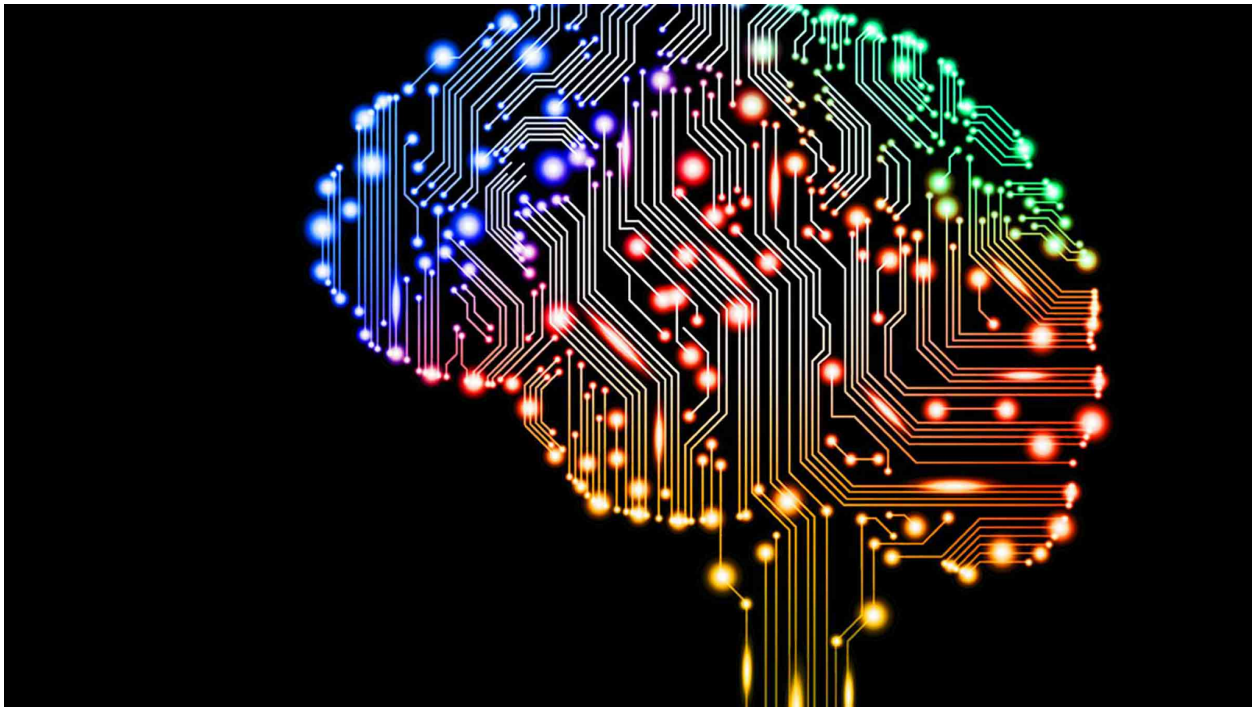
MLOps- Migrate On-Premise Machine Learning Models to AWS Sagemaker (Bring your own Model/Algorithm)

Dockerize On-premise custom model when Sagemaker's inbuilt algorithms cannot provide the desired models that is satisfactory to the client

