint-name my-endpoint
  --endpoint-config-name model1-config
```
- SageMaker SDK
    - Model(Define Model object, connects model artifact with a container that is used for serving, and execute Deployment)  -> FrameworkModel (Define ML framework model, Hosted with S3)
    - 

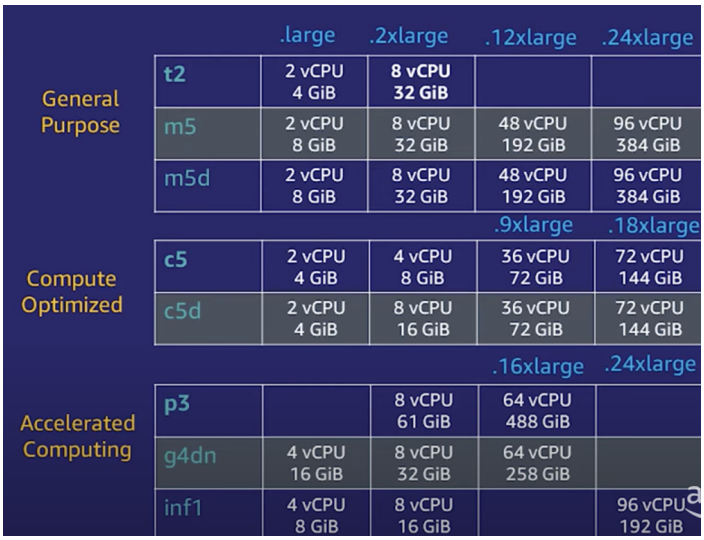
SageMaker SDK Endpoint




SageMaker Built-in Web server




ML instances in SageMaker:

| | | .large | .2xlarge | .12xlarge | .24xlarge |
|---|---|---|---|---|---|
| General Purpose | t2 | 2 vCPU 4 GiB | **8 vCPU 32 GiB** | | |
| | m5 | 2 vCPU 8 GiB | 8 vCPU 32 GiB | 48 vCPU 192 GiB | 96 vCPU 384 GiB |
| | m5d | 2 vCPU 8 GiB | 8 vCPU 32 GiB | 48 vCPU 192 GiB | 96 vCPU 384 GiB |
| | | | | .9xlarge | .18xlarge |
| Compute Optimized | c5 | 2 vCPU 4 GiB | 4 vCPU 8 GiB | 36 vCPU 72 GiB | 72 vCPU 144 GiB |
| | c5d | 2 vCPU 4 GiB | 8 vCPU 16 GiB | 36 vCPU 72 GiB | 72 vCPU 144 GiB |
| | | | | .16xlarge | .24xlarge |
| Accelerated Computing | p3 | | 8 vCPU 61 GiB | 64 vCPU 488 GiB | |
| | g4dn | 4 vCPU 16 GiB | 8 vCPU 32 GiB | 64 vCPU 258 GiB | |
| | inf1 | 4 vCPU 8 GiB | 8 vCPU 16 GiB | | 96 vCPU 192 GiB |

Batch Transform

- Inference on the entire dataset
- Suitable for big-dataset, periodic inference
- Transient Resources -> Pay for the amount that was used

Asynchronous Inference Endpoint

- Max 1GB payload
- Max 15 minutes time-out
- Auto-Scaling
- Computer Vision/Natural Language Processing

Lambda Serverless Inference

- Docker Image/Lambda function Required

SageMaker Serverless Inference

- Inference Image/ML Model

Using TorchServe to Deploy model in sagemaker

- Host server locally on Amazon EC2
- Host server locally on AMazon EC2/Amazon EKS and AWS Deep Learning Container
- Fully managed endpoint with Amazon SageMaker

