Machine Learning Company

MODEL AS A REST API ON AWS

So you've spent days, weeks or even months working on your cutting edge *machine learning model; cleaning data, engineering features, tuning model parameters and endless testing. And now that it's finally ready for production, you want to make it available in an easy and reliable fashion. One way to achieve this is to deploy it as a REST API.*

Here at Machine Learning Company, we rely on Amazon Web Services to host our machine learning models. Our automatic product classification model is hosted on their platform for example. The services and features offered by Amazon make it relatively easy to deploy large scale *machine learning models. But it can still seem like a daunting task to someone that does not have experience with either the AWS platform or model deployment in general.*

In this post *I'll show you how to deploy your machine learning model as a REST API using Docker and AWS services like ECR, Sagemaker* and *Lambda. We'll begin by saving the state of a trained machine learning model, creating inference code and a lightweight server that can be run in a Docker Container. We'll then deploy the containerized model to ECR and create a machine learning endpoint in* Sagemaker*. Then we'll finish off by creating the REST API endpoint. The model used in this post was made using Scikit Learn, but the approach detailed here will work with any ML framework in which an estimator's or transformer's state can be serialized, frozen or saved.*

All the code used in this post can be found in the following repository: https://github.com/leongn/model_to_api

Machine Learning Company

THE MODEL

For this post, we will be using a sentiment analysis model which was trained on 100000 labeled tweets (positive or negative sentiment). The model takes a sentence or series of sentences as an input and outputs the predicted sentiment for the corresponding text. It consists of two parts, the first one being Scikit Learn's TFIDF Vectorizer to process the text, the second part is a Logistic Regression classifier (also from Sklearn) in charge of predicting the sentiment of a sentence.

SAVING THE MODEL

The first step in the deployment process will be to prepare and store your model such that it can be easily re-opened elsewhere. This can be achieved through serialization, which freezes the state of your trained classifier and saves it. To do this, we will be using Scikit Learn's *Joblib*, a serialization library specifically optimized for storing large numpy array's, and thus especially suited for Scikit Learn models. If you have more than one Scikit Learn estimator or transformer in your model (for example, a TFIDF preprocessor, like we have), you can save those using *Joblib* as well. The sentiment analysis model includes two components that we will have to save/freeze: the *TFIDF* text-preprocessor

the folder *Model_Artifacts*.

```
from sklearn.externals import joblib
joblib.dump(classifier, 'Model_Artifacts/classifier.pkl')
joblib.dump(tfidf_vectorizer, 'Model_Artifacts/tfidf_vectorizer.pkl')
```

There are no limits to the size and number of model components. If the model has a larger than usual memory footprint or requires more processing power, you can simply pick a more powerful AWS instance.

CREATING THE DOCKERFILE

Once the estimators and transformer are serialized, we can create a Docker image that holds our inference and server environment. Docker makes it possible to package your local environment and use it on any other server/environment/computer without having to worry about technical details. You can learn more about Docker Images and Containers and how to define them over here: https://docs.docker.com/get-started/. The Docker image will contain all the necessary components that will enable your model to perform predictions and communicate with the outside world. We can define a Docker image with a Dockerfile that specifies the contents of the environment: we'll want to install Python 3, Nginx for the webserver and various Python packages like Scikit-Learn, Flask and Pandas.

The dockerfile can be found in the *container* folder: *Dockerfile*

The first part of the Dockerfile will install a set of Linux packages that are required to run a server and execute our Python code, the second part defines a set of Python packages we will need to use (like Pandas and Flask) and part three defines the environment variables. The last part of the file tells Docker which folder (*sentiment_analysis*) contains our inference and server code that will have to be added to the image.

We can now start creating the code that will serve your machine learning model from inside the Docker container. We are going to use the Flask microframework which handles incoming requests from the outside and returns the predictions made by your model. This is located in the folder *container/sentiment_analysis* in the *predictor.py file.*

The first part of the file imports all the necessary dependencies (any dependency used here should also be added in the second part of the Dockerfile). We load the serialized model components with Joblib. Then we create the Flask app that will serve our predictions. The first route (Ping) checks the health of the container by checking if the classifier variable exists. If it doesn't, it returns an error. This 'ping' is used by Sagemaker later on to check if the server is running and healthy.

Then comes the prediction part: the server accepts POST requests with JSON data in the following format:

```
{"input":
[{"text" : "Input text 1”},
{"text" : "Input text 2”},
{"text": "Input text 3”}]}
```

First we transform the JSON data to a Pandas DataFrame, then we transform sentences in the DF with the TFIDF vectorizer and make predictions with our classifier. The classifier outputs 0 (negative sentiment) and 1 (positive sentiment). We'll change the 0 and 1 to Negative and Positive respectively, for readability. Finally, we transform the results back to JSON which can be sent back as an answer to the request.

