**EvaDB in Amazon SageMaker**

GitHub: <https://github.gatech.edu/shrn-sthsh/EvaDB-SageMaker-Container>

DockerHub: <https://hub.docker.com/r/ssathish6/evadb-sagemaker-container> (CPU-only)

<https://hub.docker.com/r/ssathish6/evadb-sagemaker-container-cuda> (CUDA GPU)

**Introduction**

Amazon SageMaker is a cloud machine learning platform which provides developers with tools to create, train, and deploy ML models. EvaDB aligns with SageMaker’s goals but with a very different execution, abstracting many of the details of creating ML models to make model creating simpler. With this point, EvaDB could be used by SageMaker users in assisting the execution of steps in the ML model pipeline.

**Project Overview**

To be able to use EvaDB as a tool with SageMaker, first, EvaDB must be exist within a containerized environment and inserted into a SageMaker instance hosted on Amazon Elastic Container Service (ECS). After this, to effectively use EvaDB in SageMaker’s tools, a train and serve execution mode will have to be implemented. As there are two assignments for the project, and there is a logical split for the implementation, this second part of the assignment completes implementation of the train and serve execution modes along with an example.

**Implementation Details**

Consider the project directory listed below.

.

├── Dockerfile - Dockerfile for usage with a CPU and integrated graphics

├── Dockerfile.CUDA – Dockerfile for usage with a CUDA GPU

├── build\_and\_push.sh – Shell script to push to ECS and build

├── test/

│ ├── serve\_local.sh – Shell script for testing serve execution

│ └── train\_local.sh – Shell script for testing train execution

└── evadb\_instance/

├── algorithm.py – EvaDB application for usage (to be edited by user)

├── nginx.py – Application configuration

├── serve – Serve execution mode

├── train – Train execution mode

└── wsgi.py – Wrapper for Gunicorn

The directory tree above provides a high-level overview of the project. Dockerfiles exists to be used to create and image on ECS to later be incorporated in execution modes of SageMaker. The normal Dockerfile is to only be used when a CUDA-supported GPU isn’t available, but the main intension is to use the CUDA based Dockerfile.

The EvaDB instance itself is built as a Flask application using Gunicorn and NGINX for server functionality. Users will be able to edit how the application works thorough the algorithm.py file regarding training and serving. With project 2, serving and training are supported. Direct instructions to use the container are hosted on the GitHub link to the repository.

**Metrics**

From basic operation, consider some of the following interesting points:

* The Docker image to be hosted on ECS is currently about 2.23 GB before implementation of training and serving handlers
* On the other hand, the Docker image to be primarily used for local testing and development is currently about 314 MB in the same state with the handlers
* Hosting options vary in pricing largely
  + For the more common data science workflow, training will about $0.96 per hour with an ml.m4.4xlarge for 30 minutes per training run with Amazon SageMaker Debugger enabled using 2 built-in rules and 1 custom rule
  + For larger, more customized workloads, the price still stays many times under a dollar per hour, but as the time requirements increase so does the total cost
  + The increase in cost the use of EvaDB within the container was minimal; this could be because the models trained and tested were simple

**Challenges & Lessons**

Like last time, the MindsDB SageMaker container repository used to create this EvaDB SageMaker container is executed quite well with some good documentation, but the difference in usage for EvaDB compared to MindsDB is notable – it’s not a direct translation.

I have not really done much with Python outside of ML and DL courses and algorithms problems, so building a Flask application with different server libraries was new and challenging. Also, from my knowledge, EvaDB does not support the creation of predictor objects like how MindsDB does with syntax as follows.

|  |  |
| --- | --- |
| Create and train predictor object on data | Call on predictor object to predict |
| predictor = db.Predictor(data, ...) | predictor.predict(column, ...) |

So, the only way to use prediction was using the PREDICT SQL keyword along with HORIZON and FREQUENCY. This requires a user to change the algorithm they want for training more directly than in the MindsDB version of the project; by that, rather than having a predictor object that’s abstracted and the program is fed data and queries externally, within the algorithm.py file, user will have to make calls similar to how they would in a Jupyter notebook.

**Outputs**

The container uses EvaDB with all the libraries required. The environment is fitted that any EvaDB application should work as expected. As part of the implementation of serve and train, I filled in algorithm.py with a test program 16, home sale forecasting. Here is the training command:

|  |
| --- |
| cursor.query("""  CREATE OR REPLACE FUNCTION HomeSaleForecast FROM (  SELECT propertytype, datesold, price  FROM postgres\_data.home\_sales  WHERE bedrooms = 3 AND postcode = 2607  )  TYPE Forecasting  PREDICT 'price'  HORIZON 3  TIME 'datesold'  ID 'propertytype'  FREQUENCY 'W'  """) |

And the query:

print(cursor.query("SELECT HomeSaleForecast()"))

With the following results

|  |  |  |  |
| --- | --- | --- | --- |
| index | homesaleforecast.propertytype | homesaleforecast.datesold | homesaleforecast.price |
| 0 | house | 2019-07-21 00:00:00 | 766572.9375 |
| 1 | house | 2019-07-28 00:00:00 | 766572.9375 |
| 2 | house | 2019-08-04 00:00:00 | 766572.9375 |
| 3 | unit | 2018-12-23 00:00:00 | 417229.78125 |
| 4 | unit | 2018-12-30 00:00:00 | 409601.65625 |
| 5 | unit | 2019-01-06 00:00:00 | 402112.96875 |

**References**

The only sources I used for this project was the MindsDB implementation provided, Docker docs, and Amazon SageMaker docs.

* <https://github.com/mindsdb/mindsdb-sagemaker-container>
* <https://docs.docker.com/>
* <https://docs.aws.amazon.com/sagemaker/>
* <https://sagemaker.readthedocs.io/en/stable/>