Crypto Forecasting App - Project Outline

Connor Capitolo
David Assaraf
Tale Lokvenec
Jiahui Tang

Harvard John A. Paulson School of Engineering and Applied Sciences
IACS Institute for Applied Computational Science

Outline

- Project Scope
- Project Workflow
- Process/Data Flow
- Backend Infrastructure
- App Design
- Data
- Models

Problem Definition

The current state of the crypto market is extremely volatile. Due to lack of experience/involvement from traditional actors, there is a lack of systematic investment strategies in the crypto environment; therefore, there is an opportunity to extract value from an accurate prediction of the price dynamics of pairs.

The scope of this project is to create a Proof of Concept to see if there is opportunity when using Deep Learning in crypto markets.

Objectives

1. Bridging the lack of structure dealing with crypto exchanges in building a scalable and modular database architecture that will gather various features from different exchanges (starting with Binance) for ‘pairs’ (a ‘pair’ refers for instance to the dynamics of the market for BTC vs USDT)

2. Building a predictive ML/DL model using real-time predictions that will enable us to gain insights as to how the market is evolving over time in order to inform trading decision making

Final Product

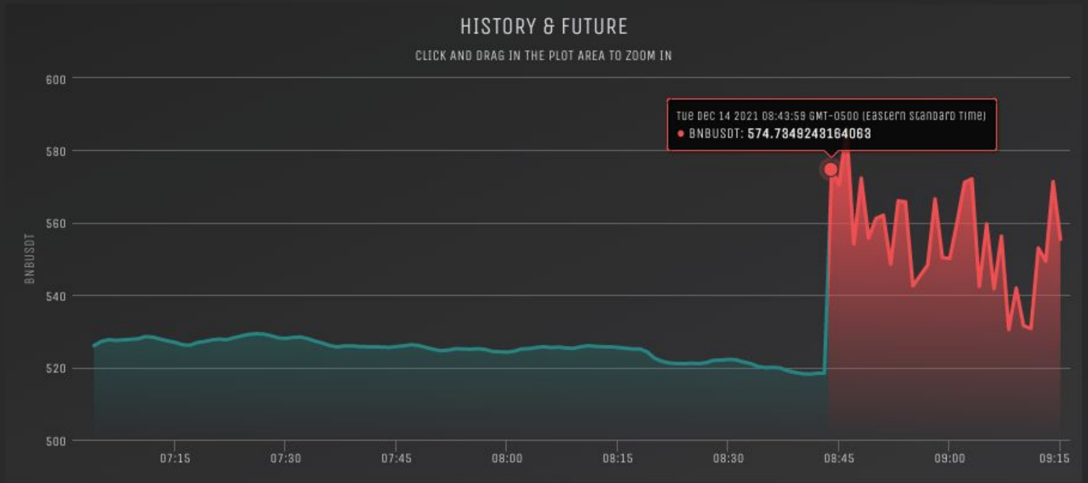
Not Secure | 34.70.108.117

Crypto Forecasting

HOME SYMBOLS

- BTCUSDT
- ETHBTC
- BNBBTC
- BNBUSDT
- ETHUSDT

Visualization Plot



Live Market Data

Symbol	Best Bid	Best Bid(Volume)	Best Ask	Best Ask(Volume)
ETHBTC	0.080364	0.0323	0.080371	1.4806
ETHUSDT	3,791.23	2.4	3,791.72	0.2083
BTCUSDT	47,171.01	0.30931	47,171.02	0.34701
BNBBTC	0.011203	2.541	0.011204	3.357

Prediction Results

Time (Local Timezone)	Predicted Price
8:43:59 AM (+0 min)	574.734924
8:44:59 AM (+1 min)	570.430176
8:48:59 AM (+5 min)	555.785645
8:53:59 AM (+10 min)	565.755066
8:58:59 AM (+15 min)	550.336426
9:13:59 AM (+30 min)	571.365173

Project Scope

Proof Of Concept (POC)

- Setup database infrastructure, gathering both the historical data and the real-time data from Binance exchange
- Perform data exploration and data processing
- Experiment on some baseline models: last price, exponential smoothing
- Develop training pipeline for one specific pair (BTC-USDT)
- Benchmarking our model against baseline models

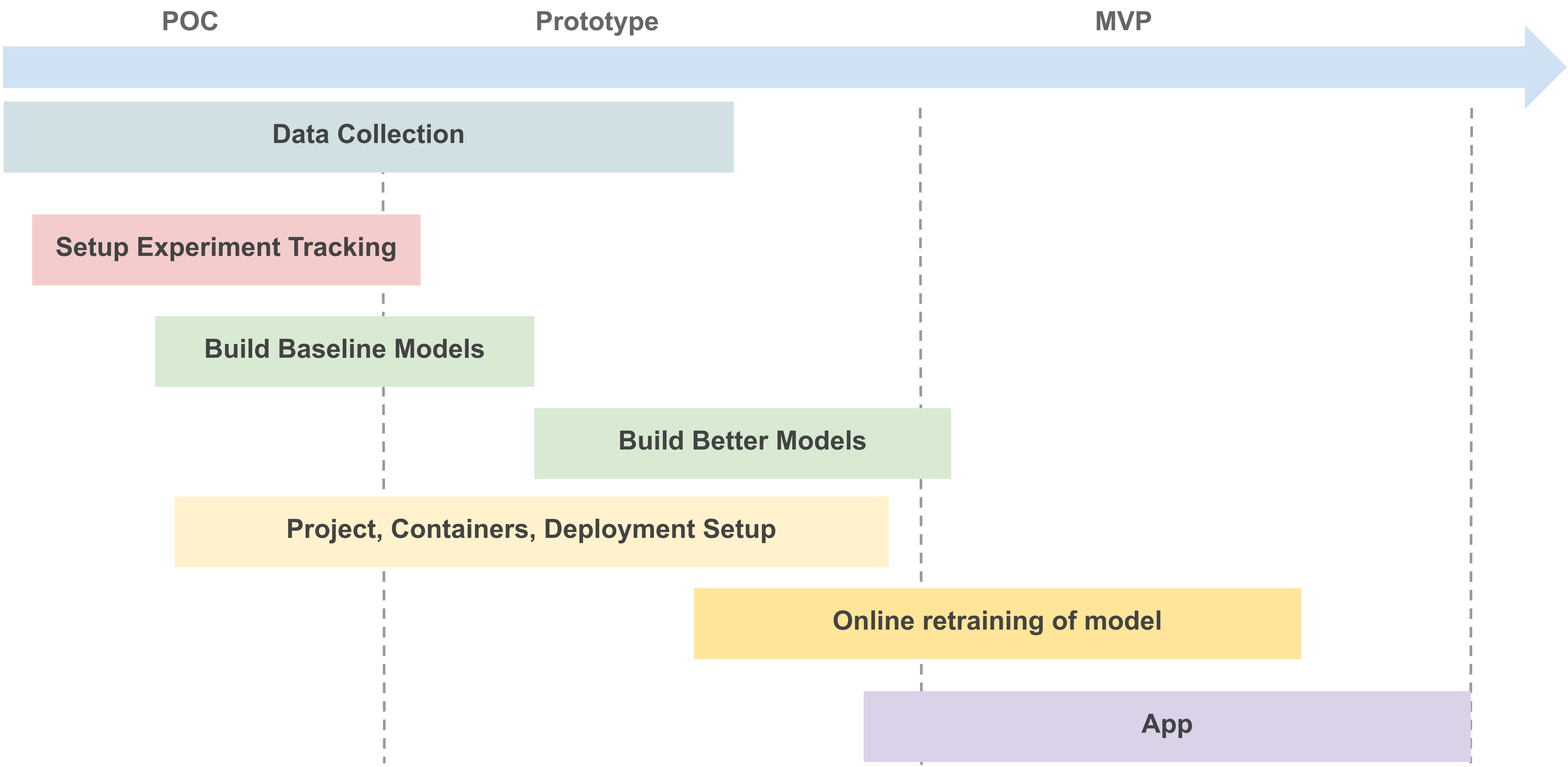
Prototype

- Develop a more advanced model, significantly improving over the different baselines
- Create a mockup of screens to see how the app will look
- Be able to launch the query of new pairs from the frontend side
- Perform regular data tests in order to verify the quality of the data
- Deploy models utilizing FastAPI to serve model predictions

Minimum Viable Product (MVP)

- Setup the front-end in order for users to interact with the API
- Provide real-time predictions using the deployed model
- Provide recommendations for user's investment
- Perform regular re-training of the deployed models

Project Workflow



Process (People)

- Collect initial data from Binance API
- Keep querying real-time data
- EDA on initial data
- Time Series modeling (single/multiple prediction approach)
- User selects a certain pair to obtain historical data and predictions
- View prediction results

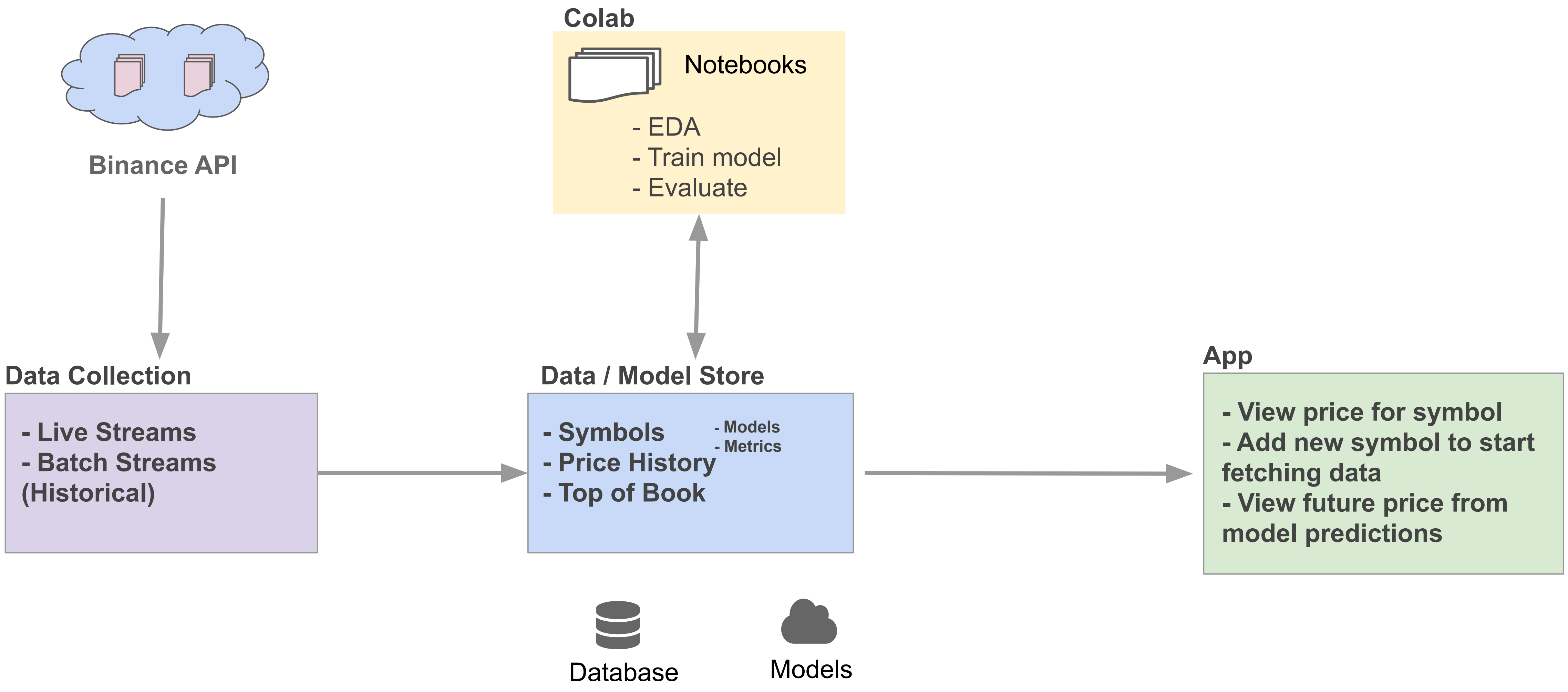
Execution (Code)

- Preprocess the initial data for both time-series and tabular modeling
- Use the best model to make prediction
- Return results to user as a minute-by-minute predictions over 24 hour period
- Track best model

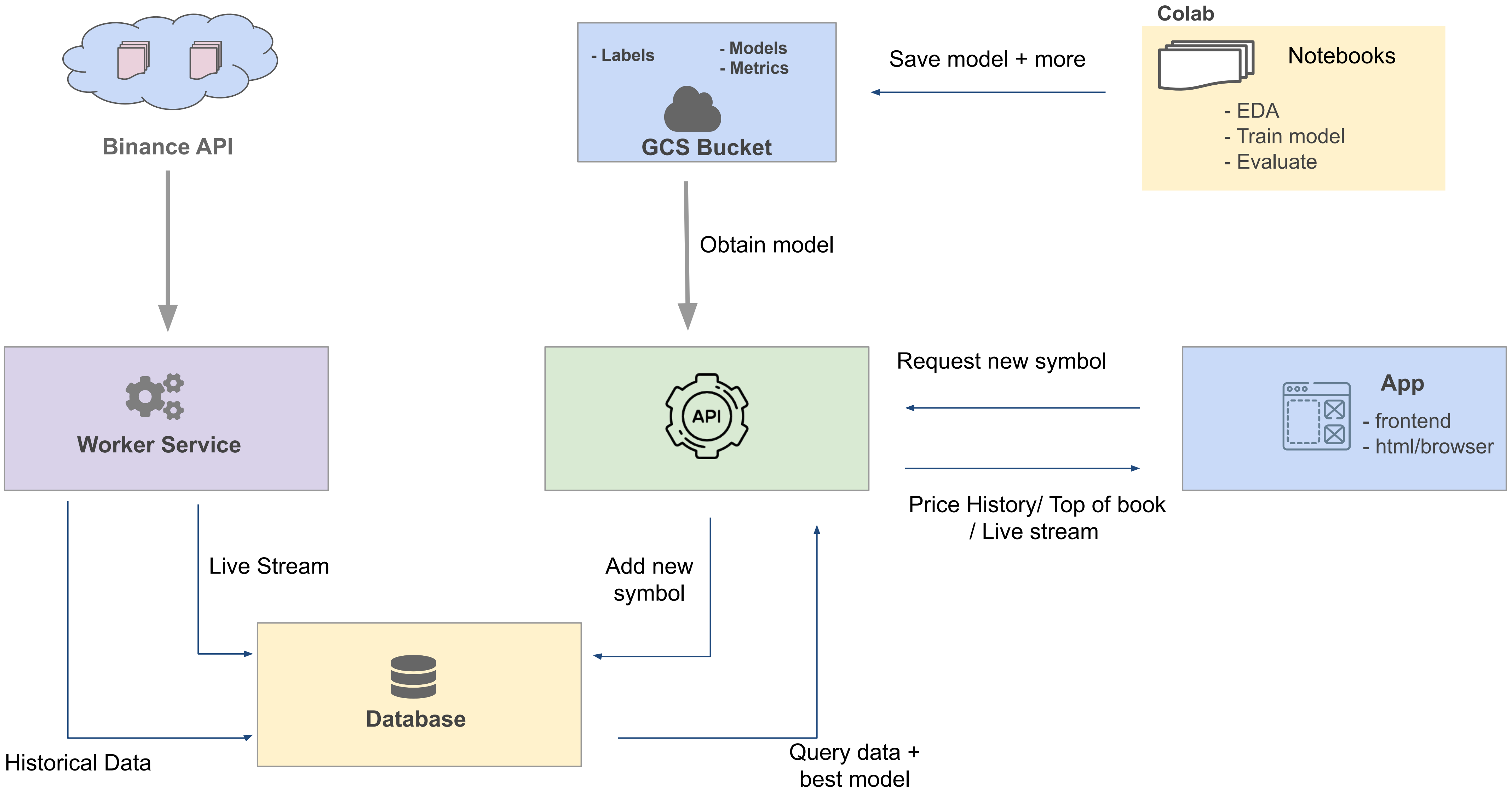
State (Source, Data, Models)