CONFIGURING THE WEBSERVER

Most people can skip this part as they will not have to modify the number of workers their server uses. But in the case your model is big (say at least 2GB) you might want to modify the model_server_workers parameter in the *serve* file. This parameter determines how many instances of the Gunicorn server will be started in parallel. The default value will use the total number of CPU's on the

Machine Learning Company

inference code, so if you have a large model, the memory on your machine/instance/server might fill up quickly. You might therefore want to set the number of workers manually. For a large model, set it to 1 and perform some testing to see how high you can go, this parameter can be defined through environment variables.

BUILDING THE DOCKER IMAGE

Now that we've finished defining the environment , the inference code and the server we can proceed to building our Docker image for testing.

Move your terminal to the *container* folder where the Dockerfile is located. Then execute the following command to build your image with the name prediction_docker_image *:*

docker build -t prediction_docker_image .

Please note the dot at the end of the command, this tells Docker to look for the Dockerfile in the current directory. If you get a permissions error, add sudo in front of the command. Docker will now start building the image and will add it to the current image repository on your system.

STARTING A DOCKER CONTAINER

Now that the docker image is built, we'll want to run and test it. But first we'll have to move all the serialized model elements and other extra files you might use to the *model* folder. The *test_dir* folder has the same folder setup as Sagemaker will use on the AWS instance.

We'll then start the image and server with the serve_local.sh script which tells the docker image where the test data is located (*test_dir*), on what ports to operate (8080) and what the name of the Docker image containing the inference code is. Move to the *local_test* folder with your terminal and execute the following command to launch the Docker container and the server on board:

./serve_local.sh prediction_docker_image

Machine Learning Company

```
[2018-07-20 13:49:58 +0000] [17] [INFO] Starting gunicorn 19.9.0
[2018-07-20 13:49:59 +0000] [17] [INFO] Listening at: unix:/tmp/gunicorn.sock (1
7)
[2018-07-20 13:49:59 +0000] [17] [INFO] Using worker: gevent
[2018-07-20 13:50:00 +0000] [21] [INFO] Booting worker with pid: 21
```

If you want to check the memory and CPU usage of your Docker container, open a new terminal and execute the command: *docker stats*

PERFORMING TEST INFERENCES

Now that the server is running, we can send it some data to test it. This can be done with the predict.sh script. This scripts sends the server a JSON file.

While still in the *local_test* folder with your terminal type the following command:

./predict.sh input.json

If everything goes smoothly, you should receive a number of predictions back. Congratulations, your Docker image is ready for deployment!

DEPLOYING THE DOCKER IMAGE

The docker image and the machine learning model artifacts are kept separate in Sagemaker. This way you can create a Docker image file with your inference code but plug in different versions of your model. We'll start by pushing the freshly created Docker image to Amazon Elastic Container Registry (ECR) which will store our image. For this to work, you'll have to have the AWS CLI installed and an existing AWS account configured on your system. If you want to check your AWS configuration, execute the following command in your terminal:

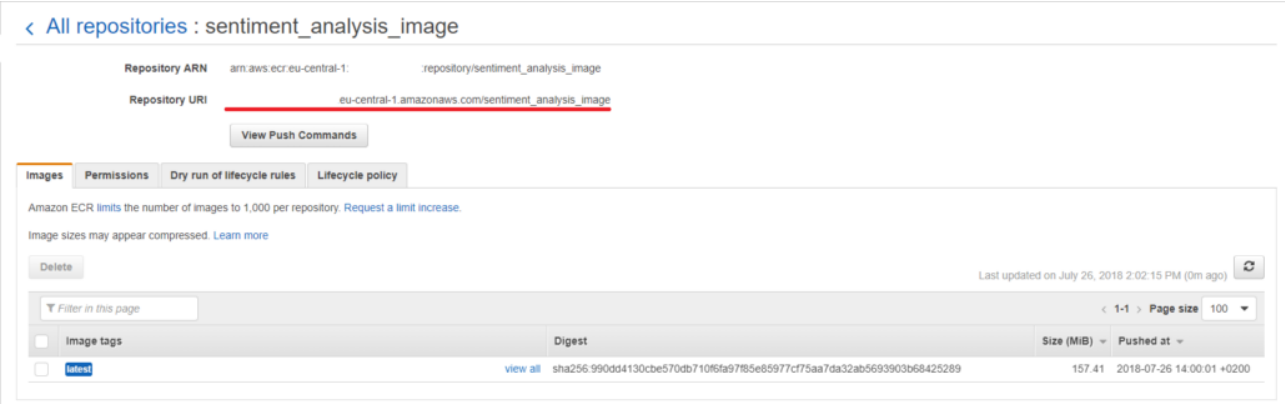
aws configure list

If there's no configuration present yet, you can easily create one with the *aws configure* command and enter your AWS credentials. It's important to note that

located in the same region for them to be able to communicate.
The push process can be performed using the build_and_push.sh script in the *container* folder. This script will automatically create a new ECR repository, if it doesn't exist already, and push your image to it. Move your terminal to the *container* folder and execute:

./build_and_push.sh prediction_docker_image

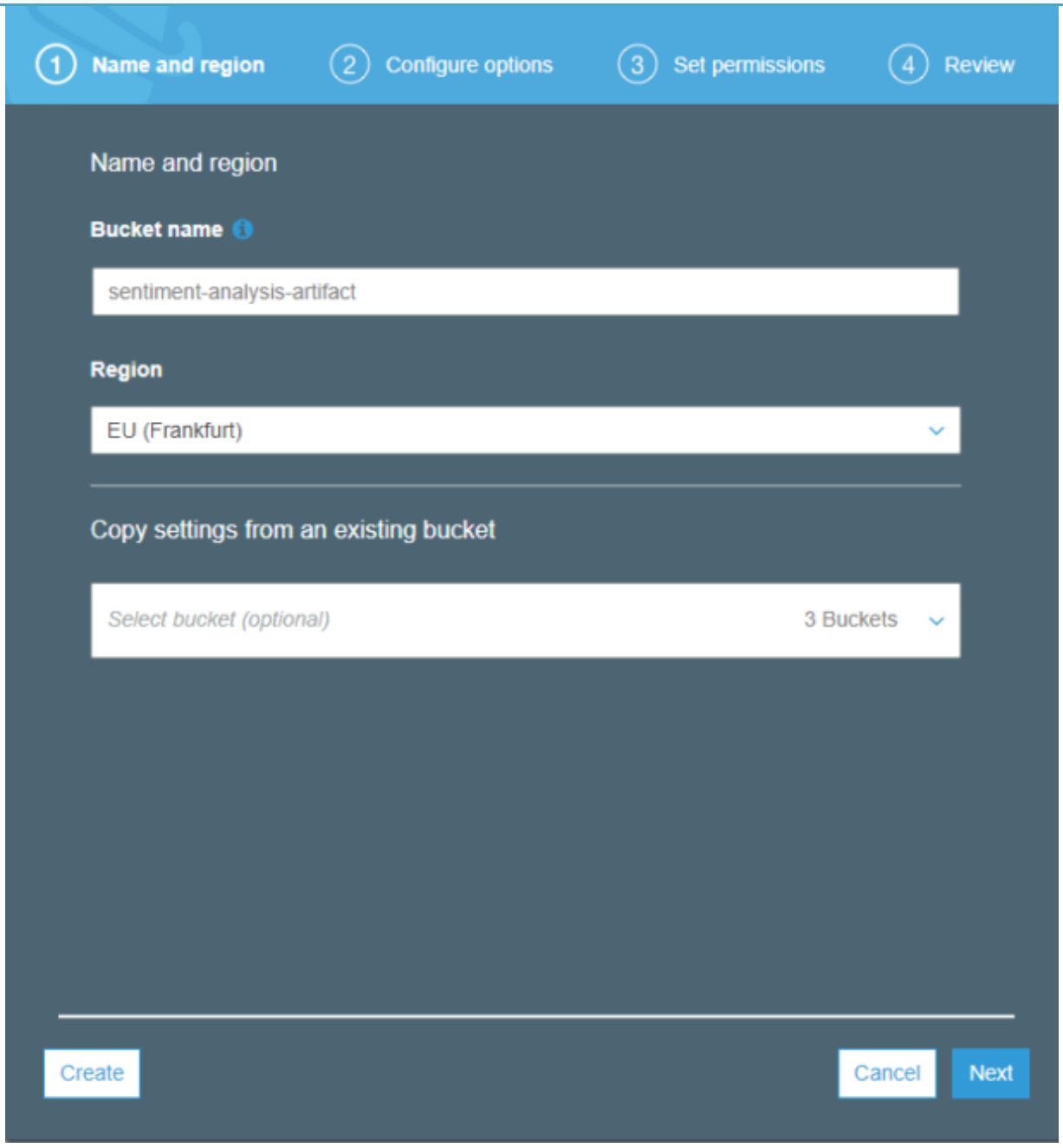
Once the image is uploaded, open the AWS console at https://console.aws.amazon.com and go to ECR and click on *Repositories* in the left pane. Then select the image you just uploaded and copy the Repository URI at the top of the page (it might be useful to create a temporary text file to hold this information as we'll need it later on in the process).



