Data Science and Openshift on the MOC

Mentors – Red Hat Inc.

Michael Clifford mcliffor@redhat.com
Anand Sanmukhani asanmukh@redhat.com

Developers – Boston University

Yousif Khariddin ykh@bu.edu
Krishna Palle pallekc@bu.edu
Zhoufa(Marco) Chen zfchen@bu.edu
Nihar Dwivedi ndwivedi@bu.edu
Ojasvi Jhamb ojhamb@bu.edu
Fangya Xu fangya@bu.edu

GitHub
Prometheus Anomaly Detector

Abstract

Cloud computing is the fastest area of application within the information systems field. The adoption of cloud computing by businesses, government agencies, and academic institutions is on the rise. Cloud computing platforms could reduce cost, offer scale, and increase information systems responsiveness for those companies that are adopting it properly. For the open-source community of the cloud, from the users to infrastructure and application engineers, to monitor the health of the application they have deployed on the cloud, a tool called Prometheus Anomaly Detector is adopted in this Project and extended to improve the forecasting capability of the tool to improve the anomaly detection capability. This tool is an open-source contribution from Red Hat Inc. currently deployed on Mass Open Cloud (an open cloud exchange public cloud platform) which tracks a metric and predicts its future and flags anomalies when the actual value of the metric is off by a threshold. This project was about to improve the forecasting ability of the tool by adding new machine learning models.

1 Introduction

To reduce the cost and improve the efficiency, cloud computing, which can the delivery of on-demand computing service is becoming popular among clients. Instead of investing for a powerful datacenter, many companies use rent access to anything from applications to storage from a cloud service provider, such as Amazon cloud, Microsoft Azure, and Oracle cloud [1]. The benefit of using cloud computing services is avoiding the upfront cost and complexity of maintaining their own infrastructure.

In cloud computing, Platform as a Service (PaaS) is playing a more and important role in cloud computing and has been one of the most critical trends in developing applications[2]. Openshift, contributed by Redhat, is a PAAS built around Docker containers orchestrated and managed by Kubernetes on a foundation of Red Hat [3]. As more and more applications have been developed and maintained in the cloud environment, monitoring the health of their deployed application is vital, thus, we need a tool to monitor the status and operation of applications in the cloud environment. A tool would watch over different metrics of that application and alert them when the application behaves unexpectedly. In this project, our goal is to develop an efficient anomaly detector model and apply the model in Mass Open Cloud(MOC).

Prometheus is an open-source systems monitoring and alerting toolkit which scraps time series data from the metrics file of the application and builds a time series database. It implements high dimensional data models, equips the user with promQL, visualization tools, high storage and easy integration with extensive library support.

Prometheus Anomaly Detector(PAD), deployed on Openshift, is a framework to analyze time-series data generated by cloud infrastructure, this includes deploying a metric prediction model to detect anomalies in prometheus metrics. This kind of anomaly detector reads metric data of an application, understands the pattern, and predicts its future behavior. Machine learning models that can be used to learn data and predict data includes: Seasonal decomposition, moving average, ARIMA, SARIMA, Prophet, and LSTM, etc. [5]. Different models have different advantages and disadvantages. Our work is to study some of these machine learning models and apply these models in PAD. The tools that will be explored and used in this project include: DataHub, a blueprint for building an AI application as a service platform on the OpenShift Container Platform. Grafana is a tool that is used for data visualization. It also calls upon PAD to show anomalies. JupyterHub, a place where people can develop their algorithms. It runs Jupyter Notebooks.

The goal of our project is to understand the current architecture and implementation of PAD and improve its forecasting capability by integrating a new machine learning model and deploying the application on the MOC. In the following sections of this report, we discuss the

details of PAD in the second section and our contribution in extending the existing PAD by implementing LSTM model in the third section. In the fourth section, we include the results of running our LSTM model on MOC and compare the performance of our model to the existing models of PAD. We finish up this report by making a conclusion in the fifth section and followed by future work in the last section.

2 Prometheus Anomaly Detector

In the introduction, we saw what Prometheus Anomaly Detector does and why it was necessary, Figure 1 demonstrates briefly where Prometheus Anomaly Detector sits in the complete cloud architecture.