- Save data to a common store and keep updating
- Create tables and save in Postgres database
- Save model weights
- Information on pre processing

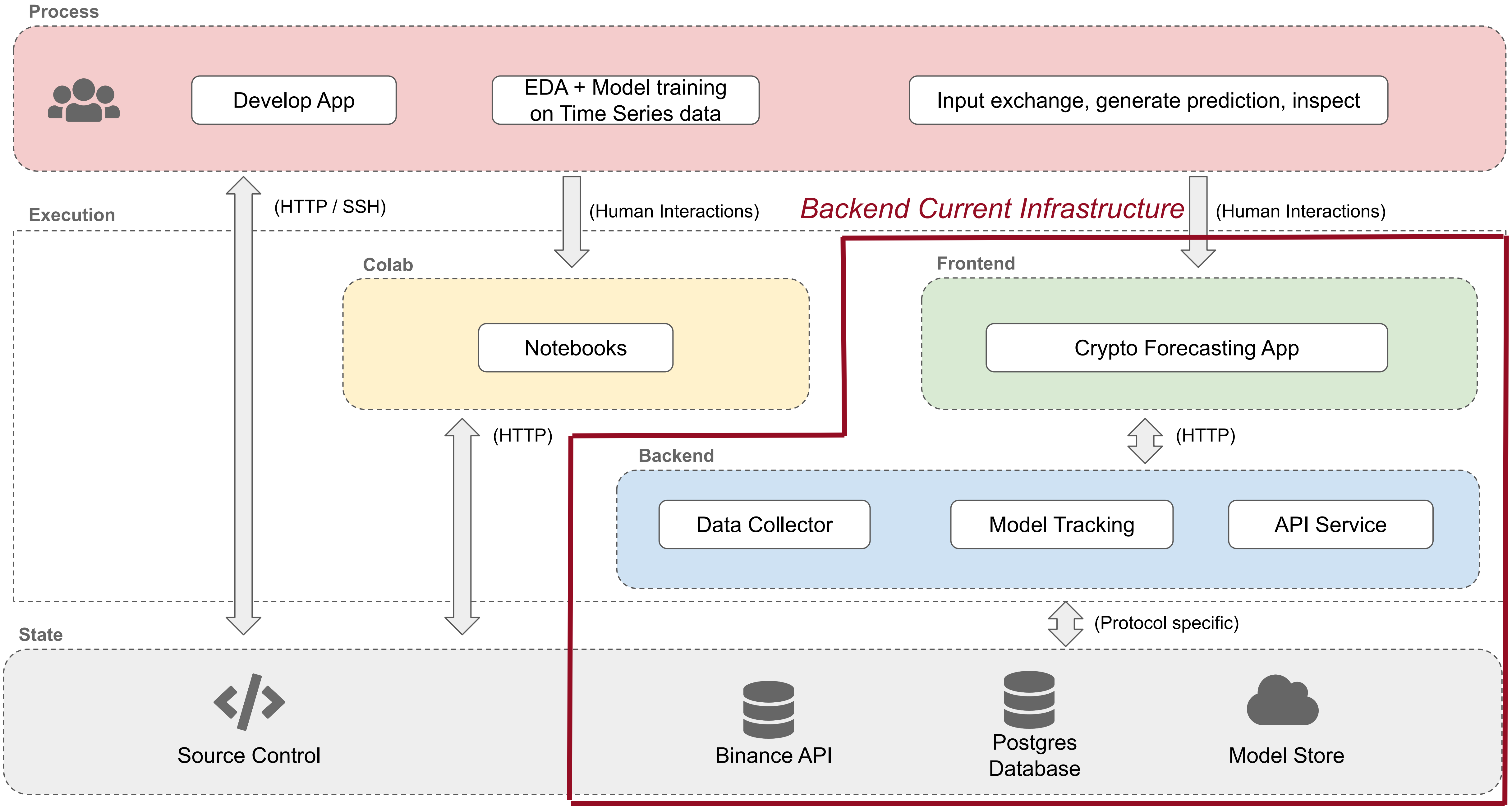
Process Flow



Data Flow

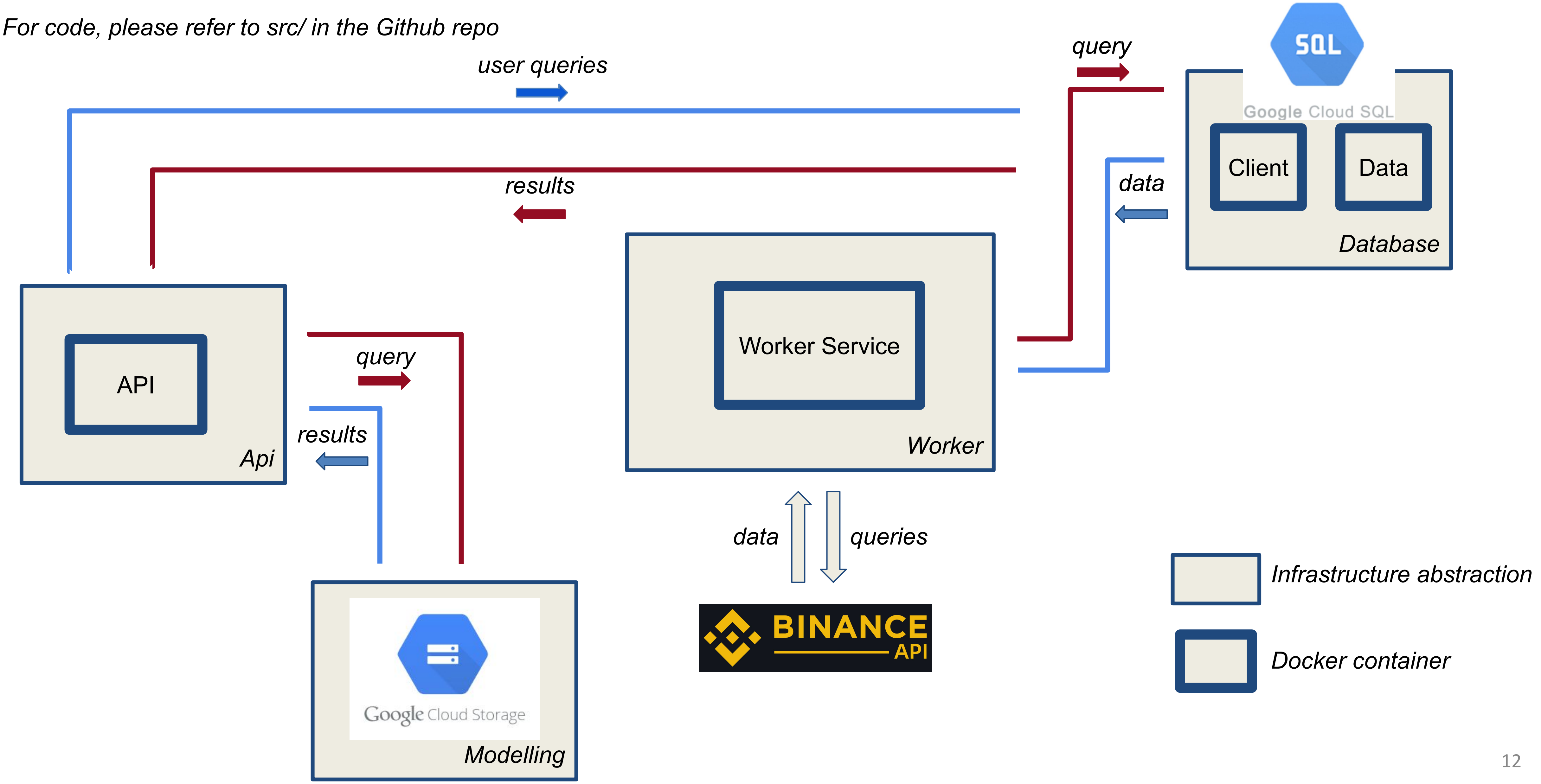


Binance API
- Labels
- Models
- Metrics
GCS Bucket
Save model + more
Colab
Notebooks
- EDA
- Train model
- Evaluate
Obtain model
Worker Service
API
Request new symbol
App
- frontend
- html/browser
Price History/ Top of book / Live stream
Live Stream
Add new symbol
Database
Historical Data
Query data + best model


Process
Develop App
EDA + Model training on Time Series data
Input exchange, generate prediction, inspect
(HTTP / SSH)
(Human Interactions)
Backend Current Infrastructure
(Human Interactions)
Execution
Colab
Notebooks
Frontend
Crypto Forecasting App
(HTTP)
(HTTP)
Backend
Data Collector
Model Tracking
API Service
(Protocol specific)
State
Source Control
Binance API
Postgres Database
Model Store