Introduction



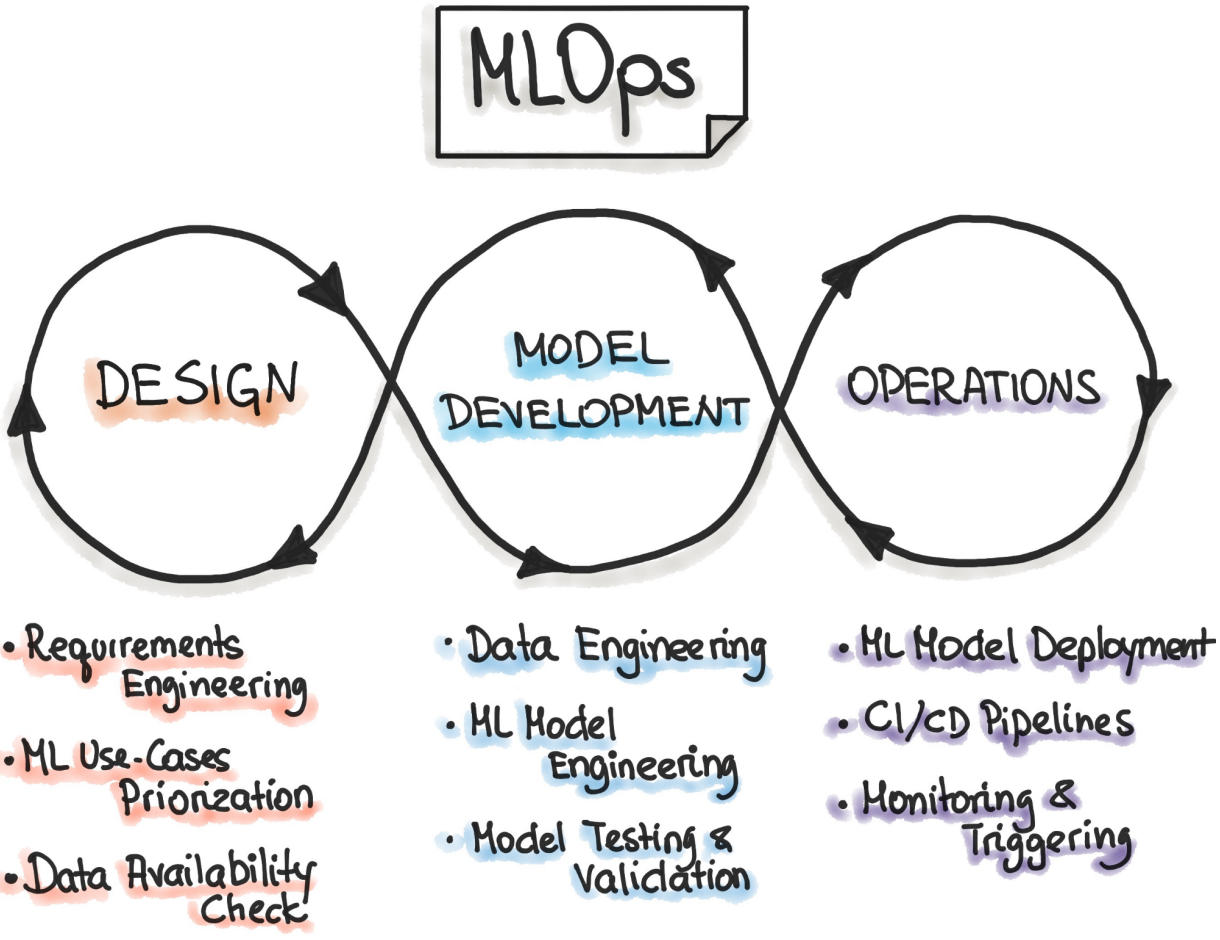
As importance of data science is at boom where the prime reason being the abundance of data generations and to understand the trends in this data can impact the business tremendously by providing various insights by generating models which provide support to data analysis

That being said , due to the 3Vs of big data it is instinctive that custom algorithms and models would perform better in providing the insights for a defined business field and there are some services that provide support to on-premise algorithm.models to be used by them

Amazon offers us a machine learning service known as AWS Sagemaker which can be used by data scientists and enthusiasts to create , train and deploy models on cloud(production ready) with ease.

Docker containers are the backbone of AWS Sagemaker tasks at time of building and runtime, all the built-in algorithms and frameworks supported by Sagemaker are available as readymade Docker images that can be used on the go for training and deploying the machine learning algorithm as models

Ways to achieve MLOps



Docker containers are the prime utility for AWS Sagemaker to perform all the tasks i.e running the scripts, training the algorithm and deploying trained models

Below are the options to train and deploy the model :-

1. Sagemaker can make use of the in-built algorithms/frameworks in most of the use cases where there is no need to think about the containers and we can train and deploy the selected algorithms for creating your Estimator using the Sagemaker console, the command line interface provide by amazon (AWS CLI) , any python notebook or even using the Amazon Sagemaker Python SDK

2. If you want to use the Docker containers containing the in-built algorithms/frameworks, Sagemakers also has the options to directly use the container for algorithms and Docker images loaded with most common and known ML frameworks.