UPLOADING MODEL ARTIFACTS TO S3

As discussed before, the Docker image only contains the inference environment and code, not the trained serialized model. The serialized trained model files, called model artifacts on Sagemaker, will be stored in a separate S3 bucket. We start off by putting all the necessary (serialized) model components in a tar.gz compressed file (found here in the repository). You can then upload them to one of your existing S3 buckets (in the same region as your ECR image), or create a new one. These steps can be performed with the AWS CLI or through the online interface.

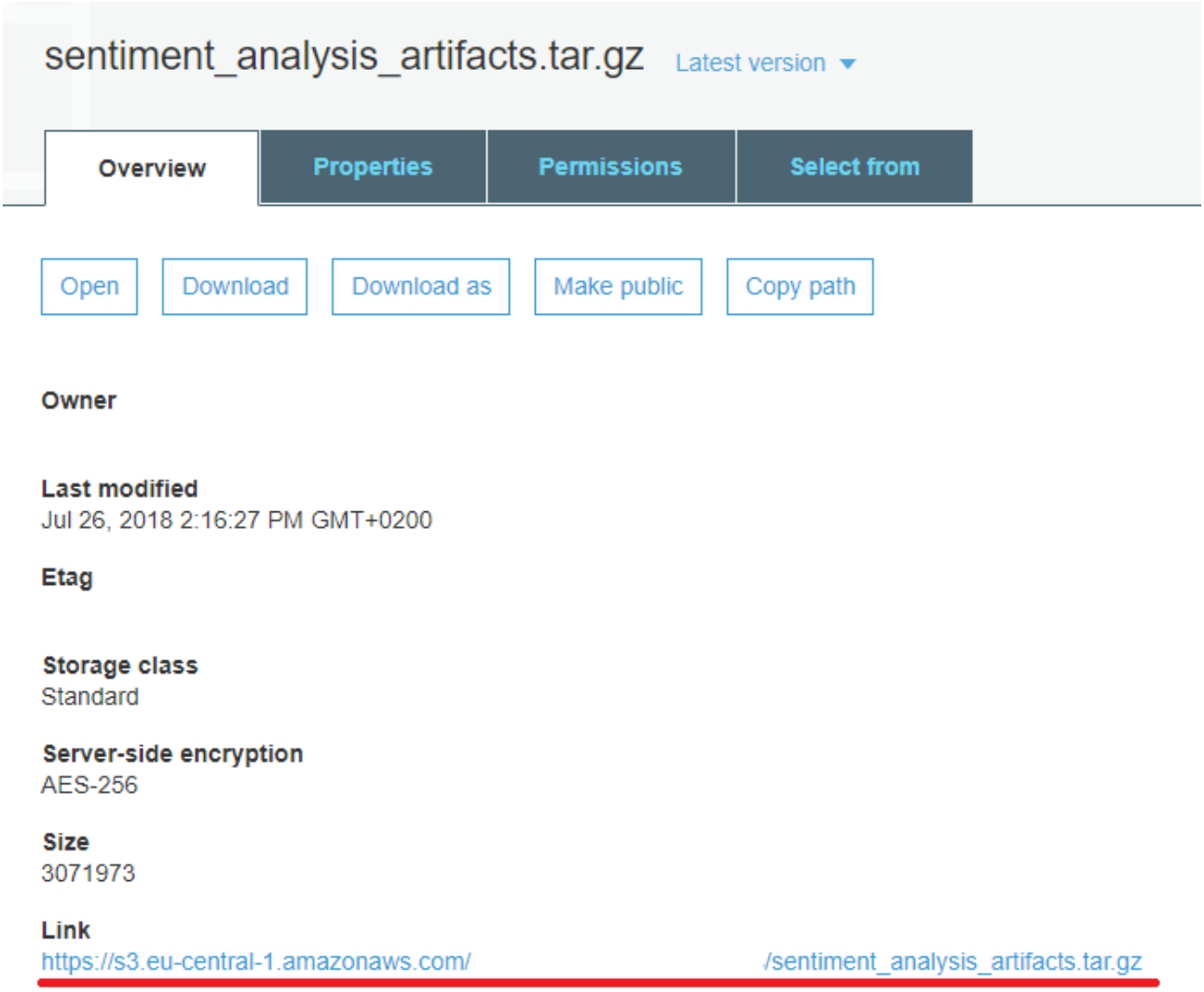
To create a new bucket with the AWS online interface, go to the AWS console and select S3. Next, click on Create Bucket, chose a name for the bucket (for



The region should be the same as the region you used for ECR. On the next *Configure options* page you can change different settings related to your bucket, I would recommend selecting automatic encryption. The third screen is related to setting access permissions. The defaults should do for most users. Finally, the last page allows you to review all your settings.