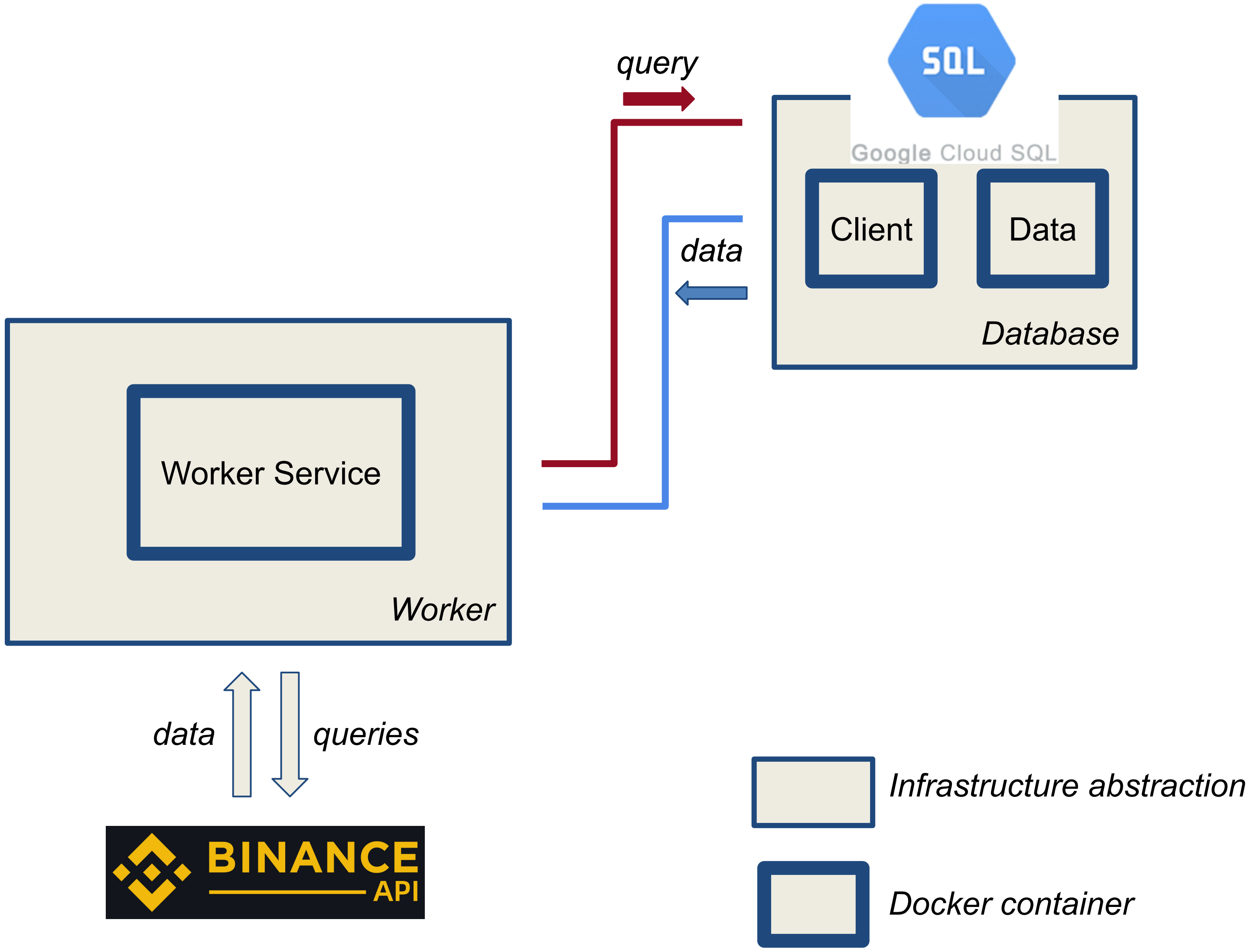
Backend Infrastructure

For code, please refer to src/ in the Github repo



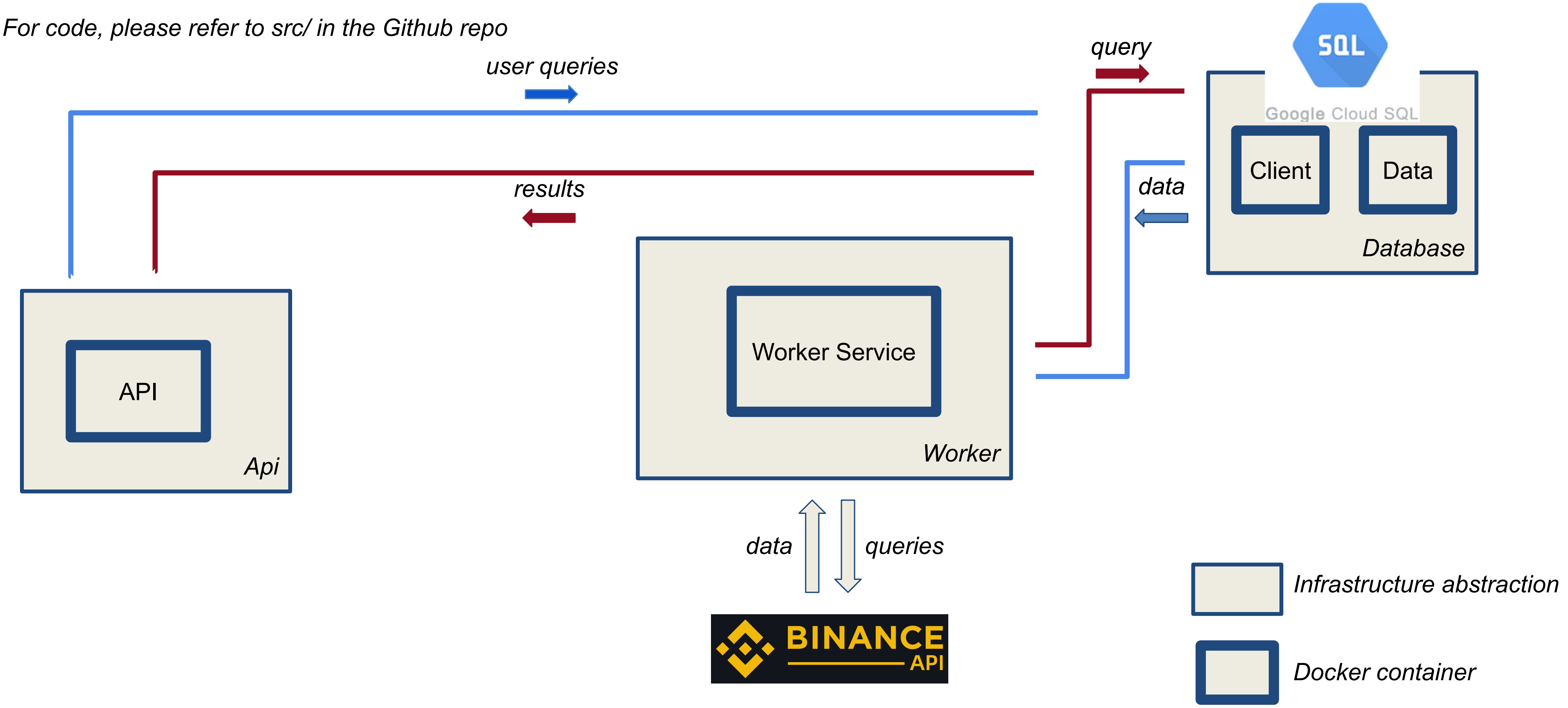
Backend Infrastructure: Update data with online streams

For code, please refer to src/ in the Github repo



Backend Infrastructure: Update data with new pairs

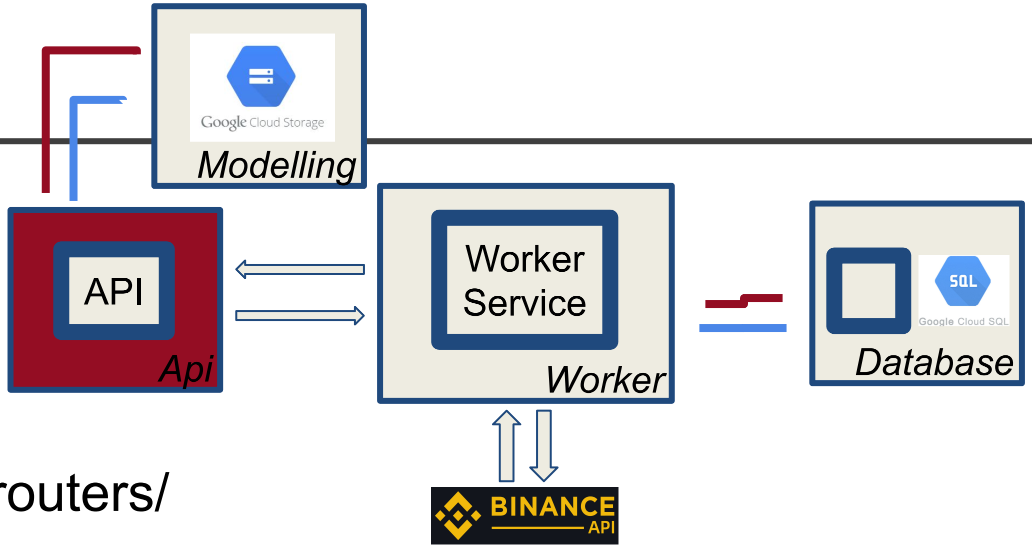
For code, please refer to src/ in the Github repo



Backend Infrastructure: Get model predictions & retraining

For code, please refer to src/ in the Github repo

Current Infrastructure: API

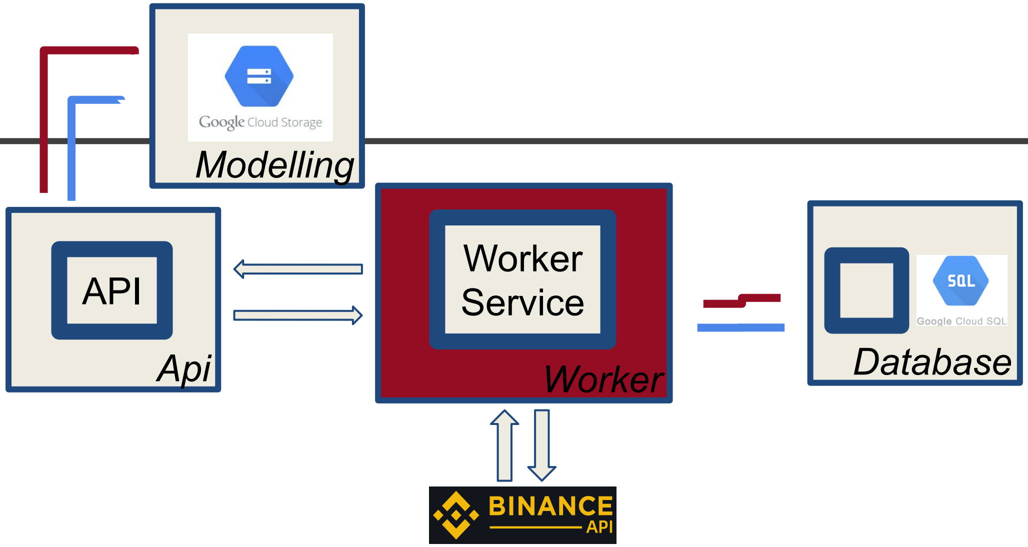


- Use FastAPI implementation of REST API
  - Implementation of tasks in api/service.py and api/routers/

- Use [Uvicorn in order to host the server]()

- Receives API call based on user input from the Frontend and either…
  - Adds symbol to *symbols* table if it's a new pair (e.g. BNBBTC)
  - Queries *price_history* table and GCS bucket for model prediction
  - Queries *price_history* and *top_of_book* table from Postgres database (either on CloudSQL or container depending on deployment type) for real-time and/or historical data for visualization purposes

Current Infrastructure: Worker



- The worker is responsible for fetching the data from Binance and writing it in the Database

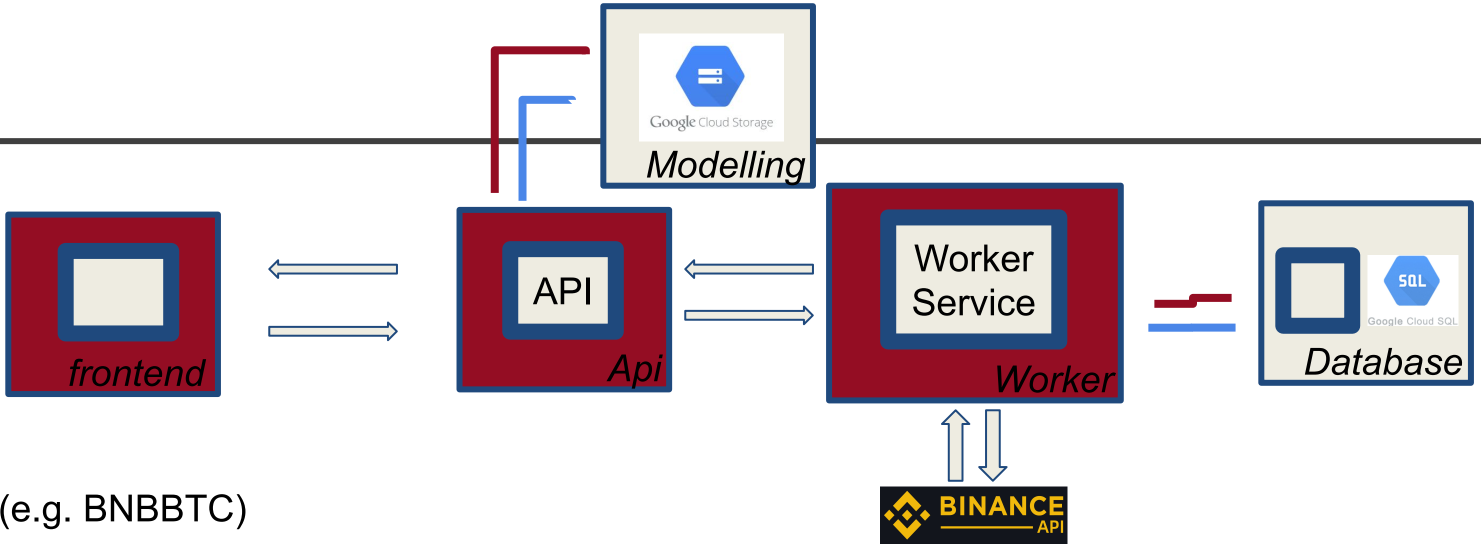
- Two types of data: Historical fetching and online streaming

- MultiProcessing:
    - One process constantly running in order to perform live streaming of the different pairs. This process uses an abstraction of a Threaded WebSocket Manager in order to be able to monitor 1000 streams at the same time
    - Several processes idle when no historical fetching happens, and then when a user queries a symbol that is not yet in the database, these processes will fetch the historical data for these new symbols

- Data Backup: when spinning up the worker, fetches the data from the last updated timestamp to the current timestamp in order for the database to contain data for every timestamp, even when the VM goes down

- Worker monitors the database for api queries

Current Infrastructure: Worker + API +Front-End

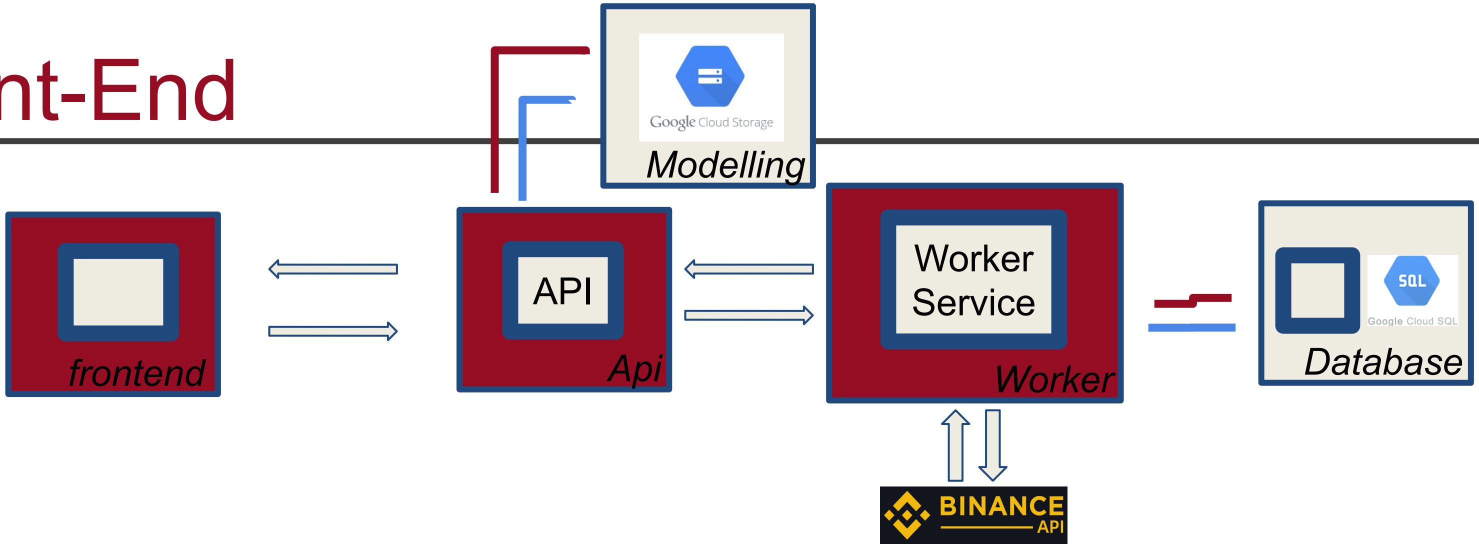


- Scenario: a user selects in the front-end side a symbol (e.g. BNBBTC)

- 2 possibilities
  - The symbol is already in the database → provide the appropriate data (top of book, candlestick) to the data
  - The symbol is not already in the database → the worker is going to fork a process that will fetch the historical data for this symbol. Once the historical fetching is done, this symbol will be added to the online streams for the 'online process' of worker service and the online data will be fetched for this symbol

- Communication between API and Worker is being done on the Database side: symbols table that contains all the important logs. The worker service is monitoring the symbols table and creates new processes based on changes on this data table