Model Archive File

- Model definition, state definition end with .mar
- Created with TorchScrpt or Eager mode

Invoke Management API

- Register model
- Scale Workers
- Set model version

EndPoint

- Allows users to invoke and return the result of an inference

Customer Side-> Runnable Inference API

Deploy pre-trained model:

Sagemaker -> Create Notebook Instance

→Upload notebook -> Upload models->

Converts the models into Tarbell file(AWS format) ->

Move the tar files to S3 bucket ->

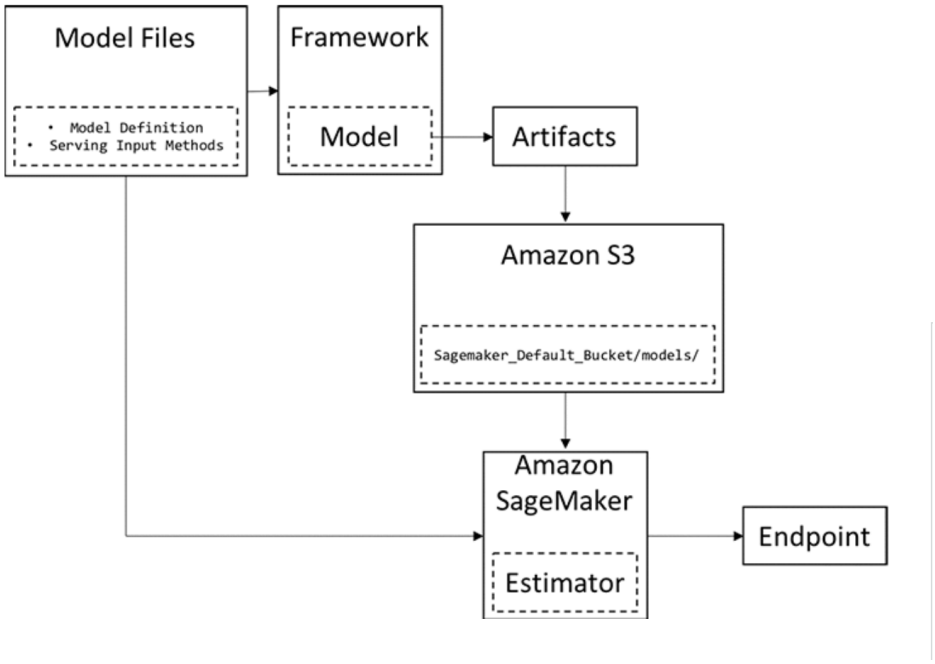
entry-point = training script, needed so create an empty file for a pre-trained model

->Create an Endpoint to access (predictor = sagemaker.deploy(~) ->

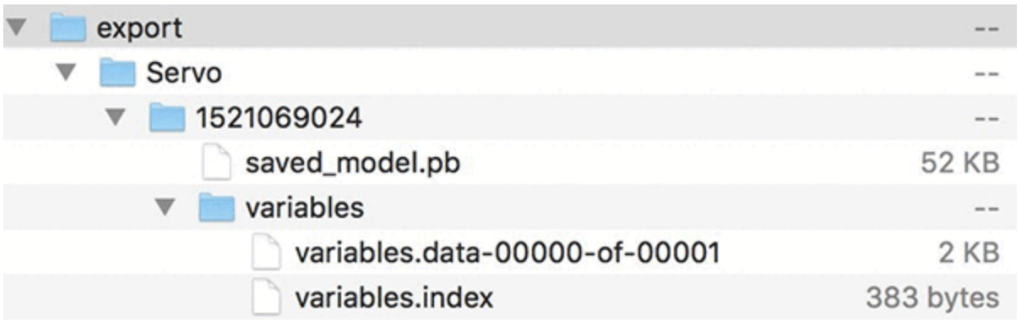
setting endpoint = predictor.endpoint->

Check if it is properly deployed by making a prediction

Pre-trained Tensorflow Models into SageMaker




- Deploying from model artifacts
  - Format
    - Model = TensorFlowModel(model data= s3 buckets where model artifact is stored, then setting role)
    - Create an endpoint by predictor = model.deploy(setting instance count, instance type)
- Available for Pre/Post-processing
  - Setting entry point = your own “inference.py”
    - Requires input_handler and output_handler
    - Or handler than using function as _process_input and _process_output
- The model needs to be exported in a way that where sagemaker could interpret (Model-Artifact:tar.gz)
- TF model exports need a serving input-> provide an input pipeline to the exported model when deployed.
- TF builder is provided to create the serving function

- Hierarchy
  - 


- Model Artifact has to be placed on an S3 bucket.
- S3 -> then be able to be deployed as an endpoint
- TF Model Artifacts

