Bring Your Own Container With Amazon SageMaker

The Most Flexible Real-Time Inference Option Within SageMaker

Ram Vegiraju

Published in

Towards Data Science

·

7 min read

·

Nov 13, 2021

105

3

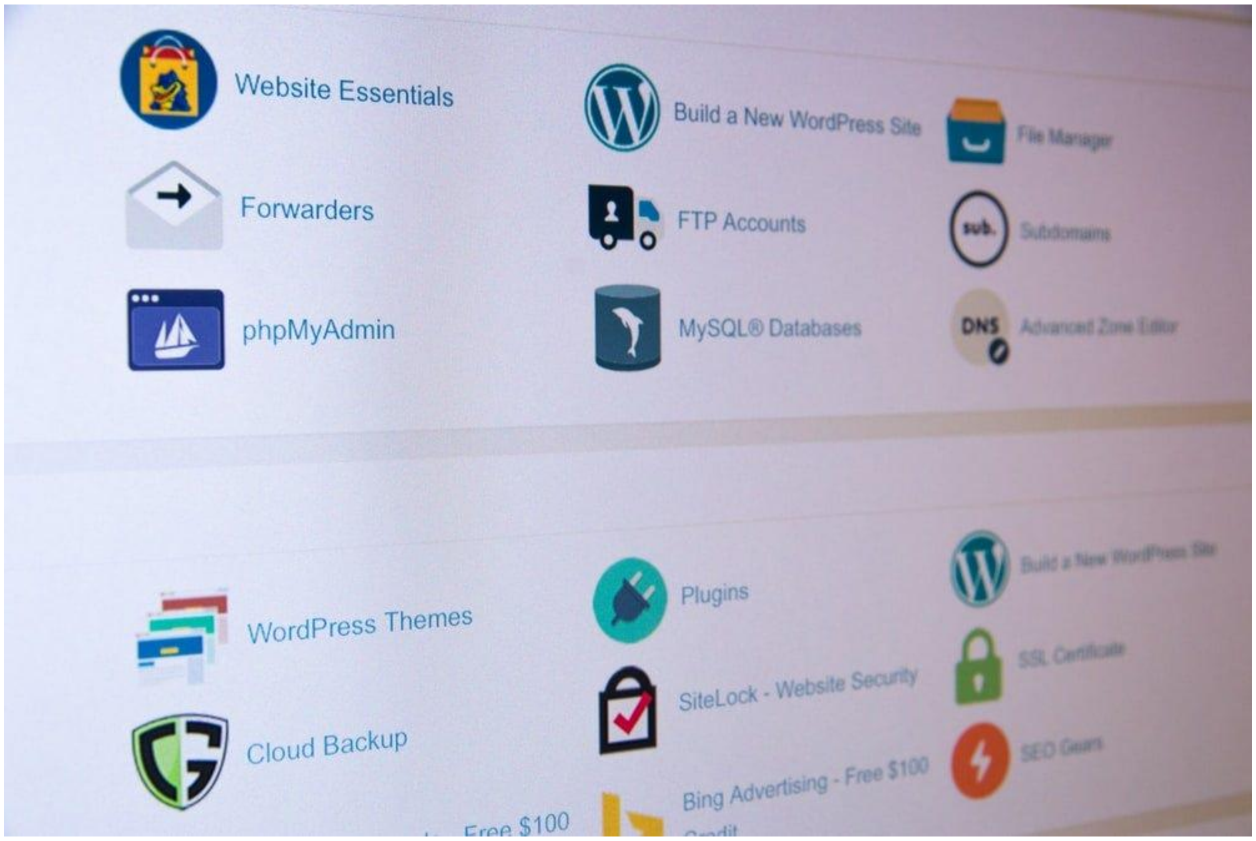


Image from [Unsplash](#) by [Stephen Phillips](#)

In the past I've talked about how to train a custom TensorFlow model on [Amazon SageMaker](#). This is made easy because SageMaker manages [containers](#) for popular frameworks such as TensorFlow, PyTorch, HuggingFace, and more. This allows for developers to use these provided containers and focus on providing a Script for training and/or inference in a method known as [Script Mode](#).

Now let's say the framework you're working with is not supported by SageMaker's existing Deep Learning containers. This is a real possibility as there's many existing ML frameworks that are being launched with every coming week. SageMaker offers a functionality known as [Bring Your Own Container](#) (BYOC) where you have full control as a developer. In this article we'll walk through an example of bringing a [Pre-Trained](#) Spacy NER model to SageMaker and walk through the deployment process for creating a [real-time endpoint](#) for inference. If you would like to skip straight to the code, check out this repository with the source code and other SageMaker examples related to inference.