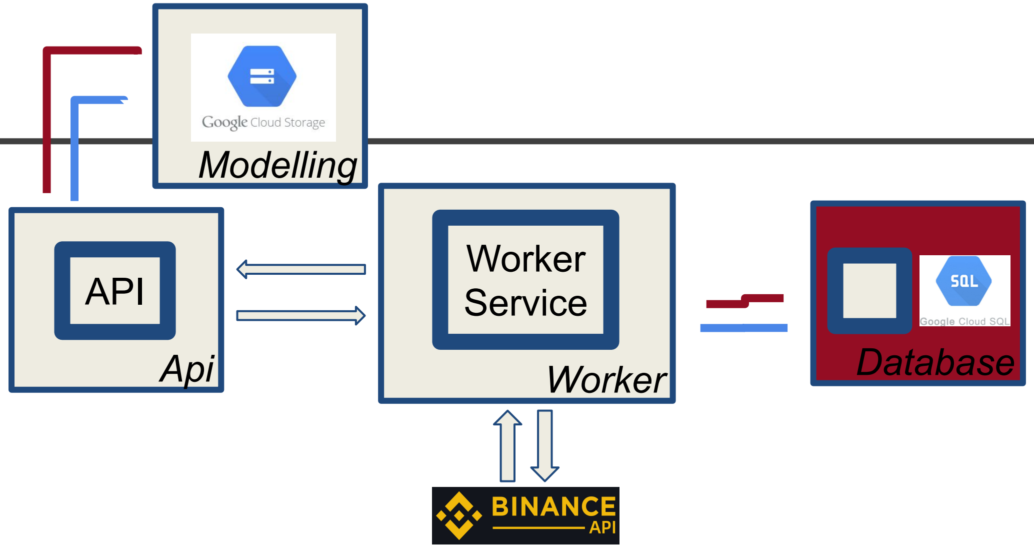
Current Infrastructure: Front-End



- Frontend Components:
    - React Javascript Library
    - Material UI, High Charts Visualization
    - Showing a live feed market data table for all symbols selected
    - Showing a visualization plot for historical data and future prediction price
    - Showing a prediction table with future prediction price

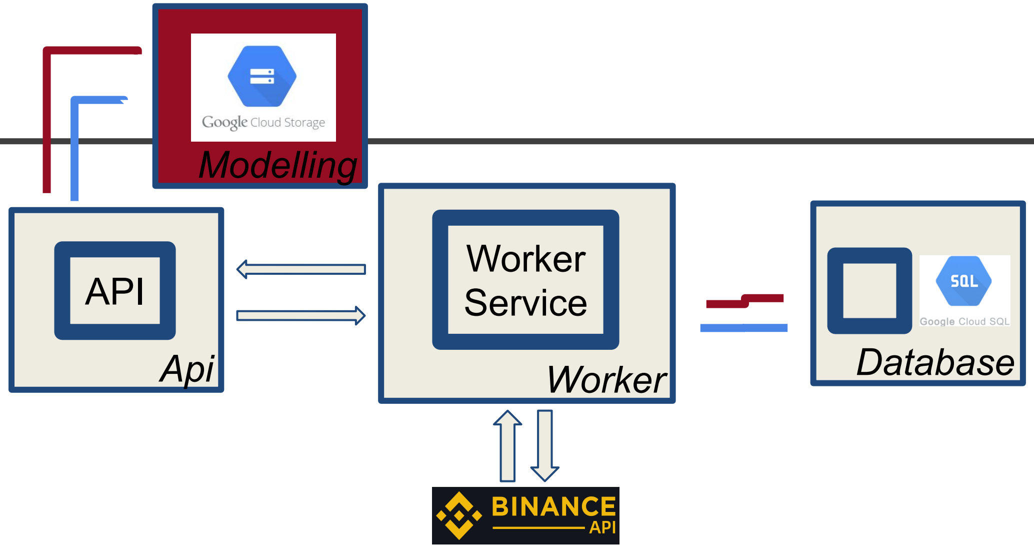
- API Calls:
    - Frontend will send selected symbol as a json dictionary to backend API service for model prediction
    - Frontend will then make API calls and get back response data from console log
    - Response data returned in json dictionary format consists of a designed interface with symbol, historic price, historical timestamp, future price, future timestamp (predicted from model) and live market data.
    - Those response data are then used in frontend for displaying time series visualization plot, live market feed and prediction price table

Current Infrastructure: Database



- Use migration code in order to monitor the different tables
- The Database is a Postgresql database
- 3 major tables:
  - Symbols: contains all the meta-information about the symbols for which we are fetching data. This table also serves the purpose of communication between API and worker - when a user queries a new symbol. It also serves the purpose of being able to get the data we missed while the worker was down
  - Price_history: contains the candle data for the different symbols. Historical data that is available for the different pairs, back until 08/17/2017
  - top_of_book: online data, will enable us to have non aggregated data on the market for visualization and backtesting purposes
- When spinning up the infrastructure for the first time, insert default symbols (the most frequent ones) in the symbols table in order for the worker to start fetching data without needing for a user querying new symbol
- Use of Google's CloudSQL to handle automatic scaling, backups, etc.

Current Infrastructure: Model



- Original models creation happens in a Google Colaboratory Notebook
- Models are uploaded to the GCS bucket
- From the GCS bucket, the API queries for the best model to use for real-time predictions
- The models are re-trained on new, incoming data on daily basis
- The raw data obtained from the GCS bucket goes to a preprocessing pipeline before being fed to the model (more information on the preprocessing pipeline on subsequent slides)
- The training architecture and the data preprocessing pipeline are common for all pairs. The only major difference is the raw data fed into the preprocessing pipeline.
- Future steps: automated model training pipeline utilizing VertexAI that independently initializes model training once a user enters a new symbol through the frontend API

Dataset(s)

For code, please refer to src/api_for_data_download in the Github repo

- Dataset(s):
  - Historical data queried from Binance API (dtype: candlesticks)
    - earliest timestamp for the BTC-USDT pair: 2017-08-17 04:00:00
  - Real time data updating from Binance API (dtype: candlesticks) using a websocket

- Dataset(s) size:
  - Number of datasets: 1,612 (one dataset per pair)
  - Size of dataset per pair (~0.3Gb)
  - Total dataset(s) size (~500Gb)

- For the EDA and the initial modelling phases focus on the BTC-USDT pair
- The modelling extended to 5 pairs for the final app (BTC-USDT, BNB-BTC, BNB-USDT, ETH-BTC, ETH-USDT)

Raw Dataset(s) Features

Candle Feature	Description
Open Time	Candle Open Time
Open	Open Price in Quote (Secondary) Asset Units
High	High Price in Quote (Secondary) Asset Units
Low	Low Price in Quote (Secondary) Asset Units
Close	Close Price in Quote (Secondary) Asset Units
Volume	Total Trade Volume in Base (Primary) Asset Units
Close Time	Candle Close Time
Qupte Asset Volume	Total Trade Volume in Quote (Secondary Asset Units
Number of Trades	Total Number of Trades
Taker Buy Base Asset Volume	Taker (Matching Existing Order) Buy Base Asset Volume
Taker Buy Quote Asset Volume	Taker (Mathcing Existing Order) Buy Quote Asset Volume
Ignore	Safe to Ignore

Dataset(s) Quality

Data Types:

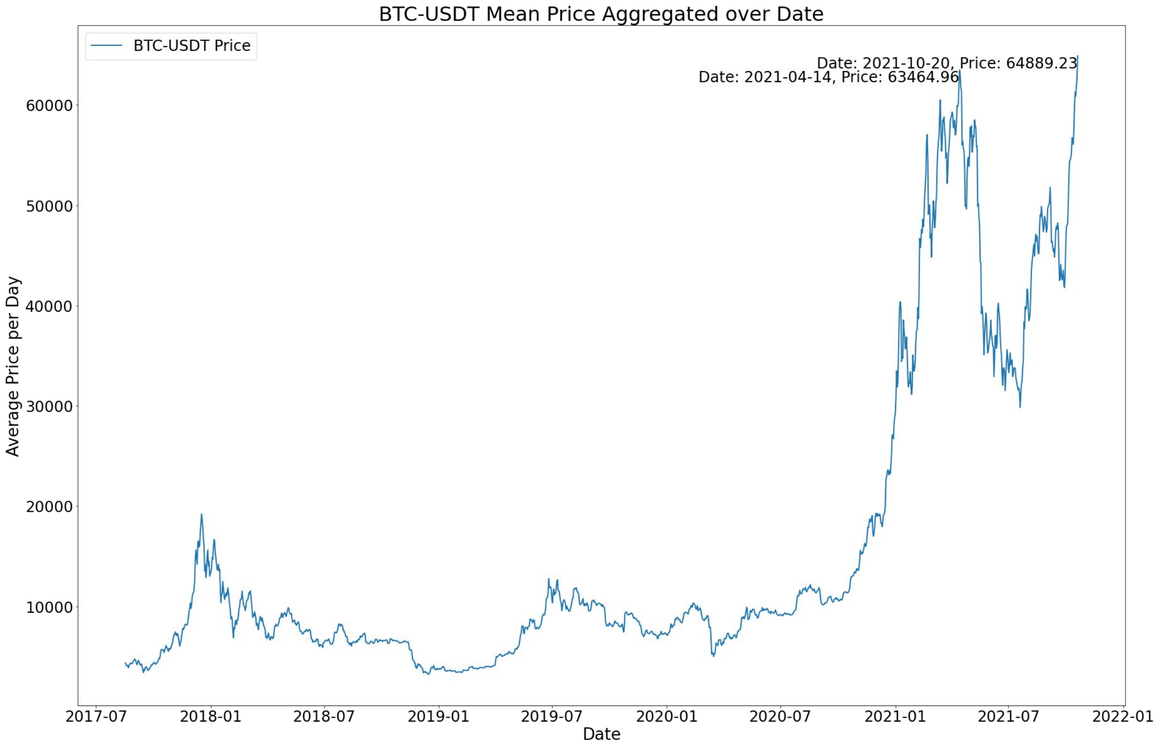
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2188604 entries, 0 to 2188603
Data columns (total 12 columns):
 #   Column                        Dtype
---  ------                        -----
 0   Open Time                     int64
 1   Open Price                    float64
 2   High price                    float64
 3   Low Price                     float64
 4   Close Price                   float64
 5   Volume Traded                 float64
 6   Close Time                    int64
 7   Quote asset Volume            float64
 8   Number of Trades              int64
 9   Taker buy base asset volume   float64
 10  Taker buy quote asset volume  float64
 11  NA                            float64
dtypes: float64(9), int64(3)
memory usage: 217.1 MB
```

Missing Values:

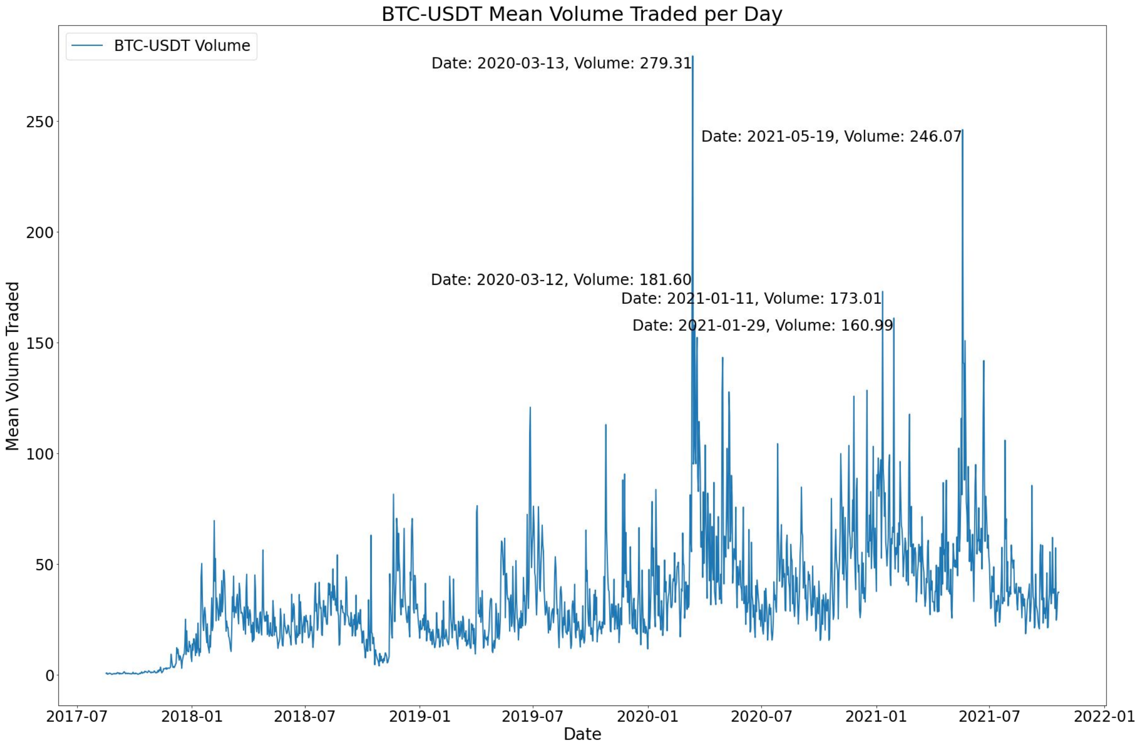
```
Open Time                       0
Open Price                      0
High price                      0
Low Price                       0
Close Price                     0
Volume Traded                   0
Close Time                      0
Quote asset Volume              0
Number of Trades                0
Taker buy base asset volume     0
Taker buy quote asset volume    0
NA                              0
dtype: int64
```

EDA - Time Series Data

Mean Price per Day



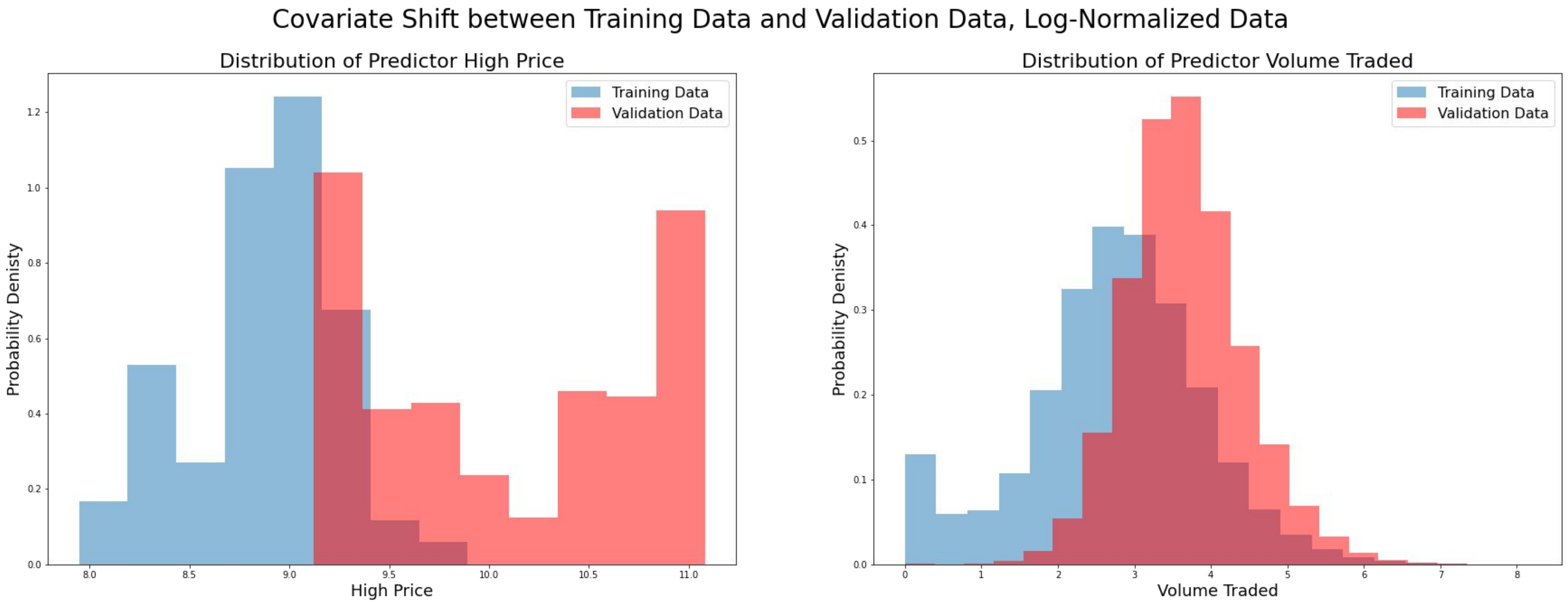
Mean Volume per Day



Key observations:

- The relative value of BTC-USDT increases rapidly due to COVID-19 pandemic
- The trading volume of BTC-USDT is fairly flat, with a few spikes due to COVID-19 pandemic

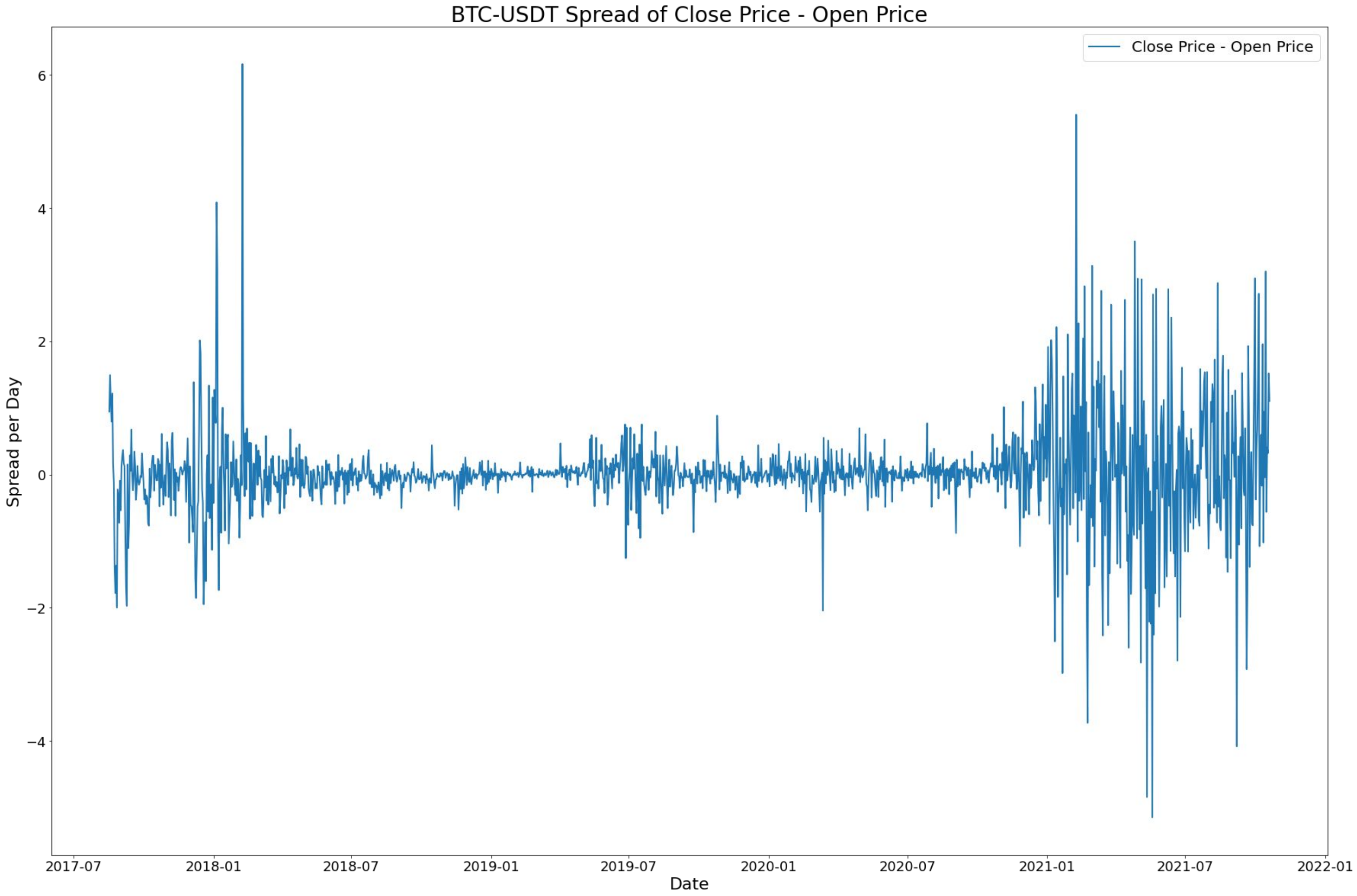
EDA - Out of Distribution Validation Data



Key observations:

- Observable covariate shift in variable `High Price`
- Behavior hinted in the previous slide; validation data post COVID-19

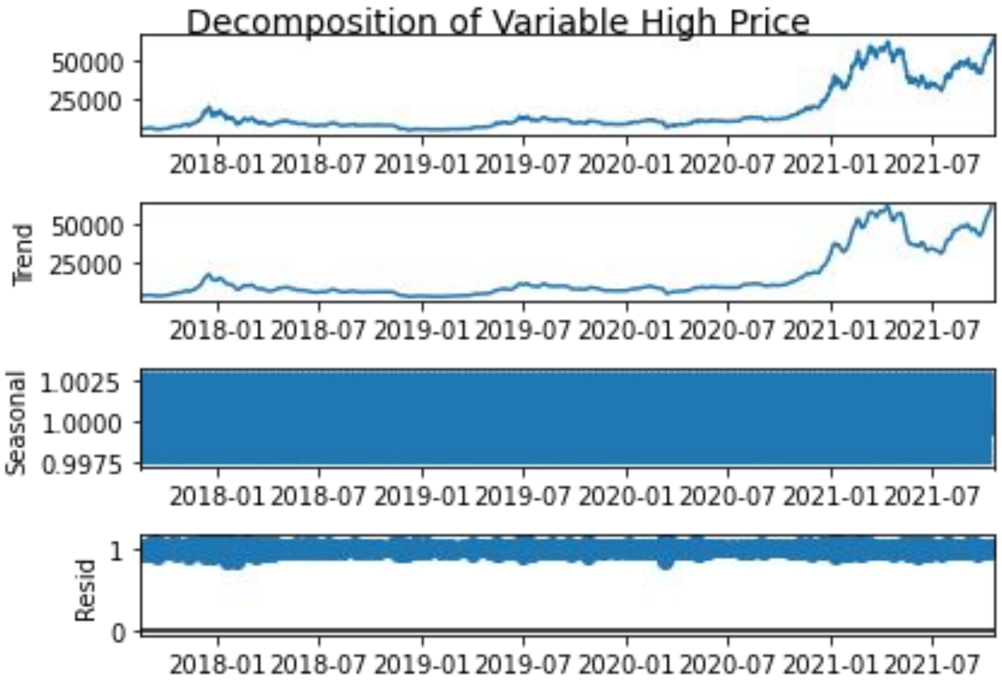
EDA - Response Variable(s)

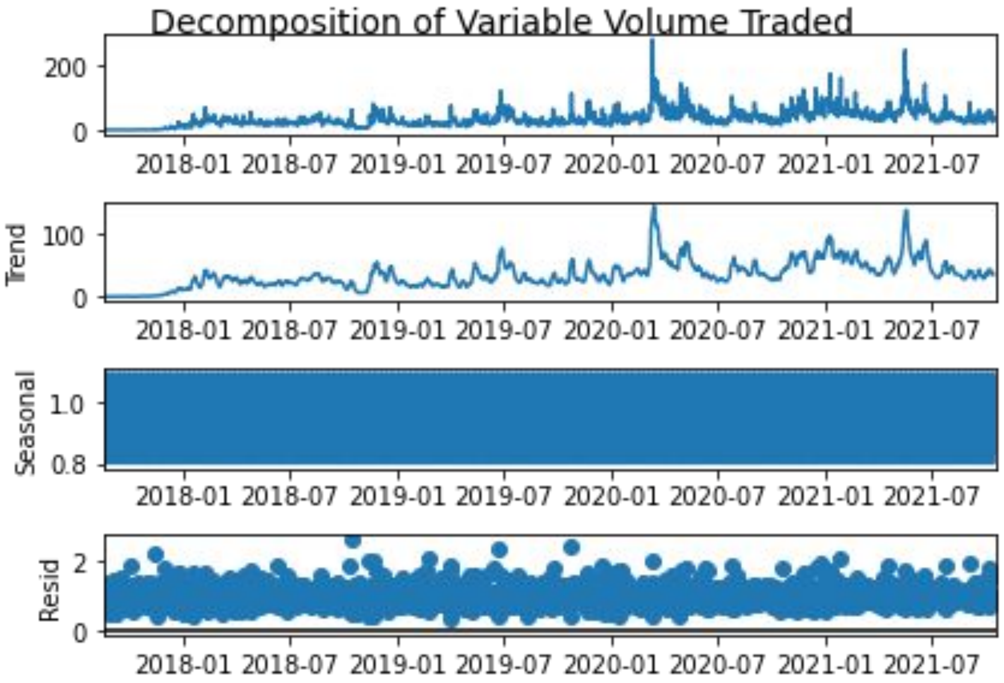


Key observations:

- The spread of `Open Price` - `Close Price` is fairly constant before COVID-19 pandemic
- Since start of COVID-19 pandemic, significantly larger volatility
- This information should be taken into account during modelling/choosing response variable

EDA - Variable Decomposition





Key observations:

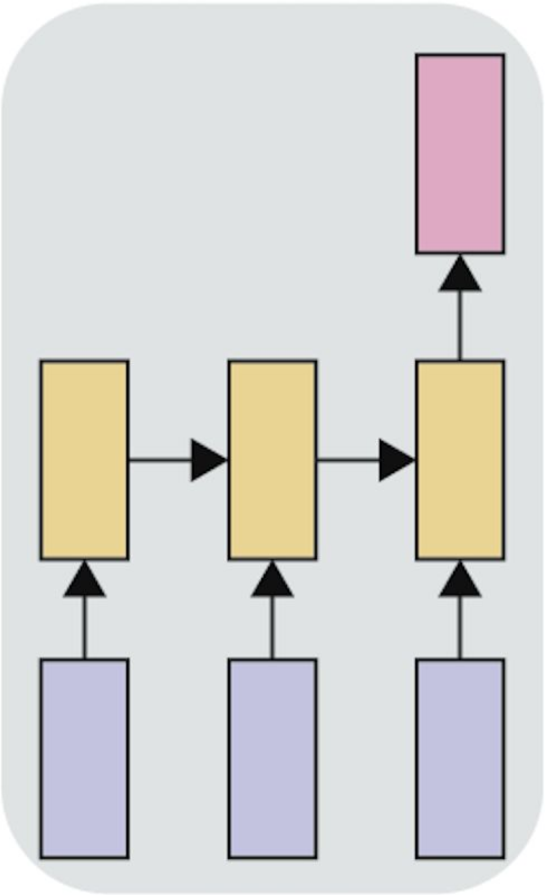
- Visible trend for variable `High Price`, as already hinted in previous EDA slides
- No clear trend for variable `Volume Traded`

Modeling Decisions

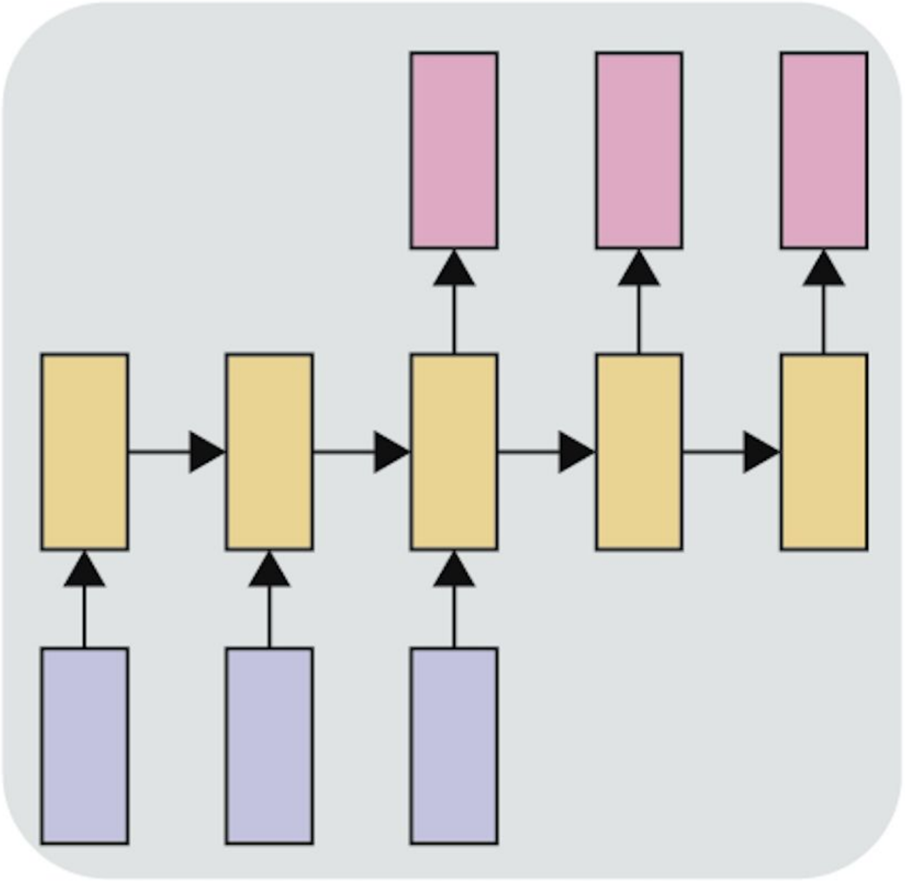
- 80-10-10 train-validation-test split (HP tuning models)
- 100-0-0 train-validation-test split (Models pushed to GCS)
- Feature Engineering for Tensorflow Modeling:
  - Remove *Close Time, Open Time, NA*
  - Time-based features
  - Statistical features
  - Domain knowledge-based features
  - Log-transform data (numerical features)
  - Standardize data (whole dataset)
- Metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE)
- Predict: standardized `mid_true - mid_baseline`

Modeling Approaches

Previous: Multi-input, Single-output
given the current input, the model infers the value of the pre-defined pair, one time step in the future



Current: Multi-input, Multi-output
given the current input, the model infers the value of the pre-defined pair, X time steps in the future



[Credit]

Models

- **Baseline - Persistent Model**
  - For Multi-input, Single-output predicts the Close Price of the next time step to be the same as the Close Price of the current time step
  - For Multi-input, Multi-output predicts the Close Price of the next X time steps to be the same as the Close Price of the current time step
- **Second Iteration - LSTM on Standardized Raw Features**
  - For Multi-input, Single-output predicts the Close Price of the next time step based on the input features of the previous X time steps
  - For Multi-input, Single-output predicts the Close Price of the X next time steps based on the input features of the previous X time steps
- **Final Model - LSTM on Engineered Features**
  - Feature Engineering on input data: Log-transformed, Standardized, Time-based features, Statistical features, Domain knowledge-based features)
  - Feature Engineering on output data: Transformed the output feature to standardized \`mid_true - mid_baseline\`

Models - Model Tuning Results TO ADD

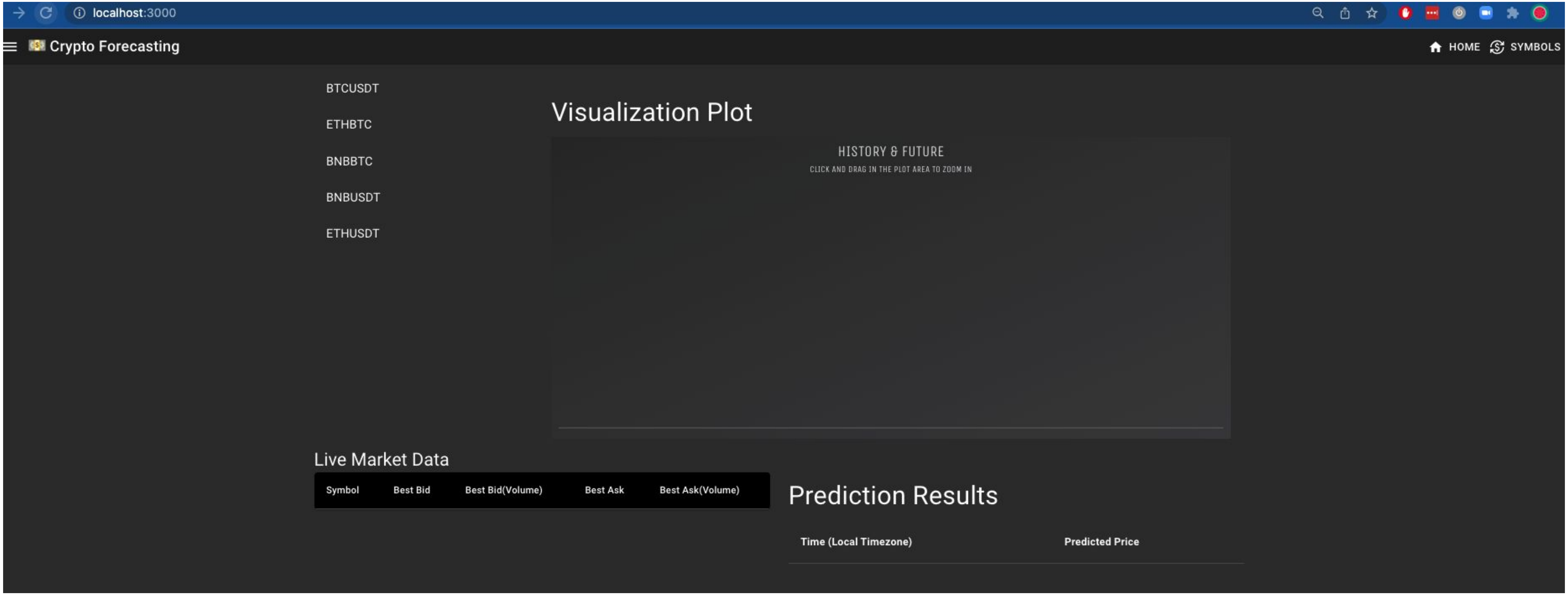
Models	Train MSE	Train MAE	Validation MSE	Validation MAE
Baseline - Single Output (SO) - /	2.8378e-6	0.0007	9.2235e-6	0.0020
Baseline - Multi Output (MO) - /	/	/	/	/
LSTM - Standardized Raw Features (SO) - input_seq_len = 32	1.0474e-5	0.0017	9.1504e-5	0.0044
LSTM - Standardized Raw Features (MO) - input_seq_len = 32	5.4911e-5	0.0035	1.6730e-4	0.0084
LSTM - Engineered Features (SO) - /	/	/	/	/
LSTM - Engineered Features (MO) - input_seq_len = 8	6.8703e-5	0.004	1.7115e-4	0.0091

*Note: The models were trained on the BTC-USDT pair. It is important to bear in mind that the target variable for the Baseline (SO), LSTM - Standardized Raw Features (SO) and LSTM - Standardized Raw Features (MO) models was `Close Price`. However, for LSTM - Engineered Features (MO) the target variable was altered to 'mid_true - min_baseline`. Furthermore, for the LSTM - Standardized Raw Features (MO) we used `input_seq_len` = 32, while for the LSTM - Engineered Features (MO) we used `input_seq_len` = 8 due to compute resource limitations.**

Frontend - React

- We used React javascript library to build our frontend UI
- Our UI consists of the following **components**
    - Time series plot with zoom in feature *(highcharts)*
        - *Green color: Historic price*
        - *Red color: Future price prediction up to 32 minutes later*
        - *Allow user to zoom in and reset at any period of time*
    - Selection of pair for prediction *(material UI)*
    - Prediction results table
    - Live market data table

Frontend - Start-Up Screen



Frontend View

Not Secure | 34.70.108.117

Crypto Forecasting

HOME SYMBOLS

- BTCUSDT
- ETHBTC
- BNBBTC
- BNBUSDT
- ETHUSDT

Visualization Plot

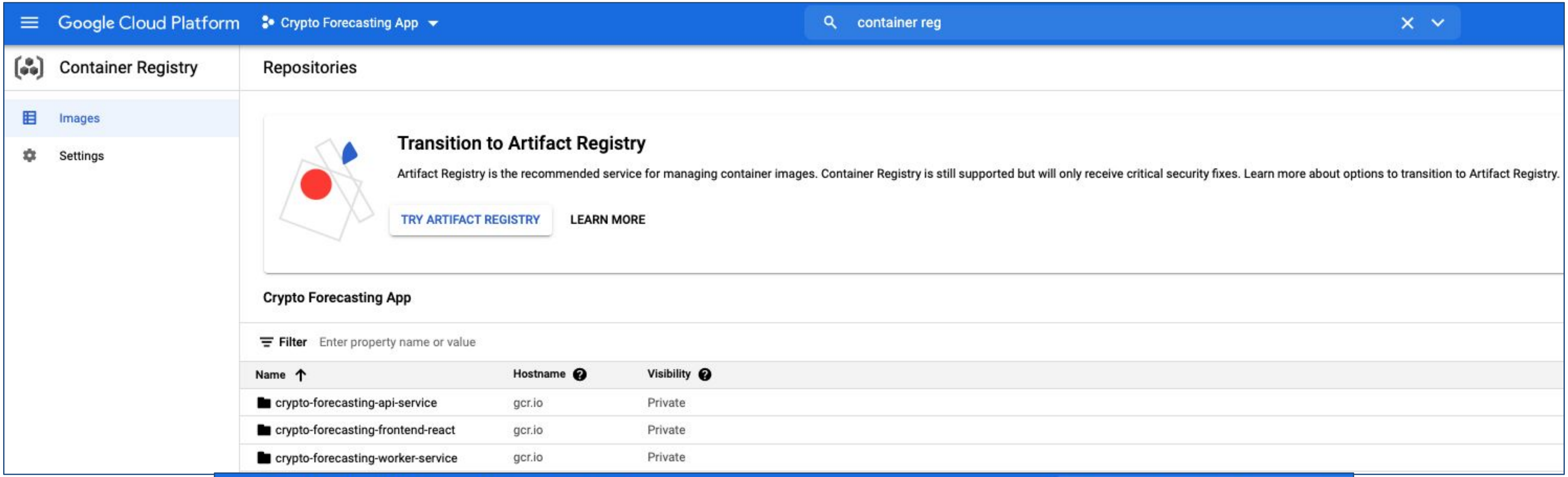
Live Market Data

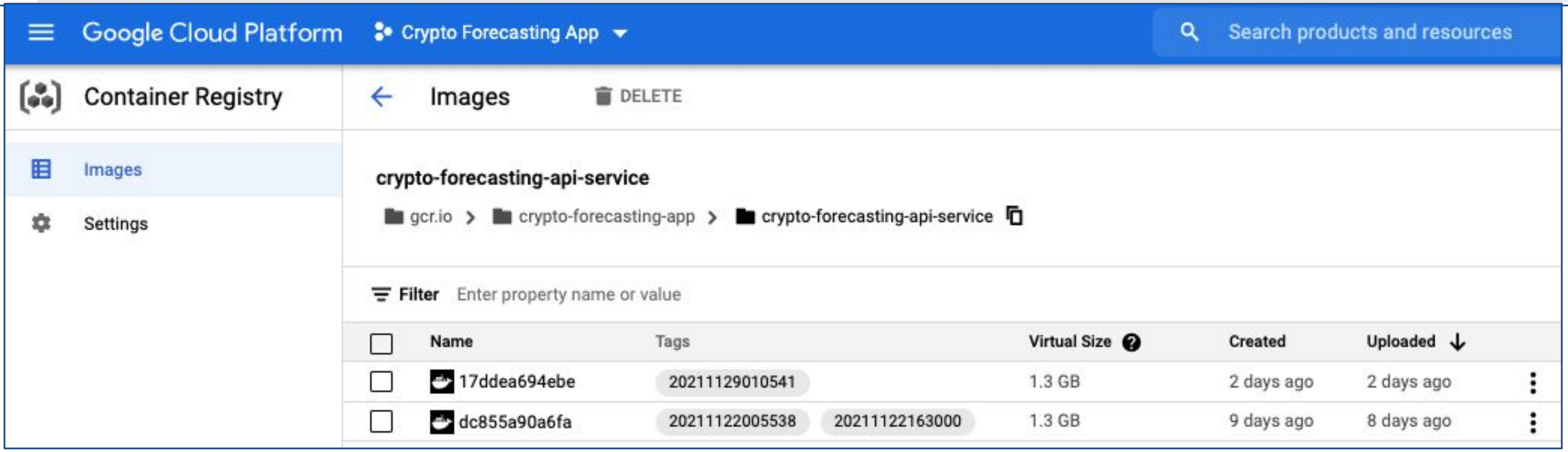
Symbol	Best Bid	Best Bid(Volume)	Best Ask	Best Ask(Volume)
ETHBTC	0.080364	0.0323	0.080371	1.4806
ETHUSDT	3,791.23	2.4	3,791.72	0.2083
BTCUSDT	47,171.01	0.30931	47,171.02	0.34701
BNBBTC	0.011203	2.541	0.011204	3.357

Prediction Results

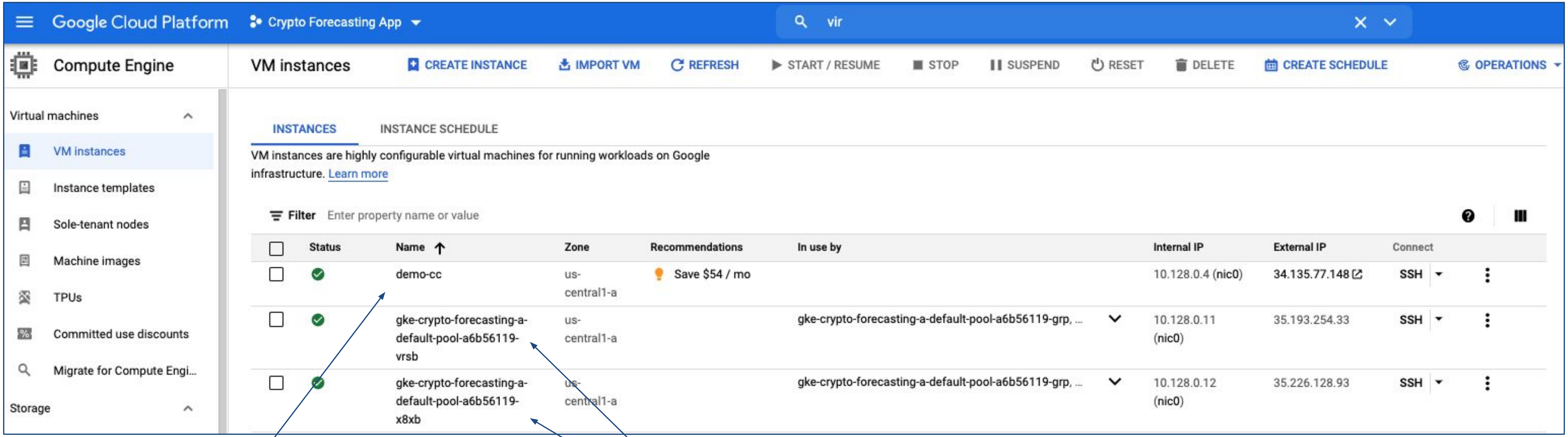
Time (Local Timezone)	Predicted Price
8:43:59 AM (+0 min)	574.734924
8:44:59 AM (+1 min)	570.430176
8:48:59 AM (+5 min)	555.785645
8:53:59 AM (+10 min)	565.755066
8:58:59 AM (+15 min)	550.336426
9:13:59 AM (+30 min)	571.365173

Deployment - Container Registry





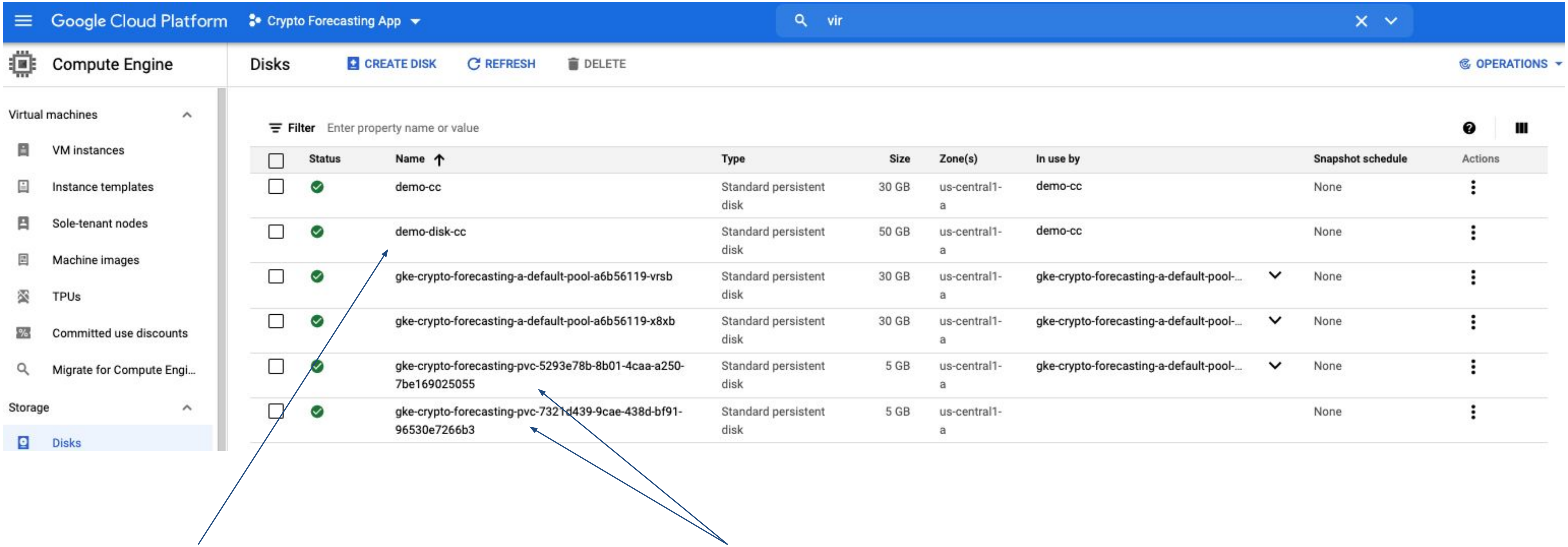
Deployment - Compute Engine



Ansible Deployment

Kubernetes Deployment

Deployment - Persistent Disk (Saving Postgres Database)



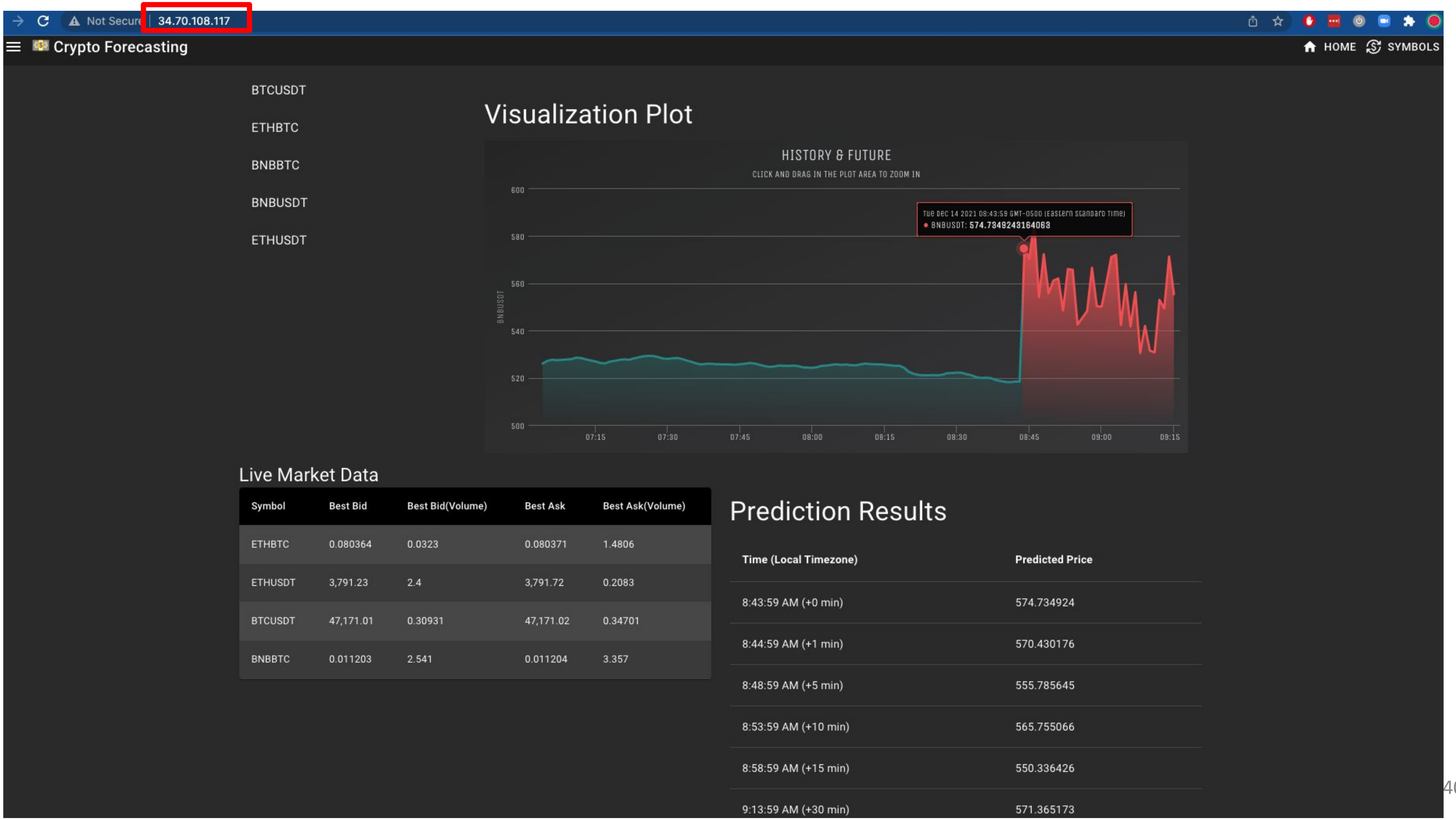
Ansible Deployment

Kubernetes Deployment

Standard Deployment - Virtual Machine SSH