```
from sagemaker.tensorflow.model import TensorFlowModel
sagemaker_model = TensorFlowModel(model_data = 's3://' +
sagemaker_session.default_bucket() + '/model/model.tar.gz',
                                  role = role,
                           entry_point = 'classifier.py')
```

- - Deploy command from the API to host the model

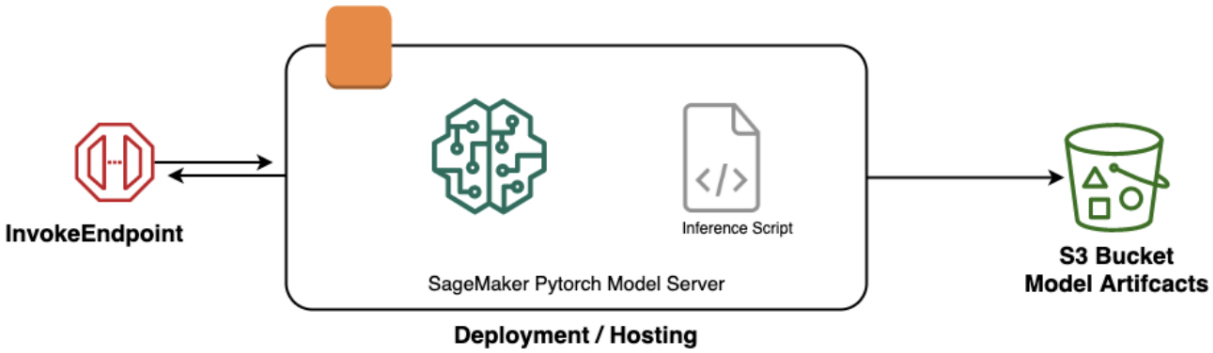
```
predictor =
sagemaker_model.deploy(initial_instance_count=1,instance_type='ml.m4.xla
rge')
```

- Predict with