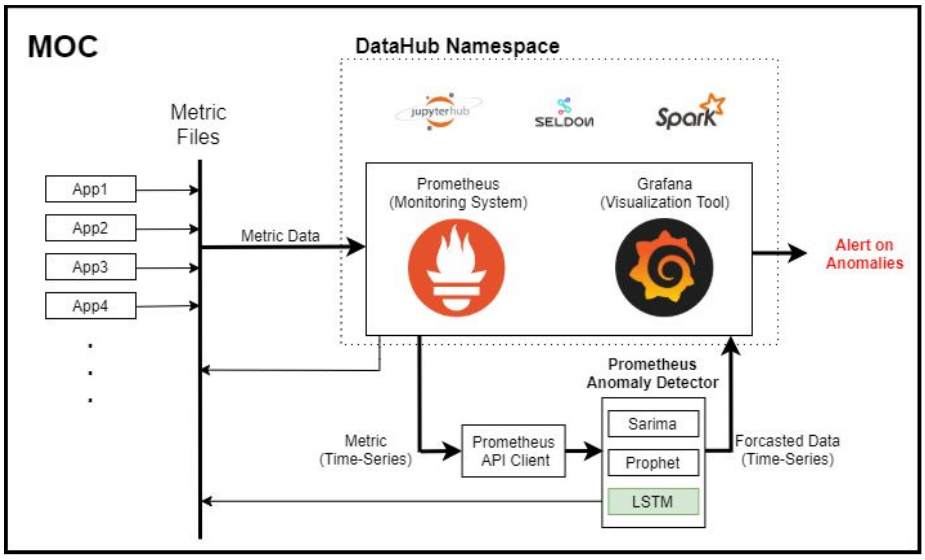


Figure 1 An schematic of the interaction between different components of PAD in the Mass Open Cloud platform.

Every application deployed on Mass Open Cloud emits metric files continuously per timestamp. Prometheus scrapes those files and saves the data for every metric that is present in the metric files per application. Prometheus anomaly detector talks to Prometheus via the prometheus API client, a communication interface that Prometheus provides, and collects all the data for a given metric. Once the metric data is collected, that data is used to train a Machine Learning model which is used to predict the values that the metric can take in the future and that prediction is passed on to Prometheus, along with the upper bound and lower bound threshold values for the future time steps.

Prometheus tracks the metric in real time and compares the real values with the predicted values that it received from Prometheus Anomaly detector and flags an anomaly if the real time value exceeds the mentioned thresholds for a timestep.

Diving further deep into the architecture of Prometheus Anomaly Detector, the following section talks about the flow of data in the PAD application and different components of PAD. To begin with, Prometheus Anomaly Detector can be broken down into 2 major components. The first component is the forecasting model, which is a machine learning model that trains on metric data and predicts the future values of the metric per timestep. The second component is the anomaly detection algorithm which calculates the upper and lower bound threshold values for the metric per timestep considering the predictions from the forecasting model.

To perform this forecasting, Prometheus Anomaly Detector was equipped with famous machine learning time series forecasting models like SARIMA and Prophet. Autoregressive Integrated moving average model, ARIMA is one of the most widely used forecasting methods for univariate time series data forecasting. ARIMA is a class of models that learns to predict the values of a time series data using its previous time step values. ARIMA consists of 3 important terms. The first term being the Auto Regressive term, which is the number of previous values in the time series data that we need to consider to predict the next value. The next term being the moving average term. This incorporates the number of forecast error terms that the model has to include. Final term being the difference term. This term is responsible for making the time series stationary.

This model, though with enough features would still fail to include the seasonality that some of the time series data would contain. To incorporate the seasonality of the data, SARIMA, an extension of ARIMA was introduced. SARIMA supports the direct modeling of the seasonal component of the series [6].

Though SARIMA proved to be a better time series forecasting model, another model from Facebook called FbPropher which is way more powerful than SARIMA was also incorporated into Prometheus Anomaly Detector. The Prophet uses a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation[7].

$$y(t) = g(t) + s(t) + h(t) + \varepsilon t$$

where,

$g(t)$: piecewise linear or logistic growth curve for modeling non-periodic changes in time series

$s(t)$: periodic changes (e.g. weekly/yearly seasonality)

$h(t)$: effects of holidays (user provided) with irregular schedules

εt : error term accounts for any unusual changes not accommodated by the model

Coming to the anomaly detection algorithm, once the forecasting model predicts the values for future time steps, to calculate the upper bound threshold, we take the predicted value and add 2 standard deviations of the forecasted values to it and similarly, the lower bound threshold is calculating by subtracting 2 standard deviations of the forecasted values to the predicted value. Any real time value, going above the upper bound threshold or going below the lower bound threshold value would be labelled as an anomaly.