Machine Learning Company

Upload. Proceed by selecting the compressed file containing your model artifacts. For the next steps, the default settings are suitable. Wait for the upload to complete, click on the newly uploaded file and copy the URL at the bottom of the page for later use.

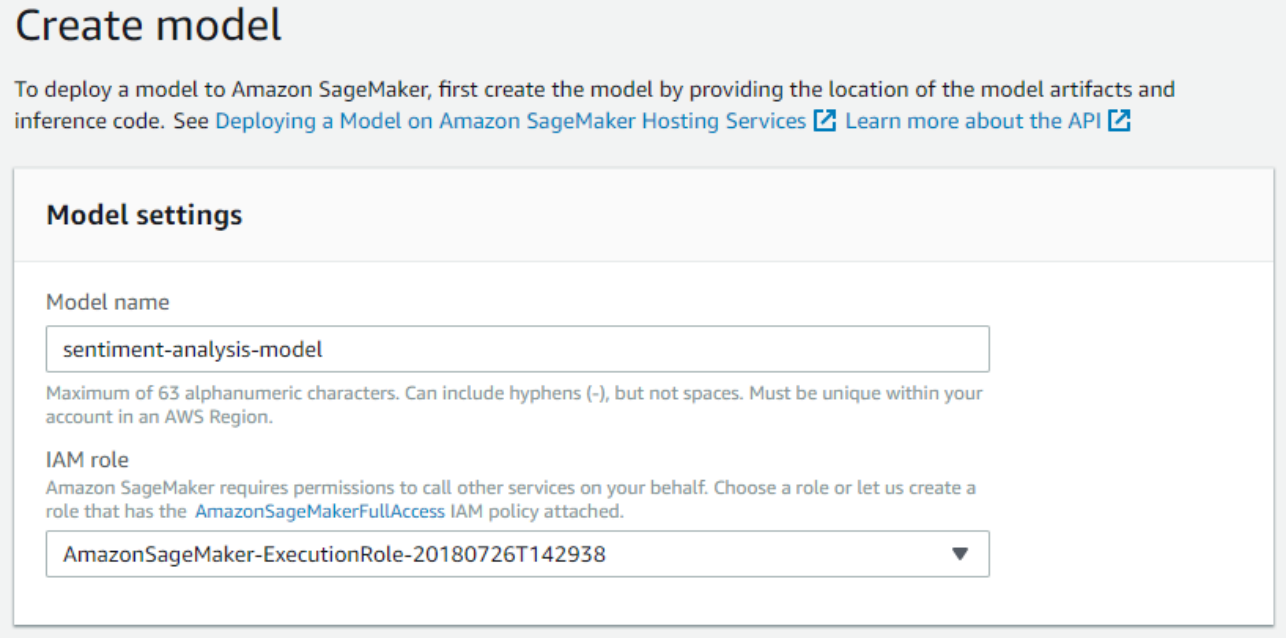


SETTING UP THE IMAGE IN SAGEMAKER

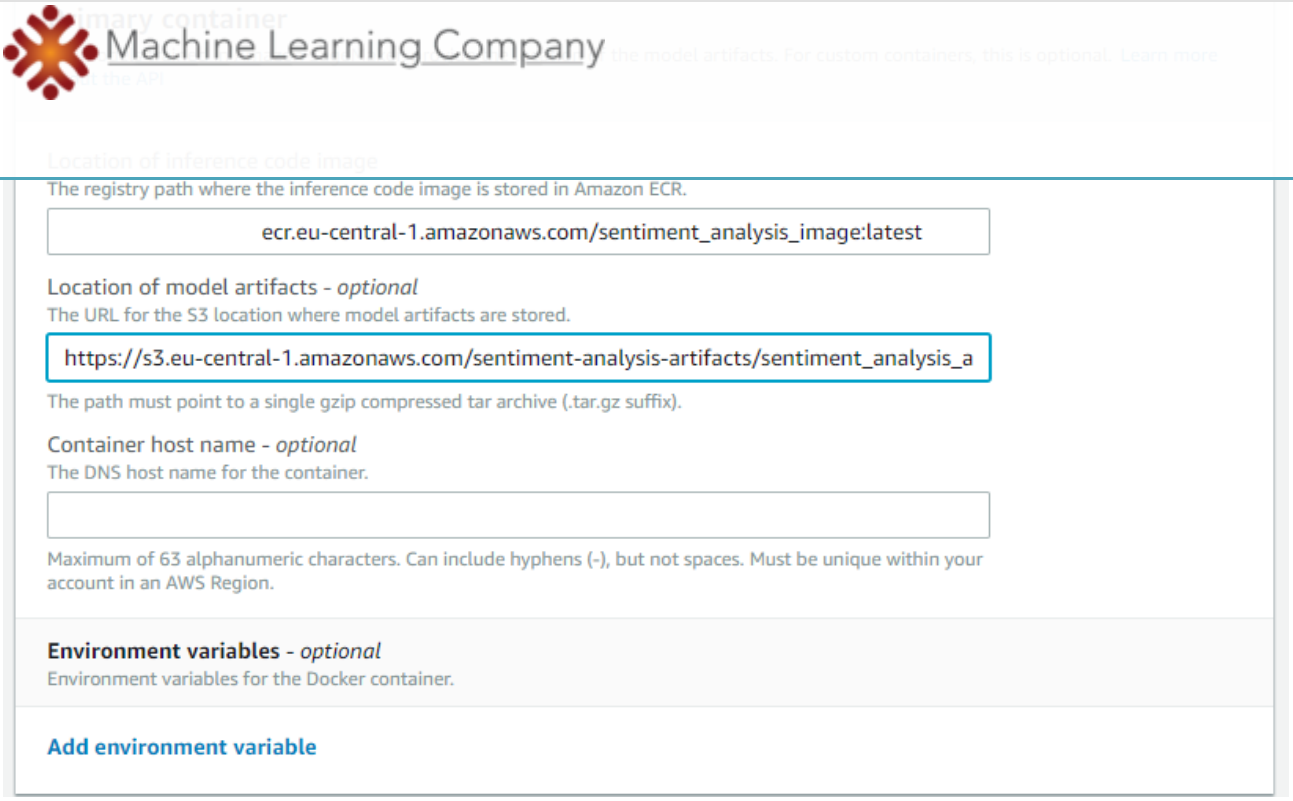
Model Creation

Now that the image is deployed to ECR and the machine learning model artifacts have been uploaded to S3 we can start configuring the Sagemaker prediction endpoint. We will have to start off by creating a Sagemaker model resource. Go to Sagemaker through your AWS console, then in the left panel under Inference, click on Models, then click on Create Model on the right side

how to give our model a name and assign an IAM role to it. If you already have a IAM role for Sagemaker, pick that one. Else, choose Create Role from the drop down menu. We want to grant our Sagemaker model access to our S3 bucket, either give the name of your S3 bucket (sentiment-analysis-artifacts) under Specific S3 buckets, or select Any S3 bucket.