3. Sagemaker even has a option to tweak the existing available Sagemaker container images if you are not satisfied with the current Estimator by extending the current Sagemaker algorithm and Docker image of the model

4. And finally if you want to utilize your existing custom container image Sagemaker provides adaptability for compatibility for Sagemaker Training pr Inference toolkit with your pre-existing custom container image by modifying the custom container

AWS Sagemaker - Under the hood

The Docker containers are ran by AWS Sagemaker by passing either ‘train’ or ‘serve’ arguments , and the effects of both these arguments are dependent on the container you are running in the following ways -

- For a not defined ENTRYPOINT in the Dockerfile, Docker will run train/serve at the time of training/serving time respectively
- With a program defined as an ENTRYPOINT it will execute the program at startup and will take train/serve as its 1st argument, and then the program will process the argument and proceed as defined for each case

- In the case of creating different containers to train and host or just creating one for an individual functionality we have the option to just ignore the 1st argument passed in when defining a program as an Entrypoint

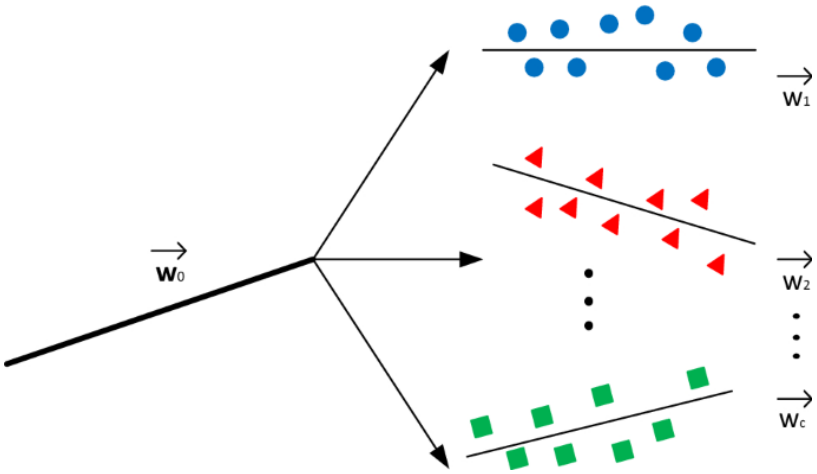
Business Abstract

Problem 1 : Manual Model Handoff

Currently in the Insurance Industry there is a manual handoff for the on premise model where the data-scientist have to provide the model to the developer which might include procedures such as code commit/push of a R file and creation of mapping documents and then the developer need to update and redeploy the model to platform , the complete manual procedure is repeated even for a slight update in the model which requires time & effort.

Problem 2 : Derivatives of Predictive Output

The predicted value provided as an output by the model is not sufficient for explaining the influence of input predictors , so we need to provide a large number of outputs that need to be derived using other functions that derive the importance of predictors for Input policy data.



For eg - The industry uses a Risk term which can be calculated -

Risk term = input predictor variable * coefficient for predictor variable

Where the above coefficient is a static value that is assigned at the time of model creation and is required

Problem 3 : Non Availability of Industry specific Algorithms/Models on Sagemaker

General linear model of R with tweedie distribution is not present as an built-in container offered by AWS Sagemaker

Problem 4 : Single output value

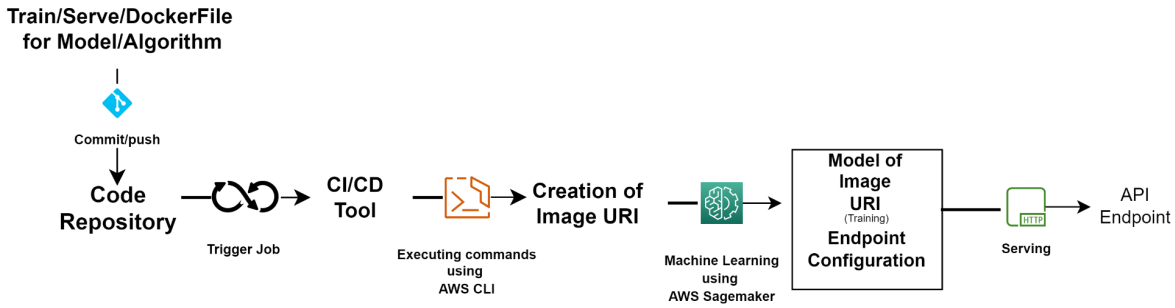
The in-built Linear learner container offered by AWS Sagemaker can only give prediction value as the output which is a problem as we require large numbers of values as a solution to problem statement 2 discussed above

