Weather Predictor Architecture

System Components

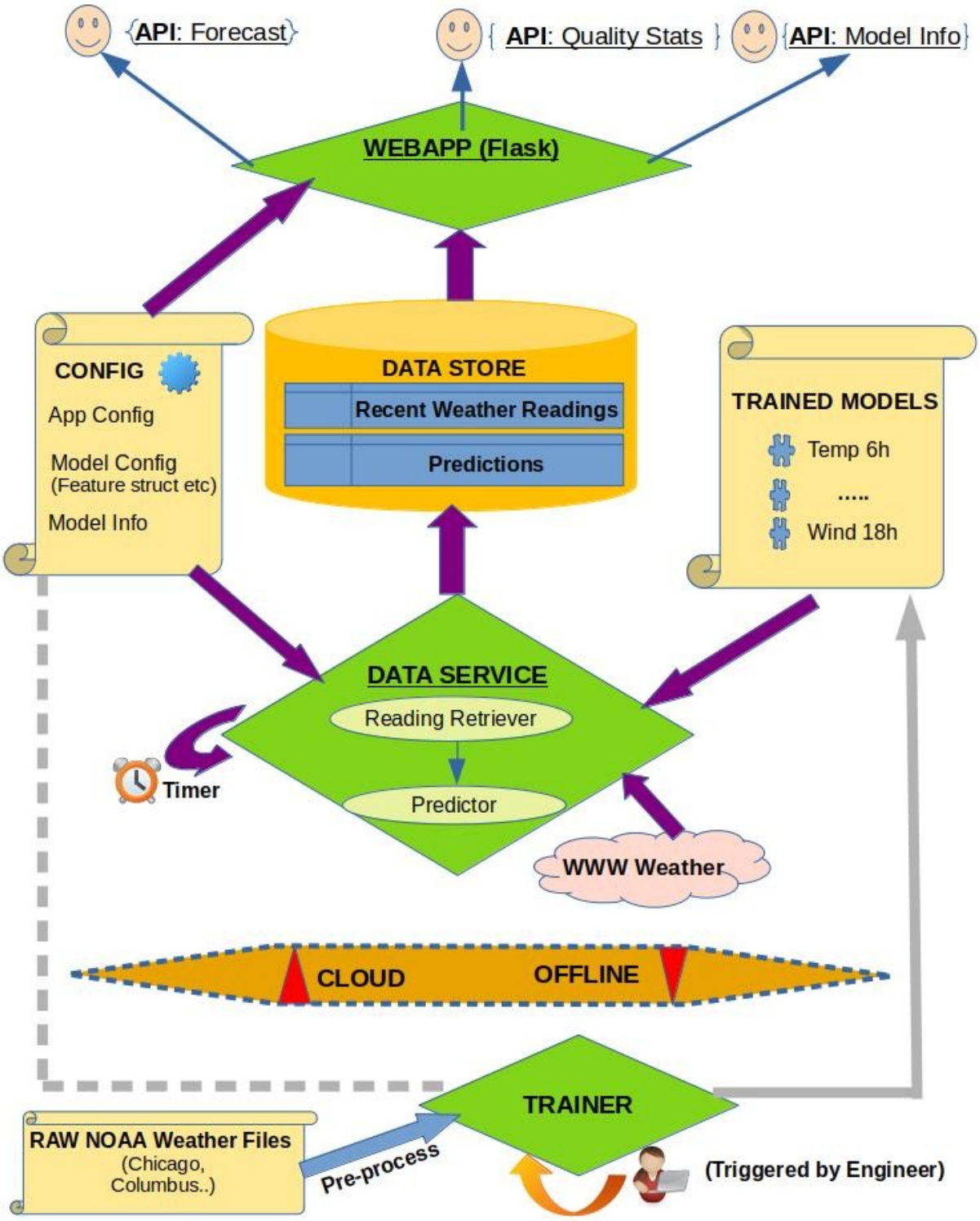
The diagram on the following page describes the modules of which Weather Predictor consists and their dependencies on each other. The following table briefly describes each component.

Note that part of the solution is deployed in the Cloud while another part of it runs offline in scripted mode and is triggered by an Engineer.