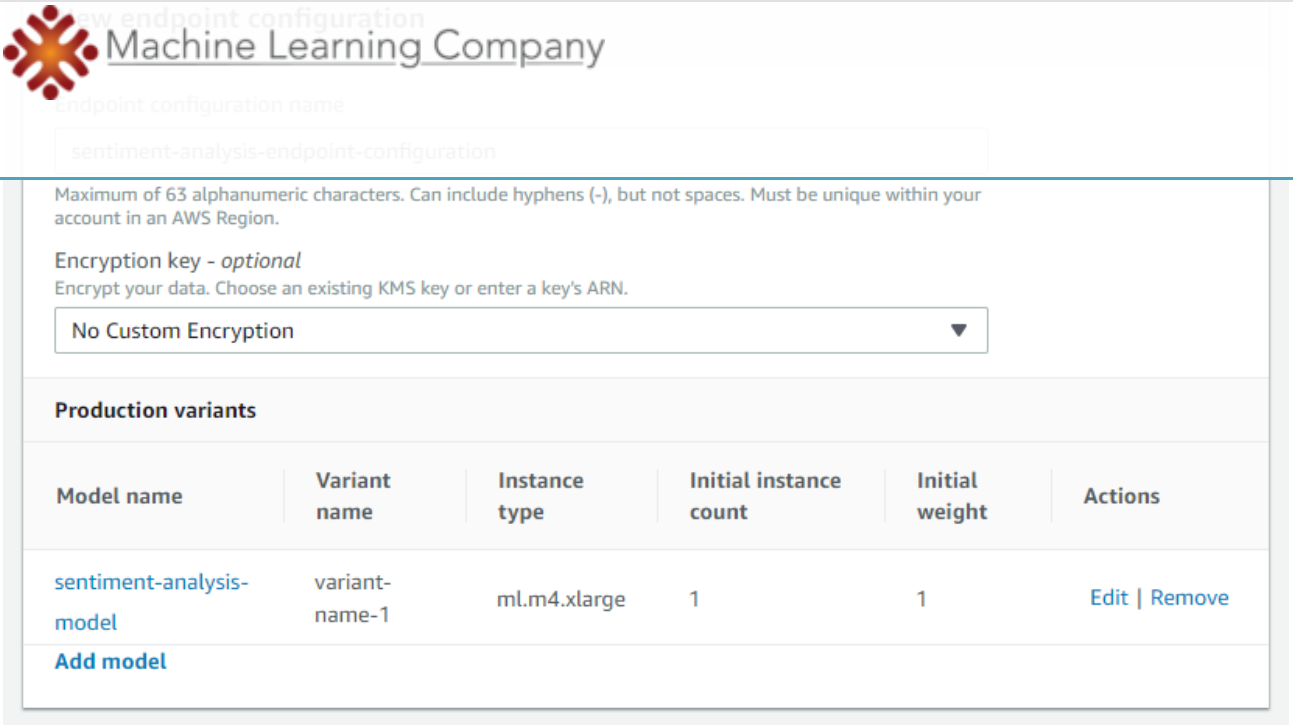


We have to tell Sagemaker where the Docker image and model artifacts are located. You can now paste the ECR URI and S3 URL that we obtained in the previous steps in the corresponding fields. Add the tag *latest* at the end of your image URI (ex: **********.ecr.eu-central-1.amazonaws.com/sentiment_analysis_image:latest) to ensure that Sagemaker always picks the latest version of your model from ECR. The Container host name is optional and adding tags like Version – 1.0 might be useful for versioning. Finish off by clicking create model.



