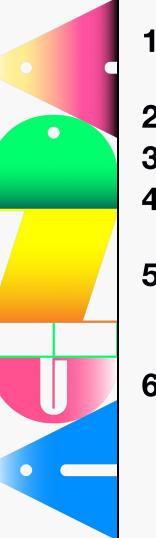
## **AWS SageMaker in Production**

Mitja Hofer





- What is AWS SageMaker?
- 2. Why SageMaker?
- 3. Deploying models
- 4. Bring your own model
- 5. Deploying into production environment
- 6. Local development

# What is AWS SageMaker?

Cel<sup>·</sup>

- A suite of tools for discovery, building, training and serving machine learning models
- SageMaker Studio
- Model and endpoint development and deployment
- Model serving endpoints

## Why do we use AWS SageMaker?

- The L3 step of Al initiative, creative scoring, requires a real-time inference model server
- An alternative to building our own Python environment with a http server, an inference model server, all supporting code, a testing and deployment pipeline.
- Support for GPU instances
- SageMaker allows for training of our model to be implemented there as well.
- SageMaker SDK

## Deploying models on AWS SageMaker

Ce

- Real-time inference
  - A managed rest API endpoint with support for payloads up-to 6MB and 60s processing time
- Serverless inference
  - For irregular traffic, 4MB payloads, 60s processing time
- Asynchronous inference
  - Queue based processing. Up to 1GB payloads and 1h processing time
- Batch transform
  - Offline processing of workloads large datasets and days of processing time

### Bring your own model (or container)

- You may create your own model in a Python machine learning framework of your choice and supply it to SageMaker
- You need to implement your inference codebase
- inference.py implements a chain of python functions
- First you need to load your model
- Then the serving is broken down into three steps:
  - Input processing
  - Prediction
  - Output processing

## inference.py

- model\_fn(model\_dir: str)
  - model\_dir is the TargetModel parameter of the
    InvokeEndpoint API call
  - It respects the SAGEMAKER\_SUBMIT\_DIRECTORY environment variable, that can point to local filesystem or S3 path

#### inference.py

- input\_fn(request\_body, request\_content\_type)
  - o Takes in a HTTP request and deserializes it into an object
- predict\_fn(input\_object, model)
  - gets the object and performs inference against the loaded model
- output\_fn(prediction, response\_content\_type)
  - takes the prediction and forms a HTTP response

## Single vs Multi Model endpoints

- Real-time inference supports single and multi-model endpoints
- .tar.gz models
- Single model endpoint may still load multiple model files if you implement inference.py that way
- Multi model endpoint will dynamically load and offload models based on usage - Multi Model Server
- The user specifies the S3 location of the model files
- InvokeEndpoint API, TargetModel parameter
  - Allows uploading new model files without even touching SageMaker
- Similarly-sized models work best

- Three constructs of a SageMaker model deployment
  - Model
    - Specifies the inference script and model files location
    - Specifies the role that SageMaker runner assumes
    - Environment variables
  - EndpointConfiguration
    - Specifies the Model
    - Instance type, autoscaling configuration, routing strategy
  - Endpoint
    - Specifies the DeploymentStrategy
    - Serves your requests

- SageMaker SDK for Python for simple model development and deployment
- The Model() constructor takes the endpoint name parameter a hard coupling between the model and the endpoint
- No option to update endpoint
- You are expected to have model files on S3, but SageMaker SDK reuploads the files with correct permissions to a new bucket

- SageMaker SDK hides away the three constructs and so you cannot update your endpoint, you must always recreate it
- What about AWS CDK?
- L1 vs L2 constructs
- Fits into infrastructure code
- Define Cloudwatch alarms for scaling and model monitoring

- Deploying inference code and model files using AWS CDK
- The instance that is part of the Model construct is long-lived
- Model files are loaded on demand
- Inference code is loaded on creation
- Two options to upload files to S3 using CDK:
- S3.Asset
  - Single file, named after the content digest
- BucketDeployment
  - Respects directory structure, may keep old files on S3

What about local development?

## What about local development?

Did you know that you can simply redirect any AWS SDK client to any domain you want? new SageMakerRuntimeClient({

endpoint: 'foo.test'

- This will make all requests from the SDK go to foo.test
- Local development?

## What about local development?

- SageMaker Model is just a managed instance for which you specify the container image it runs
- Add the same image that SageMaker uses in the cloud to your docker-compose.yml
- Bind mount inference.py and model files
- Add a reverse proxy to correctly handle model registration and rewriting of requests (headers)
- As SageMaker uses Multi Model Server in the background, you need to understand how it works

#### What we did

- Use AWS SageMaker for serving a in-house inference model
- Use a multi-model realtime inference endpoint
- Write our own inference code that loads the model, handles http requests and calls the model
- Use AWS CDK to create the three constructs needed for model serving, and upload to S3 in two different ways
- Support local development by using the same SageMaker container we are using in production