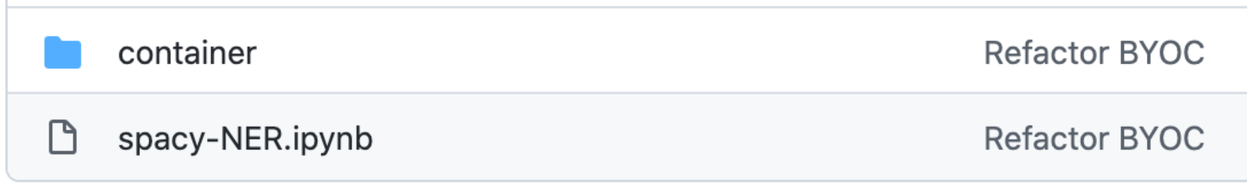
NOTE: For those of you new to AWS, make sure you make an account at the following **link** if you want to follow along. There will be costs incurred through the deployment process, especially if you leave your endpoint up and running. This article will also assume intermediate knowledge of SageMaker and AWS.

Table of Contents

1. Container Setup

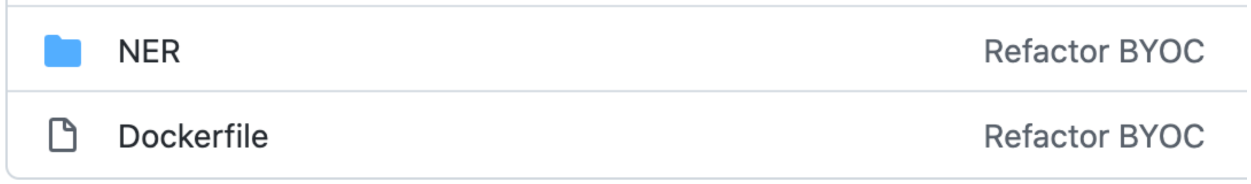
Before we can get started make sure to create a Notebook Instance and get that up and running on SageMaker. Once you have your SageMaker JupyterLab setup open, your directory should look like the following.

Create a Notebook (Python3 Kernel) and then make a directory which will contain the Dockerfile and setup of our inference/serving files.



container	Refactor BYOC
spacy-NER.ipynb	Refactor BYOC

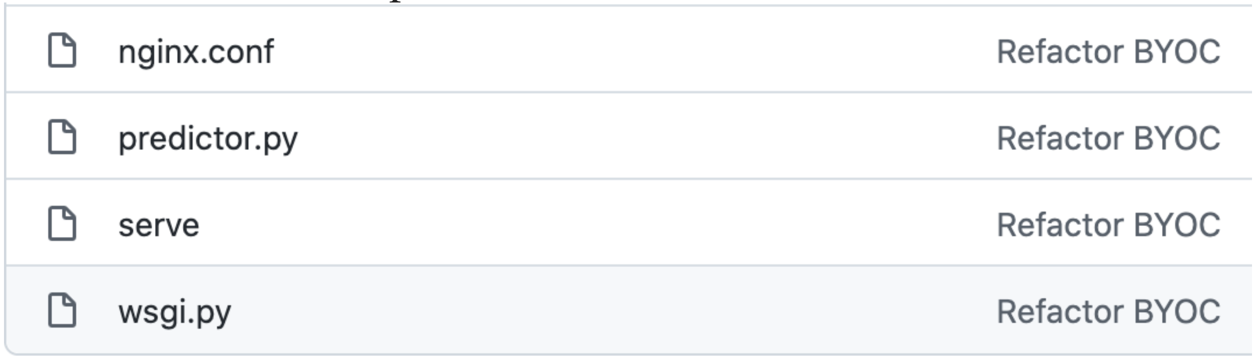
General Structure (Screenshot by Author)



NER	Refactor BYOC
Dockerfile	Refactor BYOC

Dockerfile and Directory For Serving/Prediction Code (Screenshot by Author)

Your Dockerfile will be pointing to your NER directory which contains your code to orchestrate model serving on SageMaker and what Docker will install. Within this NER directory there will be four files that contribute to this setup.



nginx.conf	Refactor BYOC
predictor.py	Refactor BYOC
serve	Refactor BYOC
wsgi.py	Refactor BYOC

NER Directory (Screenshot by Author)

- **nginx.conf**: Configures and sets up the nginx front-end. You **can directly copy this file as it is** for all BYOC examples for the most part.
- **serve**: This program starts a gunicorn server under the hood, it is started when the container is started for hosting, you **can**

also keep this file as is. Note that normally, you'd have a "train" file as well if you were BYOC for both training and hosting. For an example around that check out this code.