Endpoint configuration

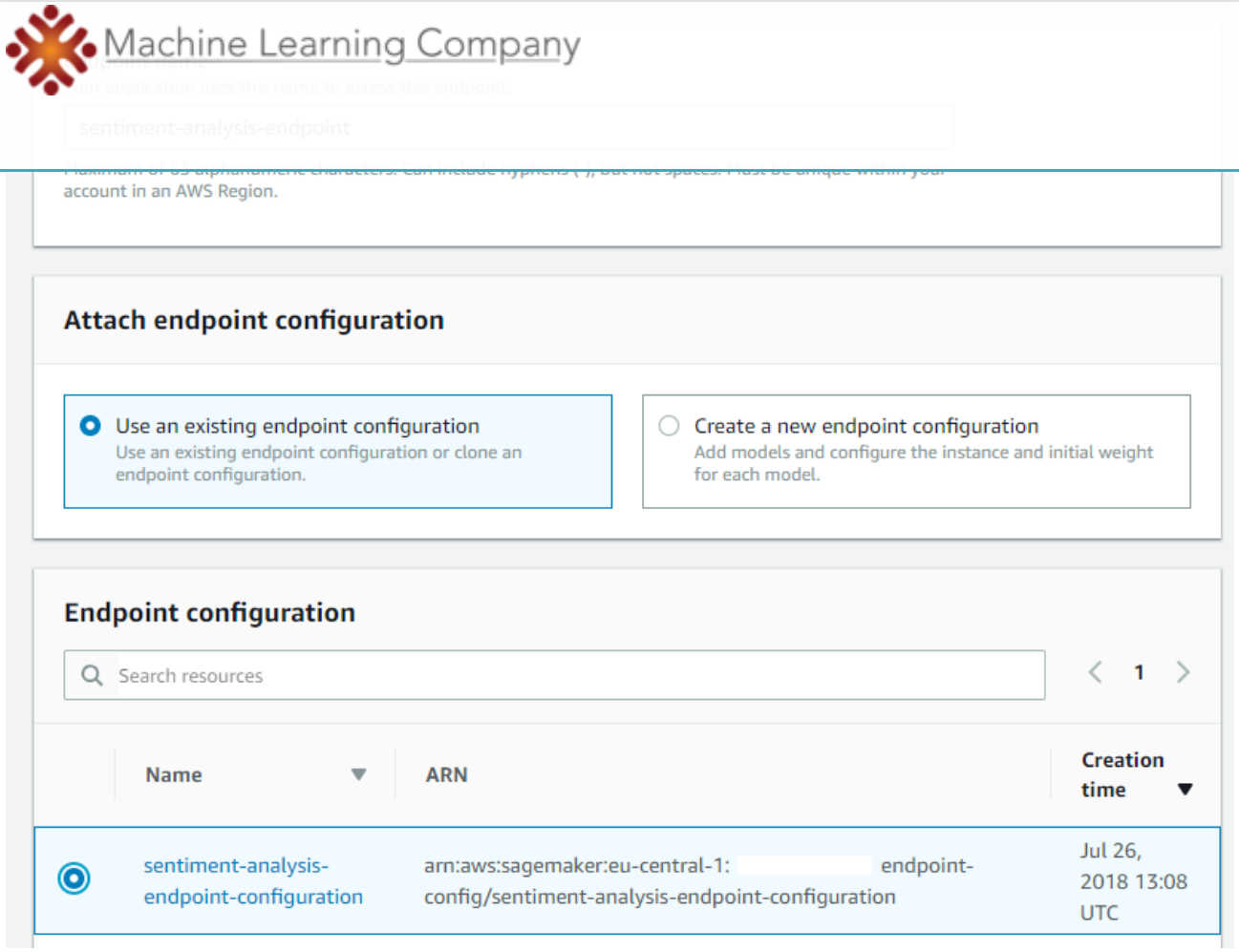
The next step is to create and endpoint configuration in Sagemaker (left pane -> Endpoint Configurations-> Create endpoint configuration). This lets us specify which model will be added to the endpoint and what AWS instance to run it on. Start off by giving it a name (ex: sentiment-analysis-endpoint-configuration). Then click Add model and select the previously created model.