To improve the existing forecasting models which are statistical machine learning, we believed that a deep neural network model would perform better and after performing some literature review we choose to implement a recurrent neural network model called Long Short Term Memory model (LSTM) to extend the forecasting capability of Prometheus Anomaly detector.

3 Our Contribution

Considering how a human would think while reading an essay, he/she would keep context of the situation in mind from the previous words he read and apply that knowledge to understand the current word. Similarly, to understand time series data, it is important to have some persistence of memory from the previous time steps. Traditional neural networks can't do this, hence recurrent neural networks were introduced where there would exist loops in the network which would help in information persistence. However the shortcomings of simple recurrent neural network models was that it would not perform well when the gap between the relevant information and the point where it is needed becomes very large. To address this issue, Long Short Term Memory(LSTM) model was introduced [8].

LSTM is a special kind of recurrent neural network model which is capable of understanding long term dependencies. The key to LSTMs in the cell state C_t, the horizontal line that runs through the top of Figure 2. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. A series of these LSTM cells are packed one after another which contain the state representation which learns the dependencies of the previous values in predicting the future values. In our application, since metrics range from simple to extremely complex time series data with values that range from 0.0001 to 50 thousand, we needed a robust model like LSTM to forecast the future.

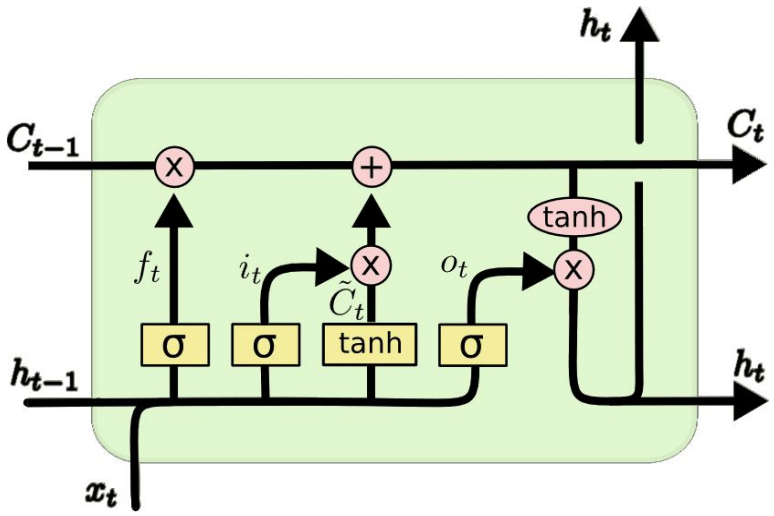


Figure 2 A Basic Unit of a common LSTM cell[9].

Now we prepare the data and run it through different architectures with LSTM models incorporated into them and tune hyperparameters to find the best model architecture that would fit in our scenario.

Data Preprocessing

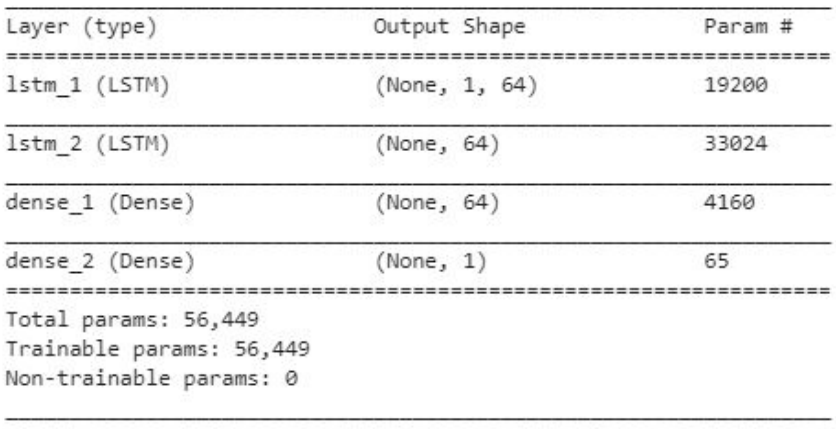
Prometheus Anomaly detector collects the metric data from prometheus and it trains the forecasting model with that data and predictS the future using the model. In the LSTM model that we implemented, the metric data is passed through a number of different steps in preparation for the model. First of all, the data is normalized between values 0 to 1. After testing with numerous metrics, this has proven to yield us better results. The idea behind this is that learning on normalized data is simpler. The raw data can have values that are extremely large or small, so the learned weights could diverge to infinity or converge to 0.