```
sample = [6.4,3.2,4.5,1.5]
predictor.predict(sample)
```

-

Pytorch Model




- Upload model artifacts (tar file)
- Generate Inference Handler Script
    - Must create an inference script that implements (at least) the model function that calls the loaded model to get prediction.
    - To process the requests properly on the loaded model

```
def model_fn(model_dir):
    pass


def input_fn(request_body, request_content_type):
    pass


def predict_fn(input_data, model):
    pass


def output_fn(prediction, content_type):
    pass
```

    - 
    - Model_fn
        - Loads the models
    - Input_fn
        - Takes request data/deserializes into object ready for prediction
        - Preprocessing the data/ returns
        - Two Arguments when Endpoint is Invoked
            - Body and Content
    - Predict_fn
        - Takes preprocessed input from Input_fn and performs Inference
    - Output_fn

        - Serializes the output of the prediction according to the response content type/ maps the predicted outputs to classes
- Deploy

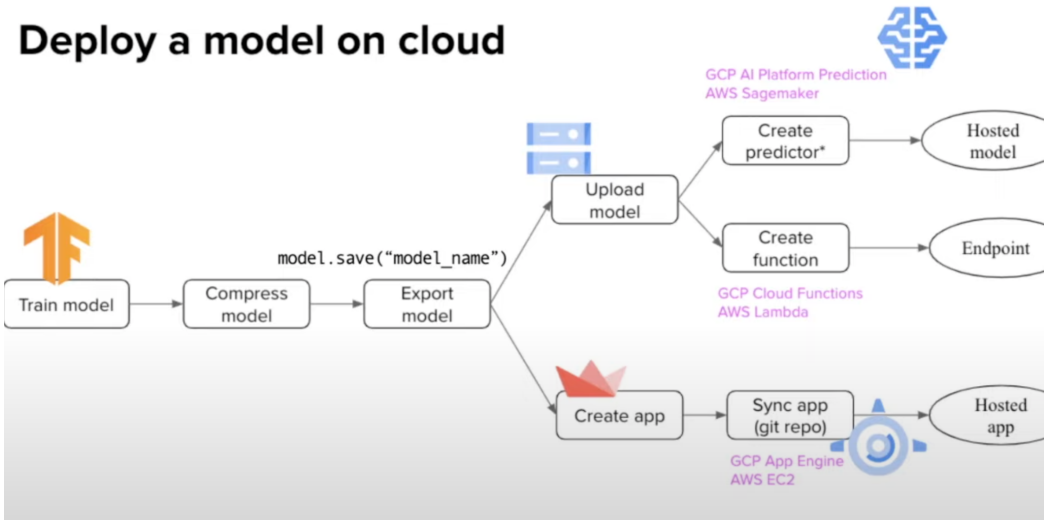
```
#filename deploy.ipynb

from sagemaker.pytorch import PyTorchModel
from sagemaker import get_execution_role

role = get_execution_role()

# You can also configure a sagemaker role and reference it by its
name.
# role = "CustomSageMakerRoleName"

pytorch_model = PyTorchModel(model_data='s3://pytorch-sagemaker-
example/model.tar.gz', role=role, entry_point='inference.py',
framework_version='1.3.1')

predictor = pytorch_model.deploy(instance_type='ml.t2.medium',
initial_instance_count=1)
```

  - Same directory as inference

Hugging Face

- SageMaker Friendly.
- Deep Learning Containers
  - Docker images are pre-installed with deep learning frameworks and libraries.
  - Allow yo
- Easy to deploy a model.
- Directly loading models from the HF hub
- Both training and inference are possible
  - Train
    - Hugging Face Estimator /Estimator.fit(start the training job with uploaded datasets as input)
      - SageMaker Studio
      - SageMaker Notebook Instances
      - Local-Environment
  - Deploy
    - Deploy with model data
      - Specify the location of the trained Sagemaker model(tar.gz file)
      - Create a model artifact from Hugging Face
        - Clone the git files (Models you want to use)
        - Move to the repo you downloaded (cd [repo])

        - Generate tar file
            - tar zcvf model.tar.gz*
        - Upload model to S3
- Deploy directly from the hub by defining environmental variables
    - HF_MODEL_ID that defines the model
    - HF_TASK that defines the task
    - Create Model Class by
        - Setting environment
        - Configuring IAM role
        - Specify transformer, PyTorch, and  python version
    - Deploy for an Inference
        - Predictor
            - Setting instance count
            - Instance type
    - Example for Request
        - Define the inputs
            - Calling Request with predictor




**Connecting Web Application**

Wav2Vec Deployment

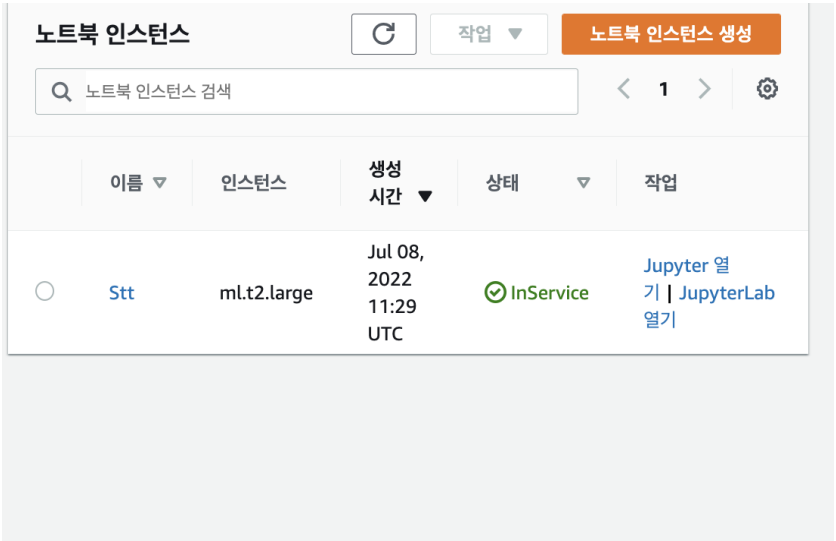
- Used the Wav2Vec2-Base-960h models (Huggingface)