We can then edit the model by clicking on the *Edit* button next to it to choose which AWS instance to run it on. The default ml.m4.large should be sufficient for most loads, but if your model is relatively small or big, compute intensive or lightweight, you can pick another one (see the types of available instances: https://aws.amazon.com/sagemaker/pricing/instance-types/ and the pricing per region: https://aws.amazon.com/sagemaker/pricing/). Finish this step by clicking on *Create endpoint configuration*.

Endpoint creation

The final step consists in creating the Sagemaker endpoint (left-pane-> Endpoints-> Create endpoint). Start off by giving the endpoint a name, this name will be used later on by your API gateway to call the endpoint. Then select the endpoint configuration created in the previous step and click on Select endpoint configuration.



Finish the process by clicking Create endpoint. Sagemaker will then begin to deploy your model, this might take a little while.

If everything goes according to plan, you will see InService appear in the Status column.

CREATING THE REST API

Once the Sagemaker endpoint is created, you can access the model from within your AWS account by using the AWS CLI or, for example, the AWS Python SDK (Boto3). This is fine if you want to perform some internal testing, but we want to make it available to the outside world, so we'll have to create and API.

This can be easily achieved using Amazon's Chalice library (https://github.com/aws/chalice). It's a microframework that allows quick

a REST API

First, you need to install Chalice and the AWS SDK for Python with the following command

sudo pip install chalice boto3

The API gateway needs to be deployed in the same region as your model endpoint. To make sure that it is correct, you can check your region configuration again by entering the following command in your terminal: *aws configure list*

If the region is incorrect, configure it correctly with: *aws configure*

Code for this part can be found in the *api_creation* folder.

The app.py file in the main folder contains the routing logic of the API: what actions will be performed when receiving certain requests (like POST). Here you can eventually change the format of the incoming data or do some additional checks if these are not performed in the inference code present in your Docker Container. The main function of this code is to invoke your Sagemaker endpoint when a POST request with data is sent to your API, and return the response.

When everything is configured, you can deploy the API gateway by moving into the api_creation folder with your terminal and executing the following command: *chalice deploy*

Chalice will then return the REST API URL. Copy it and save it.

```
Creating deployment package.
Creating IAM role: sentiment_analysis_api-dev-api_handler
Creating lambda function: sentiment_analysis_api-dev
Updating rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:eu-central-1:           :function:sentiment_anal
ysis_api-dev
  - Rest API URL: https://          execute-api.eu-central-1.amazonaws.com/api/
```

You can now use that endpoint URL to perform requests. You can use the following Python code to test it:

```
import requests
input_sentiment = {'input':
                   [{'text' : 'Today was a great day!'},
                    {'text' : 'Happy with the end result!'},
                    {'text': 'Terrible service and crowded. Would not
                   ]}
input_json = json.dumps(input_sentiment)
# Define your api URL here
api_url = 'https://*******.execute-api.eu-central-1.amazonaws.com/api/'
res = requests.post(api_url, json=input_json)
output_api = res.text
print(output_api)
```

The code above yielded the following output:

{“output”:
[{“label”: “Positive”},
{“label”: “Positive”},
{“label”: “Negative”}]}

Looks consistent with the input, your machine learning API is now fully operational!

SHUTDOWN ENDPOINT

Once you are done and you don’t need the API anymore, do not forget to delete the endpoint in Sagemaker because the instance costs will keep accruing.

CONCLUSION

In this post you have seen how to:

- Serialize your machine learning model components for deployment
- Create your own Docker image with inference and server code

Machine Learning Company