Next, the data is rolled according to the number of features chosen by the user (default is 10) and split up into features and labels. So, if the data is $[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$. And the number of features was 5. The data would then be reorganized to have features $X = [[0\ 1\ 2\ 3\ 4],\ [5\ 6\ 7\ 8\ 9]]$ and labels $Y = [5,\ 10]$. Note that in this example we are rolling unscaled data.

Finally, the data is readjusted such that the labels correspond to the change in value between the previous timestep and the new one as opposed to the actual values. So, in our dummy example above, the final $Y = [1,\ 1]$. After the data is passed through the model, the resultant predictions are finally rescaled to their original values and fed to the output

Model Architecture

After experimenting with different architectures and numerous metrics and testing their performance against a validation set for each, we decided to build the network with the below architecture



Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 1, 64)	19200
lstm_2 (LSTM)	(None, 64)	33024
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 1)	65

Total params: 56,449
Trainable params: 56,449
Non-trainable params: 0

Figure 3 The architecture of our model.

This network comprises 2 LSTM layers and 2 Dense Layers. The actual nodes within these layers, specifically layers lstm_2 and dense_1, are automatically tuned throughout our code. This ensures that the best version of this architecture is selected with respect to the input metric.

4 Results

In testing the performance of our model, we selected 2 different JupyterHub metrics of different complexities and fed their data through our model, as well as ARIMA and , the current machine learning models in PAD.

For the initial testing, we picked a real-time metric which is emitted by a JupyterHub application that is running on the MOC called ‘request duration in seconds’ and compared the performance of the existing models with our model. The results are as follows.

Metric 1: JupyterHub Request Duration in Seconds

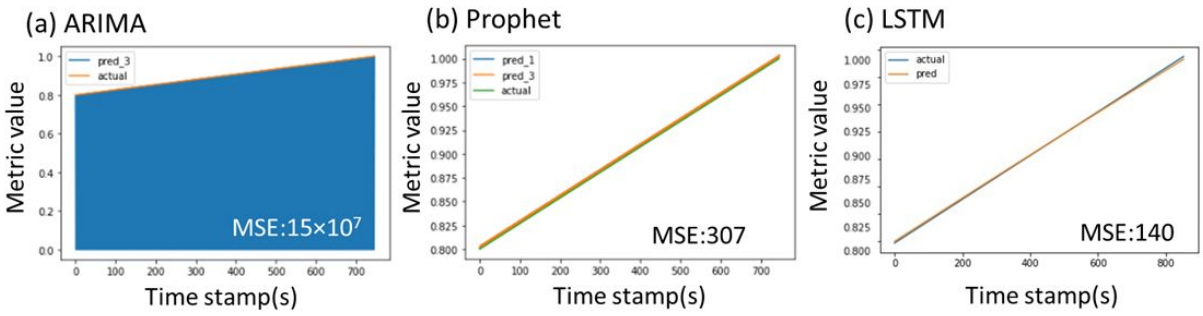


Figure 4 Comparison of performance between three different models: (a) ARIMA, (b) Prophet, and (c) LSTM. The number in each plot is the minimum square error(MSE).

This is a really simple metric. However, a significant number of metrics scraped by Prometheus look extremely similar to this, so understanding the performance of our model on this was important. From the above, it is clear that the LSTM model is performing extremely well with a mean squared error of 140, the least error of all models.

For our second test, we picked another metric from the same application called 'scraped duration in seconds' which is a little more complex than our first metric. It is also important to note that the bounds of this metric ranges between 0.015 to 0.035 unlike our previous metric which had values around 15000. The results are as follows.