```
assert sagemaker.__version__ >= "2.86.0"
```

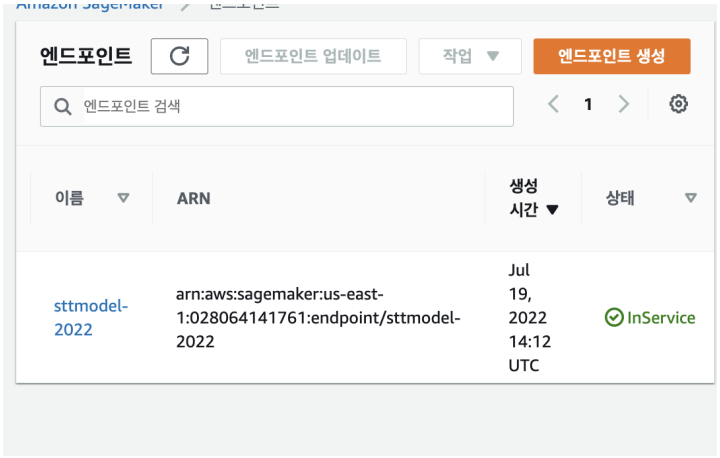
[1]:

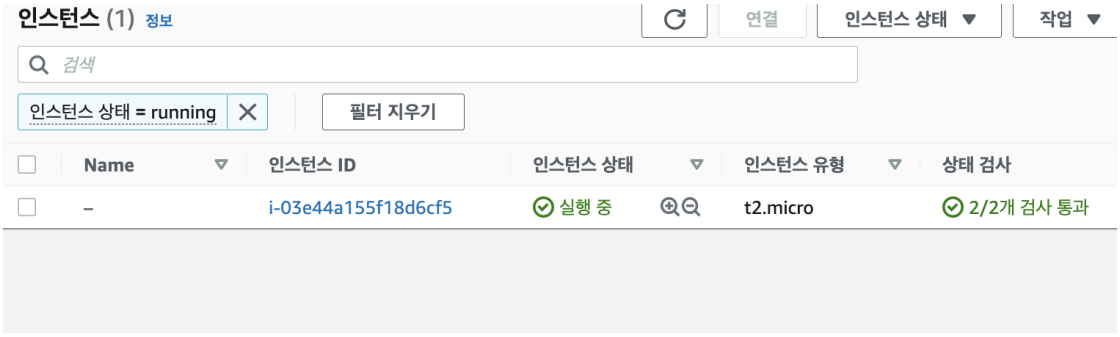
```
import sagemaker
import boto3
sess = sagemaker.Session()
sagemaker_session_bucket=None
if sagemaker_session_bucket is None and sess is not None:
    sagemaker_session_bucket = sess.default_bucket()

try:
    role = sagemaker.get_execution_role()
except ValueError:
    iam = boto3.client('iam')
    role = iam.get_role(RoleName='sagemaker_execution_role')['Role']['Arn']