- **predictor.py**: This program implements a Flask web server under the hood, there's two routes in **/ping** and **/invocations** that are crucial. Ping receives GET requests and returns a 200 if your container is healthy. Invocations is where you can have your inference logic for the model you are working with and it will return predictions from your model.

- **wsgi.py**: Is a wrapper around your predictor.py file and does not need be changed unless you named predictor.py another file name.

The file we'll go over is predictor.py as the rest of the configuration does not need to be touched or edited.
Load Spacy Model

Here we can load our Spacy model for inference. Next as we discussed there's two URLs that we are working with. We can use the **ping** route to perform a **health check** on our container.
Health Check

Now we can focus on our **inference logic** within the **invocations** route.
Spacy Inference

Here we're expecting a JSON input, so we'll configure that on the client side when feeding the endpoint. Our container is now set up for serving, we now need to create a **Dockerfile** that points to and **executes** these **files** and also **installs** our **model dependencies**.

Dockerfile for Spacy Installation

Here we install Spacy and point our code to the SageMaker /opt/program directory which SageMaker creates under the hood when serving the model.

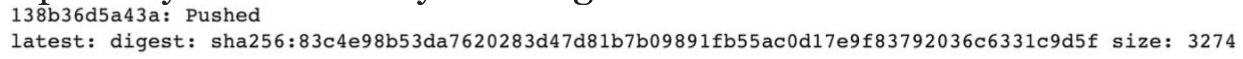
Now we can get in our Notebook and focus on pushing this image to ECR where SageMaker can retrieve and use this image for Inference.

2. Build & Push Docker Image To ECR

SageMaker Classic Notebooks come with **support** for **Docker**, so we can locally test in a terminal on the Instance as well. Before we can get into the steps of creating an endpoint we need to build and push our Docker image to ECR which is where SageMaker stores and retrieves all images for training and inference. The following shell command can be run in the Notebook, without any major changes.

Push Docker Image to ECR

Here we make the serve file executable, name our ECR repository with our algorithm name, and build the Docker image based off of our container structure and Dockerfile. This should take a bit but you should be able to see a successful image at the end and a ECR repository created with your image.



```
138b36d5a43a: Pushed
latest: digest: sha256:83c4e98b53da7620283d47d81b7b09891fb55ac0d17e9f83792036c6331c9d5f size: 3274
```

Image Pushed to ECR (Screenshot by Author)

3. Create Model

Now we can focus on getting to building a real-time endpoint with SageMaker. For real-time endpoint creation, there's traditionally three steps to follow: Model Creation, Endpoint Configuration Creation, and Endpoint Creation. We can orchestrate these steps by using the SageMaker Boto3 Client.

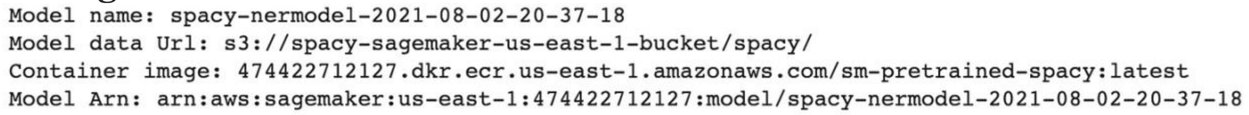
Client Setup

Notice there's two clients: sm_client for model, endpoint configuration, and endpoint creation. The runtime client is for inference and invoking the endpoint.

The first step now is model creation, here we need to **point to the ECR image** we pushed so that SageMaker can create our model. We do this by providing our algorithm name and account_id as a variable for the container location, you can also manually grab the Image URI from ECR though that is not recommended.

Model Creation

Here we also specify the instance type that we are working with so make sure to check out SageMaker Instances to see what is adequate for your use case and pricing. You should see the cell return your created model, which we will use for the next step in Endpoint Configuration Creation.