```
connorcapitolo_g_harvard_edu@demo-cc:~$ sudo docker image ls
REPOSITORY                                                    TAG              IMAGE ID       CREATED         SIZE
gcr.io/crypto-forecasting-app/crypto-forecasting-worker-service   20211129010541   7998cf254ecb   37 hours ago    650MB
gcr.io/crypto-forecasting-app/crypto-forecasting-api-service      20211129010541   e64b373aed92   37 hours ago    2.63GB
gcr.io/crypto-forecasting-app/crypto-forecasting-frontend-react   20211129010541   edfc06298332   37 hours ago    162MB
postgres                                                      latest           577410342f45   12 days ago     374MB
nginx                                                         stable           aedf7f31bdab   13 days ago     141MB
connorcapitolo_g_harvard_edu@demo-cc:~$ sudo docker container ls
CONTAINER ID   IMAGE                                                                         COMMAND                  CREATED        STATUS         PORTS                                      NAMES
7d2992b474fe   nginx:stable                                                                  "/docker-entrypoint.…"   37 hours ago   Up 37 hours    0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp   nginx
7d5bc228417a   gcr.io/crypto-forecasting-app/crypto-forecasting-worker-service:20211129010541   "/bin/bash ./docker-…"   37 hours ago   Up 37 hours                                               worker-service
78cf76469a44   gcr.io/crypto-forecasting-app/crypto-forecasting-api-service:20211129010541      "/bin/bash ./docker-…"   37 hours ago   Up 37 hours    0.0.0.0:9600->9000/tcp                     api-service
90b0a0ffbd4e   postgres:latest                                                               "docker-entrypoint.s…"   37 hours ago   Up 37 hours    0.0.0.0:5432->5432/tcp                     postgres
8ab2619b5094   gcr.io/crypto-forecasting-app/crypto-forecasting-frontend-react:20211129010541   "/docker-entrypoint.…"   37 hours ago   Up 37 hours    0.0.0.0:3000->80/tcp                       frontend
```

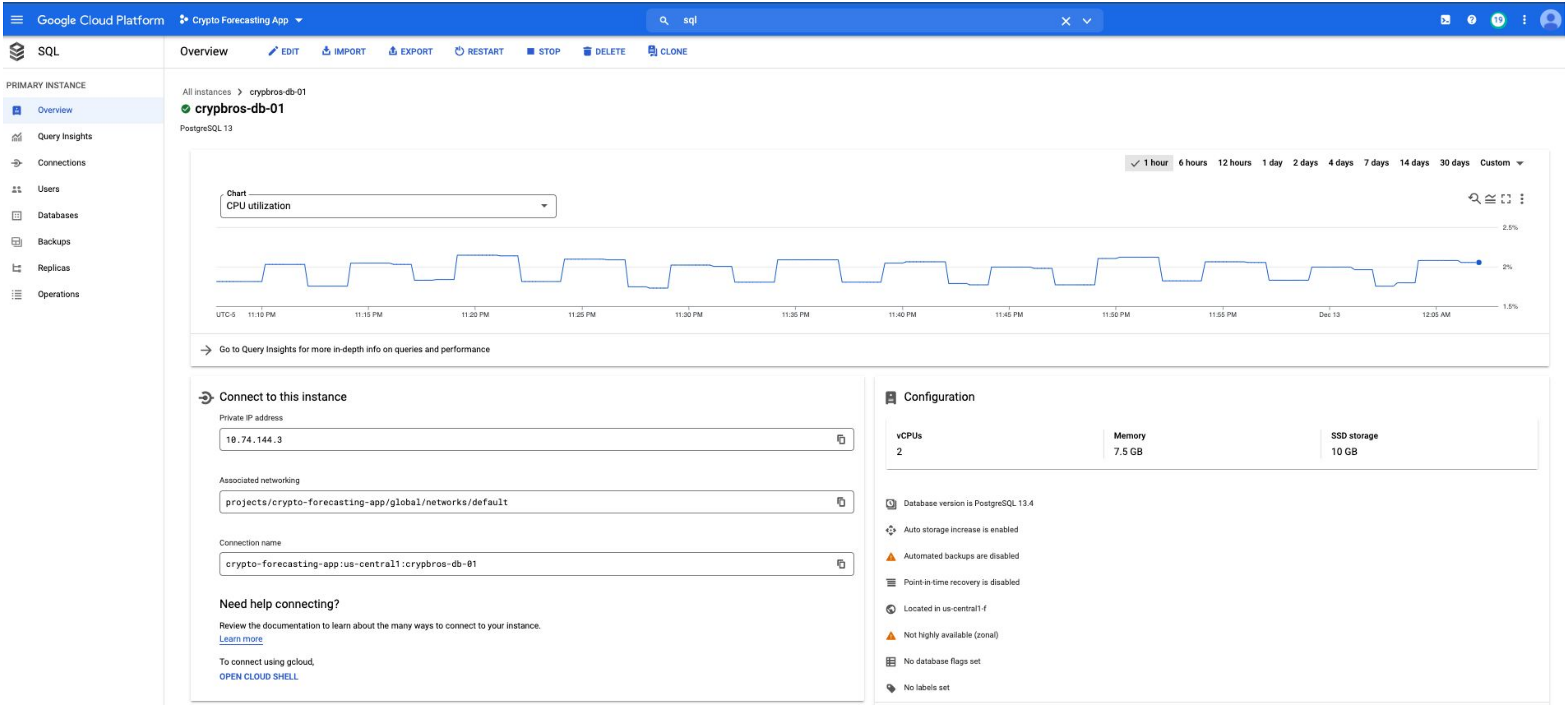
Standard Deployment - Crypto Forecasting App



Kubernetes Deployment - Cloud SQL Script