Component	Function	Implementation	Dependencies
Cloud - Deployed			
Webapp	Exposes public URLs through which the User can access forecast information and see it in HTML	Flask Python Application that renders data found in Data Store	Config (to display deployed Model Info)
Data Service	Populates the Data Store for Webbapp to use to generate User HTML.	Regularly scheduled Python Service. Runs on two schedules: (1) Hourly to get recent readings and to generate forecast for the next 24h. (2) As needed to backfill less recent readings and forecast based on them (in order to enable historic forecast quality audit)	Weather Forecasting Service on the Web (wunderground) to populate Recent Readings (Reading Retriever) Trained Models to feed those readings to generate Forecast (Predictor) Config to know how to process Recent Readings and feed them to Models; also to map locations to wunderground station codes.
Data Store	Source of all data displayed by Webapp. Has two data collections: recent weather readings from required locations and predictions based on those readings and Trained Models.	CSV files with designated names and in designated locations inside Container's file system	Data Service populates the Data Store
Trained Models	16 models (4 predicted variables X 4 lookahead intervals) used to generate forecasts	.h5 TensorFlow files residing inside Container's File System	Trainer must generate the models offline to be included in deployment.
Config	(1) information on features required for each model (so that Data Service can generate the right Tensors to feed to Models)	Human+Machine Readable file inside Container's File System	Trainer must generate model info for each model trained

	(2) Geographic location mapping for Weather Undergound (3) Model Info generated during Training to be displayed by Webapp		
Offline / Engineer-triggered			
Trainer	Trains Models to be used by Data Service for Forecasting	Python/TensorFlow script triggered by Engineer as part of Deployment Pipeline	NOAA Files that contain Training Data for associated geographic locations (must be processed by Preprocessor) Config containing all models and the definitions of what features and in what order are needed for each model to be trained
Preprocessor	Pre-processes esoterically formatted NOAA files into a form suitable for ML	Python Script triggered by Engineer as part of Deployment Pipeline	NOAA Files
Raw NOAA Weather Files	Manually ordered from NOAA CSV files containing weather records from a number of geographic locations that are proximate to the Target Location of Chicago	CSV files stored in project directory structure	A person needs to order from the NOAA website manually