Solutioning and Implementation

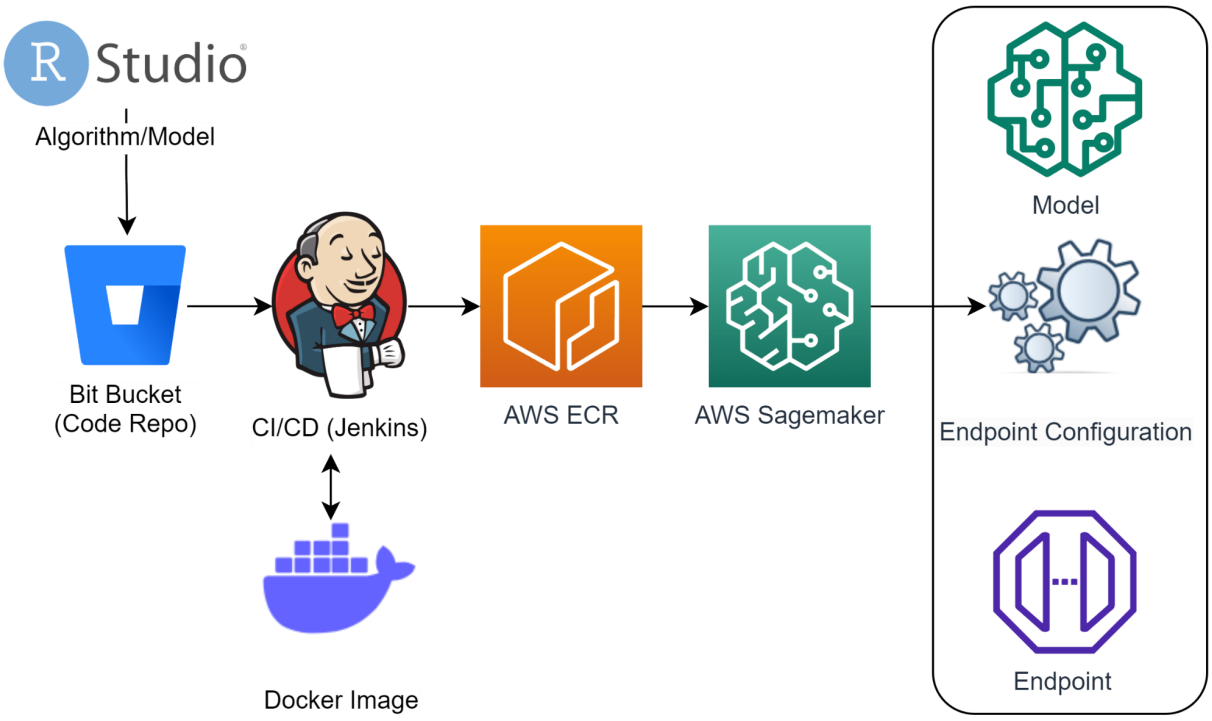
Solutioning

With all are requirements ready, we are all set to flow thru our architectural design

Architecture



- Any code changed
- Commit/Push to code repository
- CI/CD tool will trigger a job on any change in the repository
- The job will use the AWS CLI/SDK to connect with image registry
- Creation of the container image URI
- The image URI is then passed to AWS Sagemaker which provides us with the Model/Endpoint Configuration and a callable api endpoint which will act as a server to our trained model



- The custom Algorithm/Model is created using RStudio on our machine which has a satisfactory result we name it (Train.r) which is the ENTRYPOINT in our case and has both train and serve function declared in it , the serve function uses