```
root@afe289233bf4:/app# ansible-playbook deploy-setup-cloud-sql.yml -i inventory.yml

PLAY [Configure Cloud SQL] ****************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************************
[DEPRECATION WARNING]: Distribution debian 10.11 on host 34.121.219.119 should use /usr/bin/python3, but is using /usr/bin/python for backward compatibility with prior Ansible releases. A future
Ansible release will default to using the discovered platform python for this host. See https://docs.ansible.com/ansible-core/2.11/reference_appendices/interpreter_discovery.html for more information.
This feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [34.121.219.119]

TASK [Copy db migration scripts] **********************************************************************************************************************
ok: [34.121.219.119]

TASK [Run DB migrations using dbmate] *****************************************************************************************************************
changed: [34.121.219.119]

PLAY RECAP ********************************************************************************************************************************************
34.121.219.119             : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

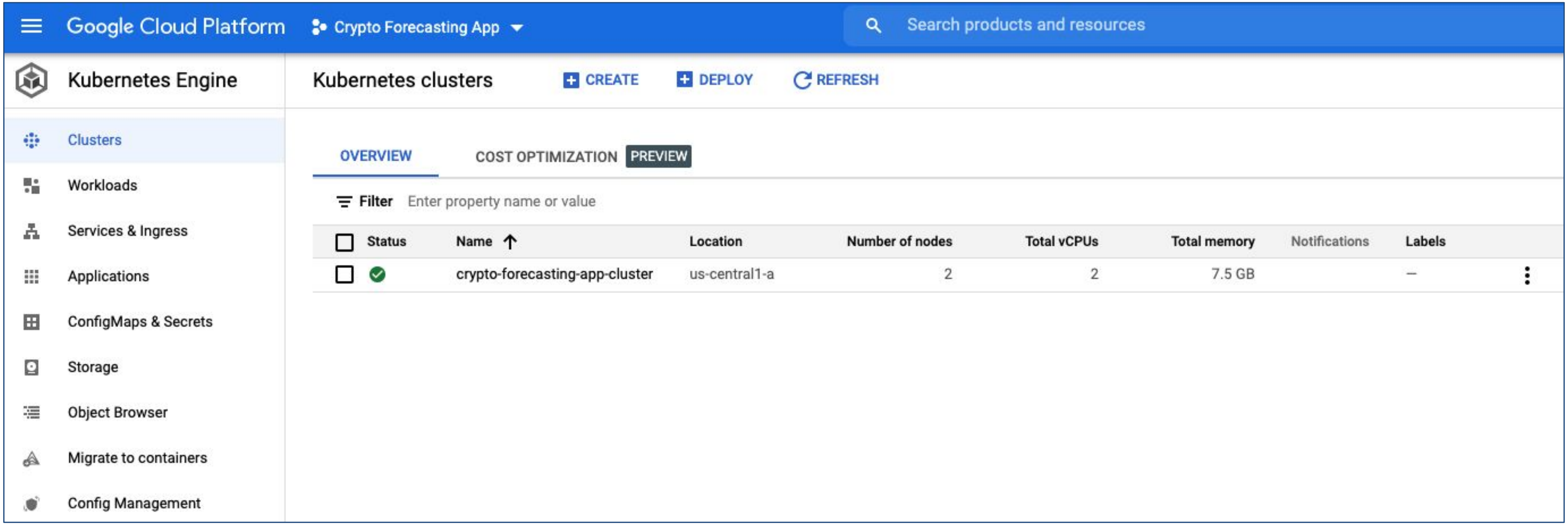
Kubernetes Deployment - Google Cloud SQL