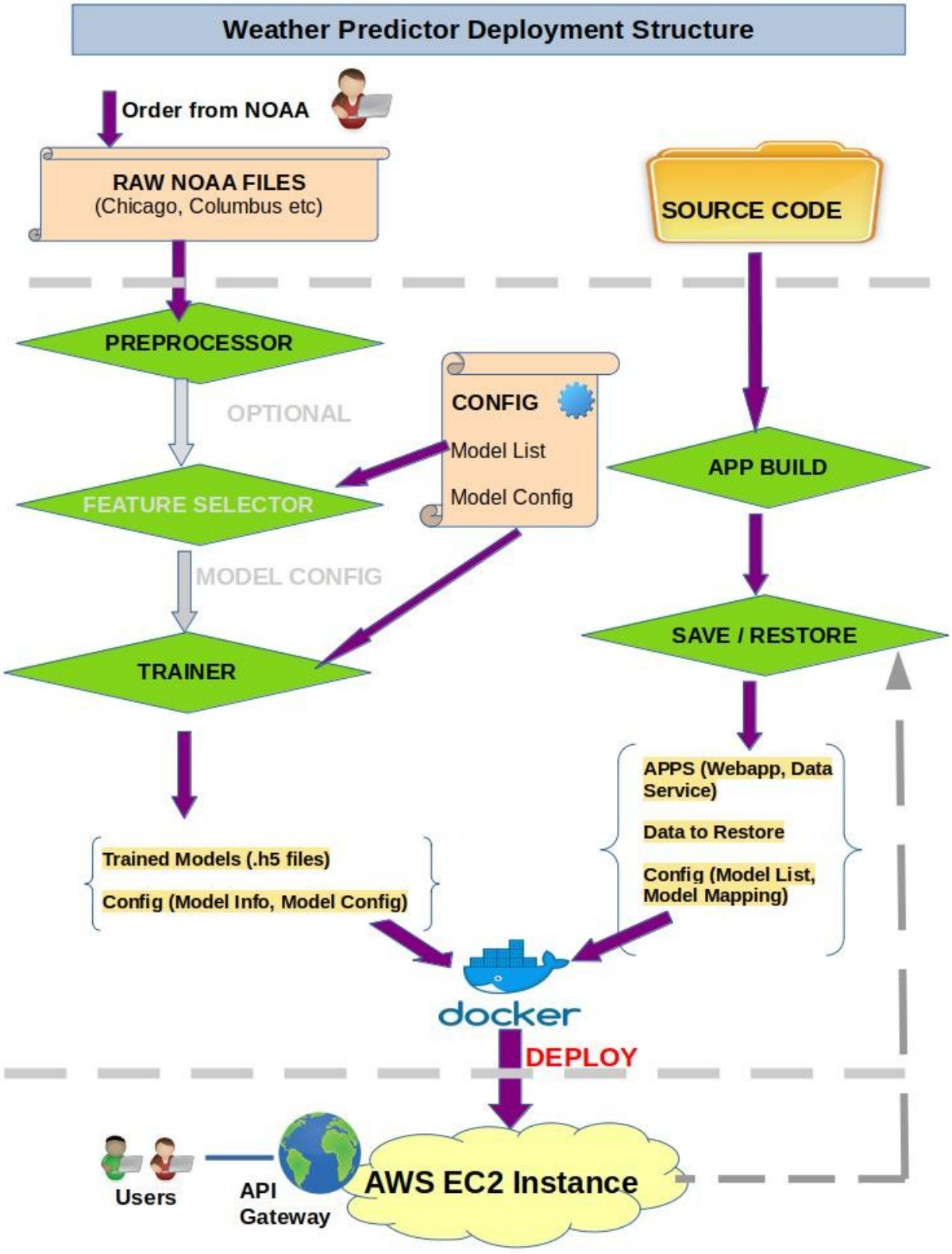
Weather Predictor Architecture



Deployment Structure

The diagram on the following page describes the deployment structure for Weather Predictor. Overall design notes on aspects not covered in the System Components section above:

- The application and the files it depends on shall be packaged into a single Docker Container and deployed on an AWS EC2 instance, fronted by AWS API Gateway to provide public User access
- (NOTE: such bundling is for launching quickly; we may choose to scale the application horizontally by re-implementing Data Store as a DBMS and separating Webapp and Data Service to run on different nodes)
- The pipeline consists of two sub pipelines: one for Model Training and the other for Code Building & Packaging; they are initiated/implemented as Python Scripts to execute on Engineer's machine.
- Model re-training is triggered manually by the Engineer when they believe that they can improve on already deployed models (such as training on more data or using other TF Model architectures)
- Once a new set of Models is deployed, we make no attempt at preserving the predictions of the past models; the User will only ever be exposed to the workings of the presently deployed Models
- Notice that in the Model Training sub-pipe there is the optional component Feature Selector. The idea is to use one-by-one feature selection approach to auto-generate the part of CONFIG that deals with Model Configuration. Basically, choose only those features that improve model performance.
- The Save/Restore component pulls data that takes a while for the deployed component to collect in order to make for a more seamless deployment experience. It then includes that data into the built Docker container to be immediately available on re-deploy. Specifically, this is useful for the Recent Readings data that is pulled from Weather Underground and takes a while to accumulate.
- Full list of components included into the Docker component to be deployed:
  - Webapp Flask Application
  - Data Service Python Service
  - Pre-trained Tensor Flow .h5 files
  - Application Configuration, including Model Info. Parts of it are auto-generated (Model Info and Model Features, if Feature Selector was used)


Weather Predictor Deployment Structure
Order from NOAA
RAW NOAA FILES
(Chicago, Columbus etc)
SOURCE CODE
PREPROCESSOR
OPTIONAL
CONFIG
Model List
Model Config
APP BUILD
FEATURE SELECTOR
MODEL CONFIG
SAVE / RESTORE
TRAINER
APPS (Webapp, Data Service)
Data to Restore
Config (Model List, Model Mapping)
Trained Models (.h5 files)
Config (Model Info, Model Config)
docker
DEPLOY
Users
API Gateway
AWS EC2 Instance