Plumber library as it provides simple and quick api functionality and we have a Dockerfile for configuring our container

- BitBucket repository is used for the SCM(source code management)
- Jenkins will serve to detect any changes in our repository to trigger our jobs
- Docker image URI will be created by the tag-team of AWS CLI and AWS ECR called thru jenkins job
- Finally we have our model's api endpoint created for us when we pass the URI to sagemaker service defining the api configurations

Case Study

To manifest our architectural design into an industry deliverable, we need to create the following files :

1. Train: Train.r creates functions to train and serve our model.
2. Serve: apiEnpoint.r uses the plumber package to create a lightweight HTTP server .
3. Dockerfile: This specifies the configuration for our docker container

1. Train.r

Train.r contains the main scripts for creation, training and serving the model of specific algorithms like glm, mars etc.

Entry Point option specifies initial R script(i.e train.R) to Run which will further invoke APIEndpoint.r script.

2. APIEndpoint.r

APIEndpoint.r script will listen Http POST request on Port 8080 using the Plumber library and upon receiving request will load on-prem R model specified in the directory and return the model prediction as Http response, within this we will call explains.R to calculate the derivative variables required for industrial operations and define a customized serve function for generating and providing the expected output values,

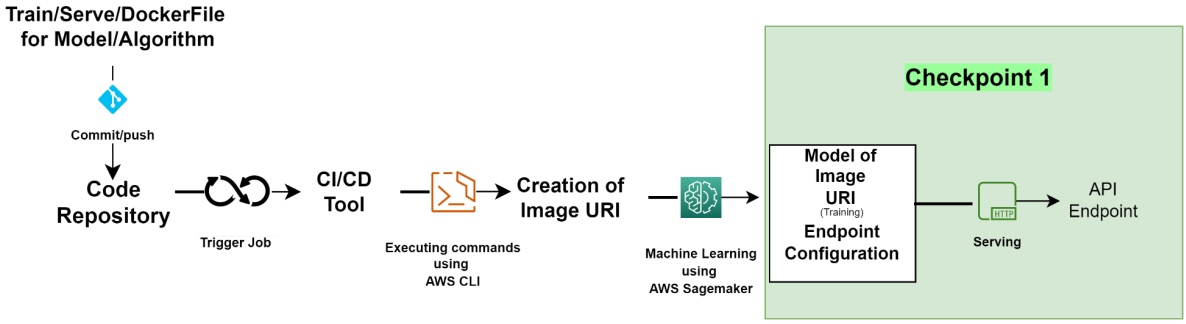
3. **Dockerfile**

Smaller containers are preferred for Amazon SageMaker as they lead to faster spin up times in training and endpoint creation, so this container is kept minimal. This docker file starts with base R image, installs required libraries and their dependencies, then adds Train.r and APIEndpoint.r , and finally sets Train.r to run as the entrypoint when launched.

Implementation

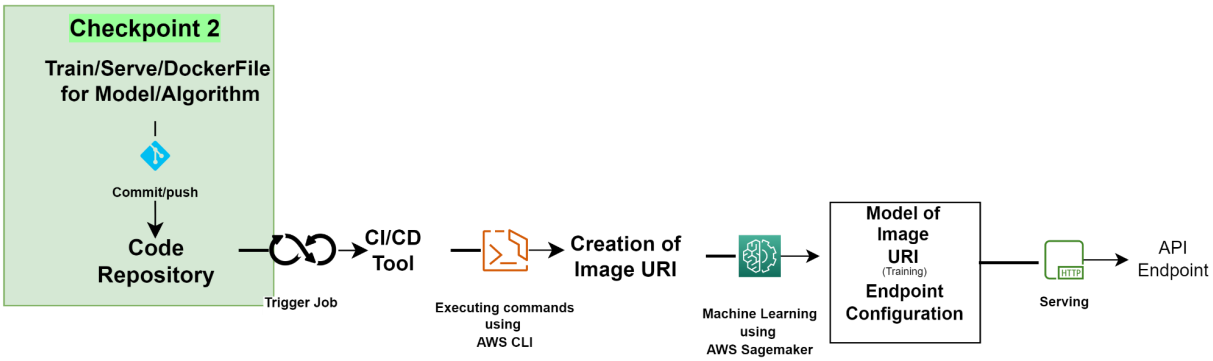
Checkpoint 1 : Dynamic handoff of model

The solution to our problem resides with the api endpoint for the model provided by AWS Sagemaker where the data scientist can directly provide the endpoint for invoking the model and no actual handoff is required