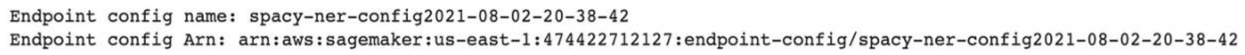
```
Model name: spacy-nermodel-2021-08-02-20-37-18
Model data Url: s3://spacy-sagemaker-us-east-1-bucket/spacy/
Container image: 474422712127.dkr.ecr.us-east-1.amazonaws.com/sm-pretrained-spacy:latest
Model Arn: arn:aws:sagemaker:us-east-1:474422712127:model/spacy-nermodel-2021-08-02-20-37-18
```

Model Created (Screenshot by Author)

4. Create Endpoint Configuration

We can now use our model name and simply feed it as an input for our Endpoint Configuration, which will provide the details necessary for the type of endpoint we are creating.

Endpoint Configuration



```
Endpoint config name: spacy-ner-config2021-08-02-20-38-42
Endpoint config Arn: arn:aws:sagemaker:us-east-1:474422712127:endpoint-config/spacy-ner-config2021-08-02-20-38-42
```

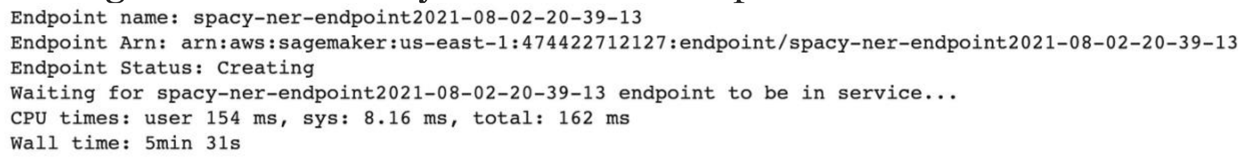
Endpoint Configuration Creation (Screenshot by Author)

5. Create Endpoint

We can now use the Endpoint Configuration to next create the Endpoint we will be using for Inference with another simple Boto3 call.

Endpoint Creation

This should take a few minutes to execute, but if you see a successful message we’re now ready to see some sample inference.



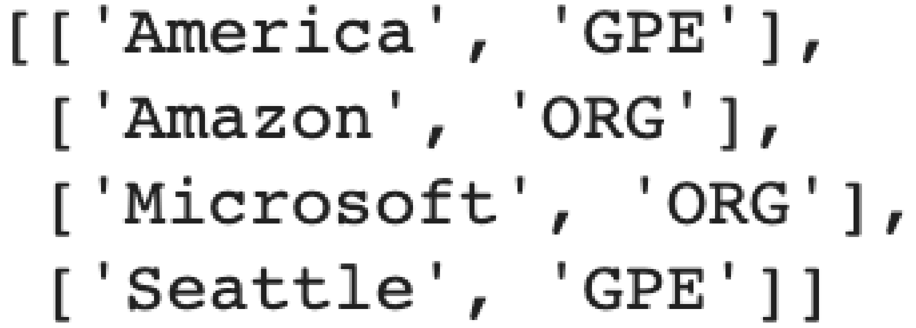
```
Endpoint name: spacy-ner-endpoint2021-08-02-20-39-13
Endpoint Arn: arn:aws:sagemaker:us-east-1:474422712127:endpoint/spacy-ner-endpoint2021-08-02-20-39-13
Endpoint Status: Creating
Waiting for spacy-ner-endpoint2021-08-02-20-39-13 endpoint to be in service...
CPU times: user 154 ms, sys: 8.16 ms, total: 162 ms
Wall time: 5min 31s
```

Endpoint Created (Screenshot by Author)

6. Inference

Now we can invoke the endpoint. We need to remember we fed in a sample input json on our predictor.py file so we configure and serialize our input properly before invocation.
Sample Invocation

Now we see some sample NER results from our endpoint.



```
[['America', 'GPE'],
 ['Amazon', 'ORG'],
 ['Microsoft', 'ORG'],
 ['Seattle', 'GPE']]
```

Screenshot by Author

If you don't want your endpoint up and running make sure to run the following [Boto3 call]() to delete your endpoint to not incur costs.
Delete Endpoint

7. Conclusion & Additional Resources

SageMaker-Deployment/RealTime/BYOC/PreTrained-Examples/SpacyNER at master ·…
Compilation of examples of SageMaker inference options and other features. …
github.com

To access the **entire code** for the example check out the **link above**, there's also plenty of other SageMaker Inference examples and resources. BYOC can be intimidating at first but it gives incredible flexibility to configure and adjust SageMaker to work with your framework. At the end of the day by changing a few files within this current setup you can run Inference at Scale with whatever models and frameworks you’re using.

To understand what SageMaker Inference option to use check out this article. To learn how to work with existing SageMaker Images check out this article.