sess = sagemaker.Session(default_bucket=sagemaker_session_bucket)
from sagemaker.huggingface.model import HuggingFaceModel
from sagemaker.serializers import DataSerializer

hub = {
   'HF_MODEL_ID':'facebook/wav2vec2-base-960h',
    'HF_TASK':'automatic-speech-recognition',
}
huggingface_model = HuggingFaceModel(
   env=hub,                      # configuration for loading model from Hub
   role=role,                    # iam role with permissions to create an Endpoint
   transformers_version="4.17",  # transformers version used
   pytorch_version="1.10",        # pytorch version used
   py_version='py38',            # python version used
)
audio_serializer = DataSerializer(content_type='audio/x-audio') # using x-audio to support mu

predictor = huggingface_model.deploy(
    initial_instance_count=1, # number of instances
    instance_type='ml.g4dn.xlarge', # ec2 instance type
    endpoint_name = 'sttmodel-2022',
    serializer=audio_serializer, # serializer for our audio data.
)
```

Deployment Snippets:

- Notebook Instance
  - 


- Endpoint

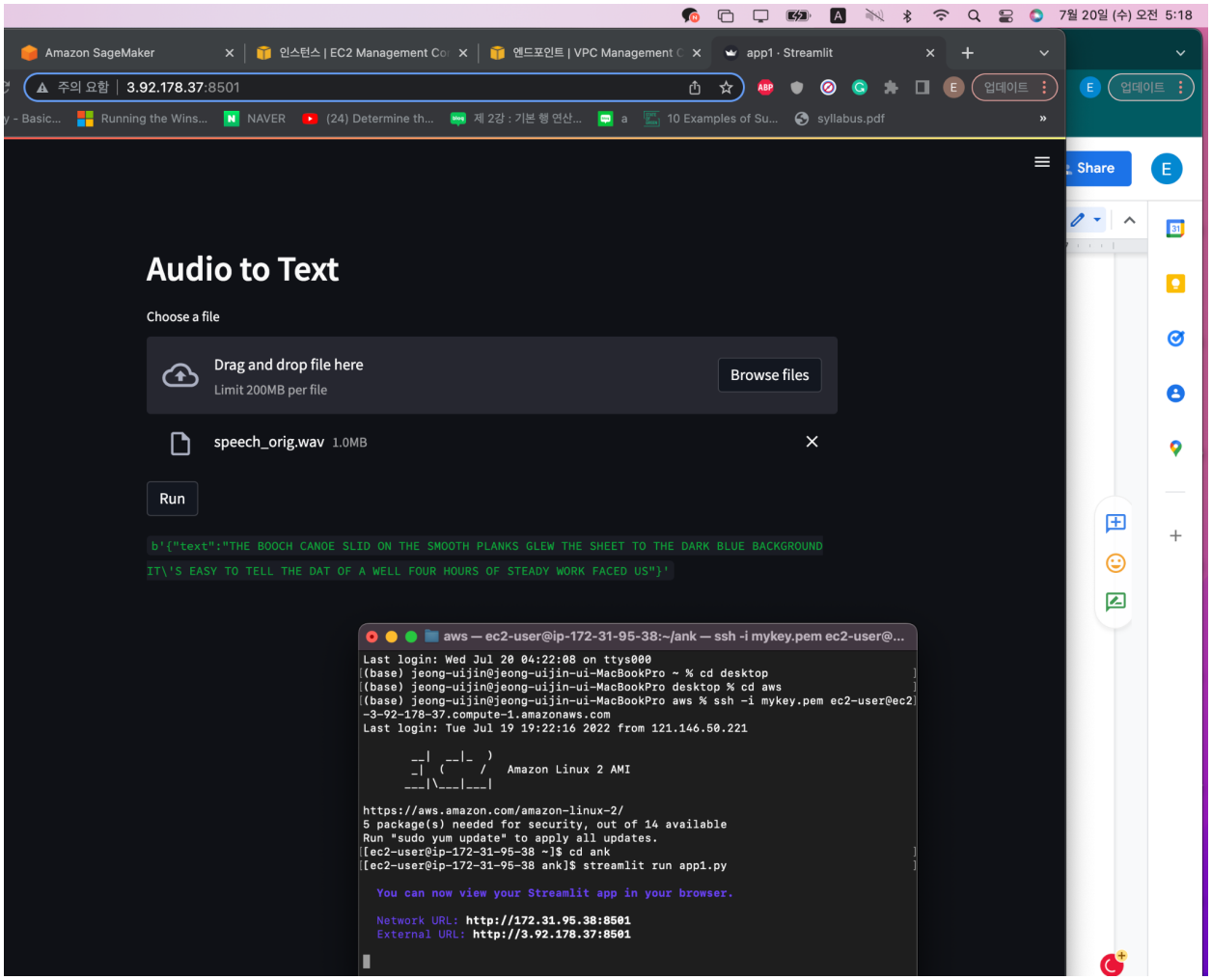
- 


- EC2 Instance (Hosting)
  - 

WebPage (Using Streamlit)

- Code