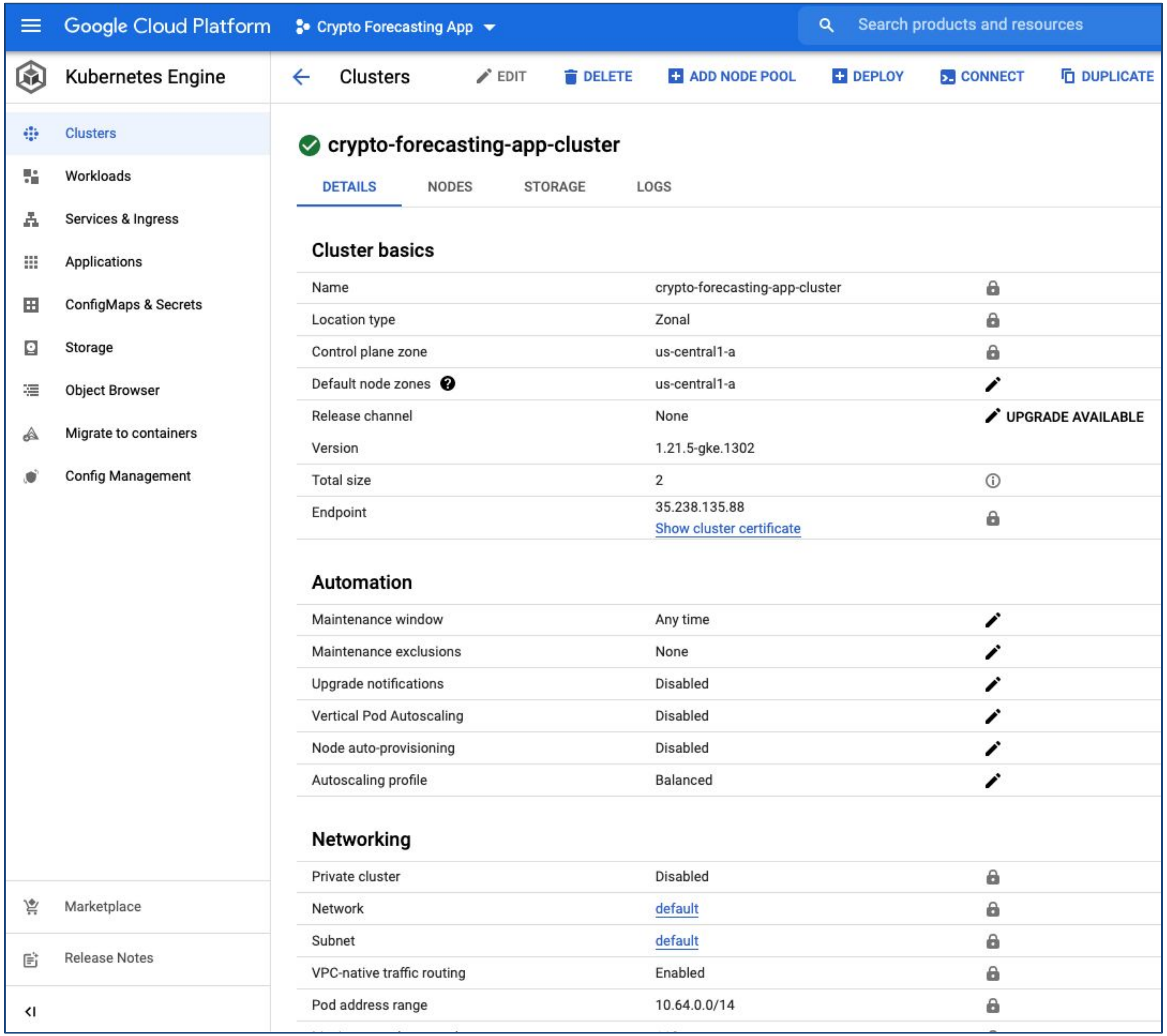
Kubernetes Deployment - Kubernetes Script (Partial View)

```
root@425fa92ea59f:/app# ansible-playbook deploy-k8s-cluster.yml -i inventory.yml --extra-vars cluster_state=present
[DEPRECATION WARNING]: community.kubernetes.helm_repository has been deprecated. The community.kubernetes collection is being renamed to kube
rnetes.core. Please update your FQCNs to
kubernetes.core instead. This feature will be removed from community.kubernetes in version 3.0.0. Deprecation warnings can be disabled by set
ting deprecation_warnings=False in ansible.cfg.
[DEPRECATION WARNING]: community.kubernetes.helm has been deprecated. The community.kubernetes collection is being renamed to kubernetes.core
. Please update your FQCNs to kubernetes.core
instead. This feature will be removed from community.kubernetes in version 3.0.0. Deprecation warnings can be disabled by setting deprecation
_warnings=False in ansible.cfg.

PLAY [Create Kubernetes Cluster and deploy multiple containers] ****************************************************************************
************************************************

TASK [Create a GKE cluster] ****************************************************************************************************************
************************************************
changed: [localhost]

TASK [Create a Node Pool] ******************************************************************************************************************
************************************************
changed: [localhost]

TASK [Connect to cluster (update kubeconfig)] **********************************************************************************************
************************************************
changed: [localhost]

TASK [Create Namespace] ********************************************************************************************************************
************************************************
changed: [localhost]

TASK [Add nginx-ingress helm repo] *********************************************************************************************************
************************************************
ok: [localhost]

TASK [Install nginx-ingress] ***************************************************************************************************************
************************************************
changed: [localhost]

TASK [Copy docker tag file] ****************************************************************************************************************
************************************************
ok: [localhost]
```

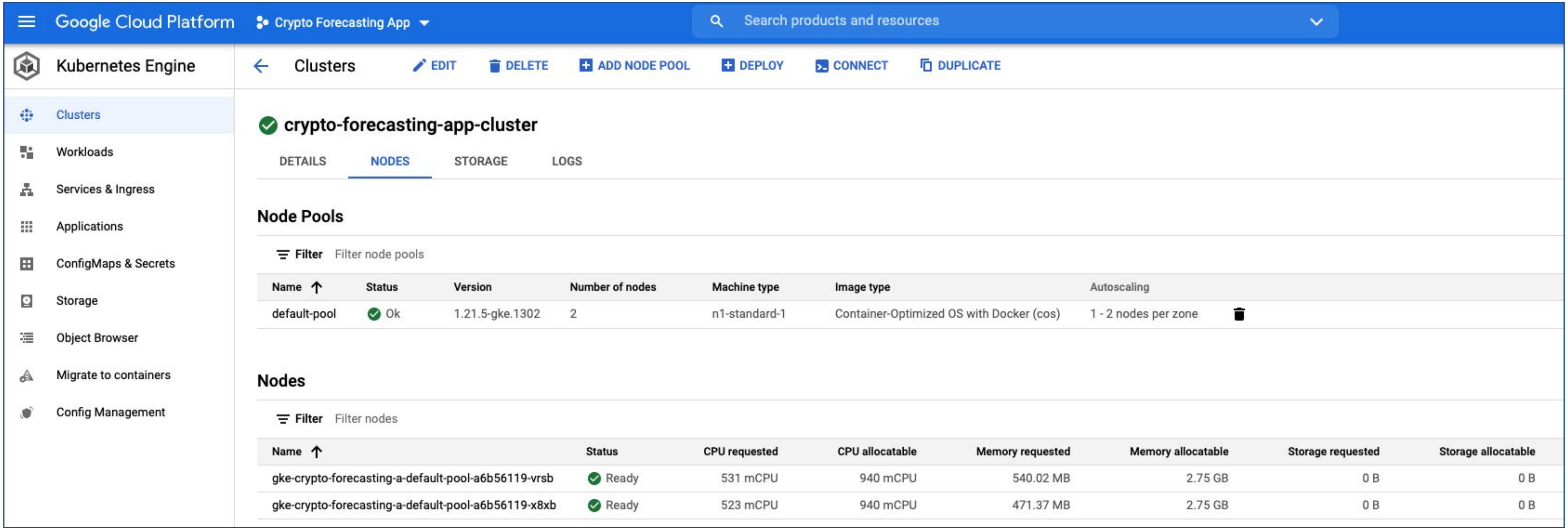
Kubernetes Deployment - Kubernetes Engine Homepage



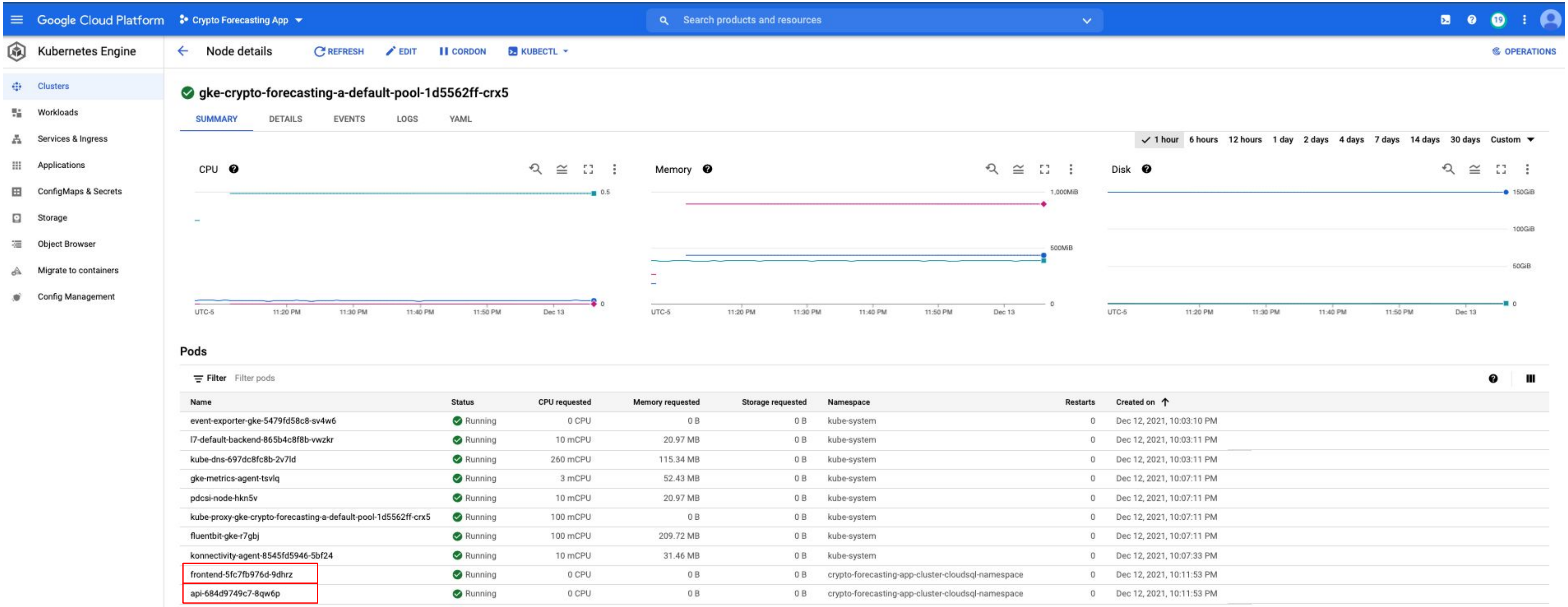
Kubernetes Deployment - Kubernetes Cluster



Kubernetes Deployment - Kubernetes Node Pool & Nodes



Kubernetes Deployment - Kubernetes Pod



Kubernetes Deployment - Kubernetes Crypto Forecasting App

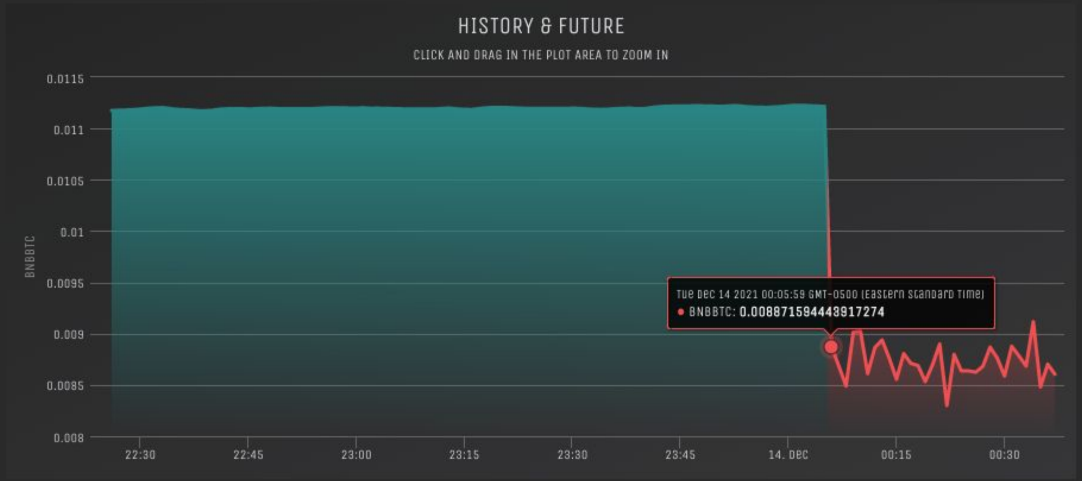
Not Secure 34.135.249.24.sslip.io

Crypto Forecasting

HOME SYMBOLS

- BTCUSDT
- ETHBTC
- BNBBTC
- BNBUSDT
- ETHUSDT

Visualization Plot



Live Market Data

Symbol	Best Bid	Best Bid(Volume)	Best Ask	Best Ask(Volume)
ETHBTC	0.080353	0.1083	0.080362	0.0461

Prediction Results

Time (Local Timezone)	Predicted Price
12:05:59 AM (+0 min)	0.008872
12:06:59 AM (+1 min)	0.008682
12:10:59 AM (+5 min)	0.008611
12:15:59 AM (+10 min)	0.008806
12:20:59 AM (+15 min)	0.008898
12:35:59 AM (+30 min)	0.008702

Next Steps

- Work on latency issues on the API side when processing raw data, loading the best model and performing predictions
- Build an automated modeling and training pipeline using VertexAI in order to automatically train new models when a user inputs a new symbol
- Improve current performances of the models
- Fix issues on the worker service side (not yet robust to all edge cases)
- Perform data tests
- Update the front-end to support the dynamic addition of symbols
- Improve UI by providing more relevant features for different use cases (e.g. log-in accounts, news information, new symbols to track and query, trading platform)
- Setup the backtesting and trading API (longer term)