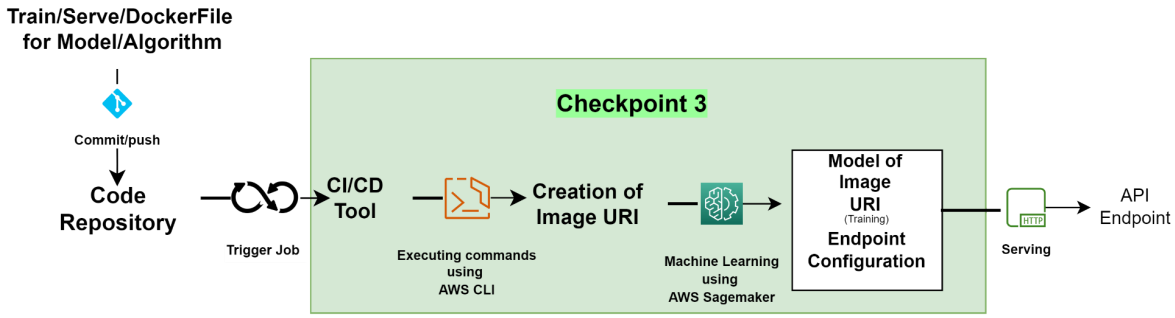
Checkpoint 2 : Providing derivatives with the Model

The custom explains.R file created and pushed to repository will contain the functions required to perform the generation of the derivative variables



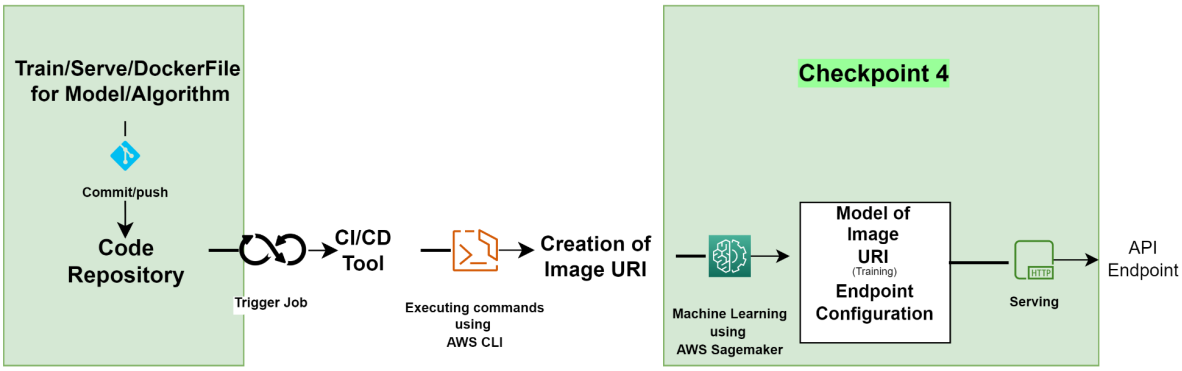
Checkpoint 3 : How to Dockerize the On-Premise Model

AWS Sagemaker comes with abundant in-built ready to use algorithms but is not limited to them and we can use our own custom models/algorithms using the bring-your-own-algorithm feature provided with this service using your own dockerized containers.



Checkpoint 4 : Customized serve function

Using the APIEndpoint.r file we can create a custom serve function which can be set to return any number of values required by the specific industry like explain variables and risk terms



About the Author

Subodh Pandey - Subodh Pandey is an Engineering Graduate in Information Technology with more than 10 years of industry experience. He has done Executive Program in Data and Decision Sciences from the Indian Institute of Technology Delhi, he is also author of white paper. Currently, he is working as a data scientist with one of GlobalLogic's Insurance domain Clients. Also, he is a core member of AI/ML practice in GlobalLogic, India