```python
import streamlit as st
import boto3
import json


session = boto3.Session(
    aws_access_key_id="ASIAQNCGA2HA3WD7ZT4I",
    aws_secret_access_key="WPJbYzJsbiKmcNM6WerwPjbXTZZFpkJayrSVpg2B",
    aws_session_token="IQoJb3JpZ2luX2VjENP//////////wEaCXVzLWVhc3QtMSJIMEYCIQC5pS3g
sagemaker_runtime = session.client('sagemaker-runtime', region_name="us-east-1")

endpoint_name='sttmodel-2022'

def generate_text(audio):
    payload = audio
    response = sagemaker_runtime.invoke_endpoint(
                                EndpointName=endpoint_name,
                                ContentType='audio/x-audio',
                                Body=payload
                                        )
    result = response['Body'].read()
    text = result
    return text


st.header("Audio to Text")
audio = st.file_uploader("Choose a file")

if st.button("Run"):
    generated_text = generate_text(audio)
    st.write(generated_text)
```


7월 20일 (수) 오전 5:18
Amazon SageMaker
인스턴스 | EC2 Management Co
엔드포인트 | VPC Management C
app1 · Streamlit
주의 요함 | 3.92.178.37:8501
업데이트
Running the Wins...
NAVER
(24) Determine th...
제 2강 : 기본 행 연산...
a
10 Examples of Su...
syllabus.pdf
Share
Audio to Text
Choose a file
Drag and drop file here
Limit 200MB per file
Browse files
speech_orig.wav 1.0MB
Run
b'{"text":"THE BOOCH CANOE SLID ON THE SMOOTH PLANKS GLEW THE SHEET TO THE DARK BLUE BACKGROUND IT\'S EASY TO TELL THE DAT OF A WELL FOUR HOURS OF STEADY WORK FACED US"}'
aws — ec2-user@ip-172-31-95-38:~/ank — ssh -i mykey.pem ec2-user@...
Last login: Wed Jul 20 04:22:08 on ttys000
(base) jeong-uijin@jeong-uijin-ui-MacBookPro ~ % cd desktop
(base) jeong-uijin@jeong-uijin-ui-MacBookPro desktop % cd aws
(base) jeong-uijin@jeong-uijin-ui-MacBookPro aws % ssh -i mykey.pem ec2-user@ec2-3-92-178-37.compute-1.amazonaws.com
Last login: Tue Jul 19 19:22:16 2022 from 121.146.50.221
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
5 package(s) needed for security, out of 14 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-95-38 ~]$ cd ank
[ec2-user@ip-172-31-95-38 ank]$ streamlit run app1.py
You can now view your Streamlit app in your browser.
Network URL: http://172.31.95.38:8501
External URL: http://3.92.178.37:8501