Metric 2: JupyterHub Scraped Duration in Seconds

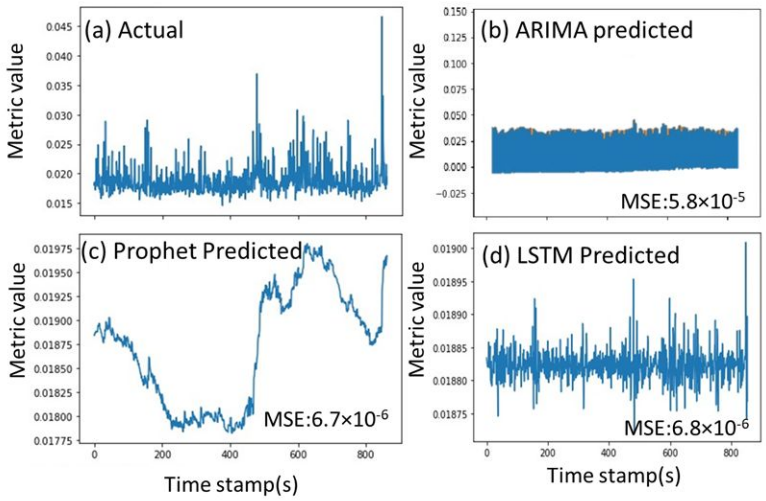


Figure 5 Comparison of performance between three different models: (a) The actual data. (b) The results predicted by the ARIMA model, (c) The results predicted by the Prophet model. (d) The results predicted by LSTM. The number in each plot is the minimum square error(MSE).

From the above, it is clear that none of the models were able to properly learn this metric. This metric was specifically chosen because of its complexity. This metric is seemingly random and extremely noisy, making it an extremely difficult metric. It is important to note that although LSTM has also failed at understanding this metric properly, it was able to somewhat understand the flow of the metric and "mark" its spikes. Technically, prophet does have a lower MSE here, but it is quite clear that prophet was not able to identify the metric's behavior whatsoever.

5 Deployment

Once the model was built, we then forked the PAD repository and wrote our model into the existing infrastructure. We then opened our Kubernetes instance and deployed our application/model and its image using the required YAML files.

The pod can be seen running below.

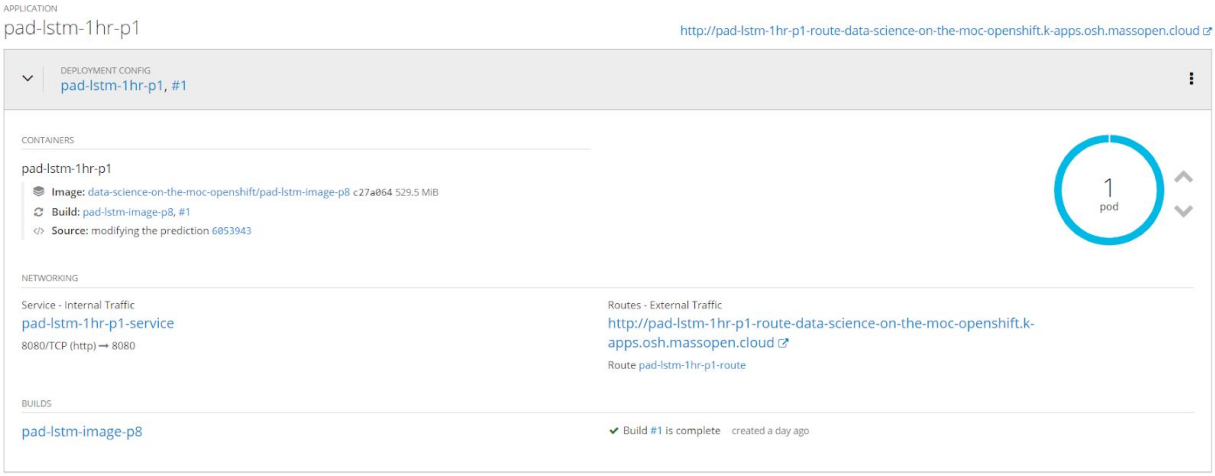


Figure 6 An image of the interface when running the pod.

Now that we've deployed our model, we can move to Grafana and visualize our model in work. This can be seen in the figure below. Here, the model has been running for 2 hours . It retrains every hour. The green line marks the metric we are monitoring. The orange line marks our predictions, and the other two lines are upper and lower bounds we have placed. Anything predicted beyond these bounds is marked as an anomaly. Since operation seems normal here, the second graph is completely flat, marking no anomalies. Another thing to observe is, though the graph of predictions seem to sway away from the actual, the change is value from the actual and predicted is 0.3%.

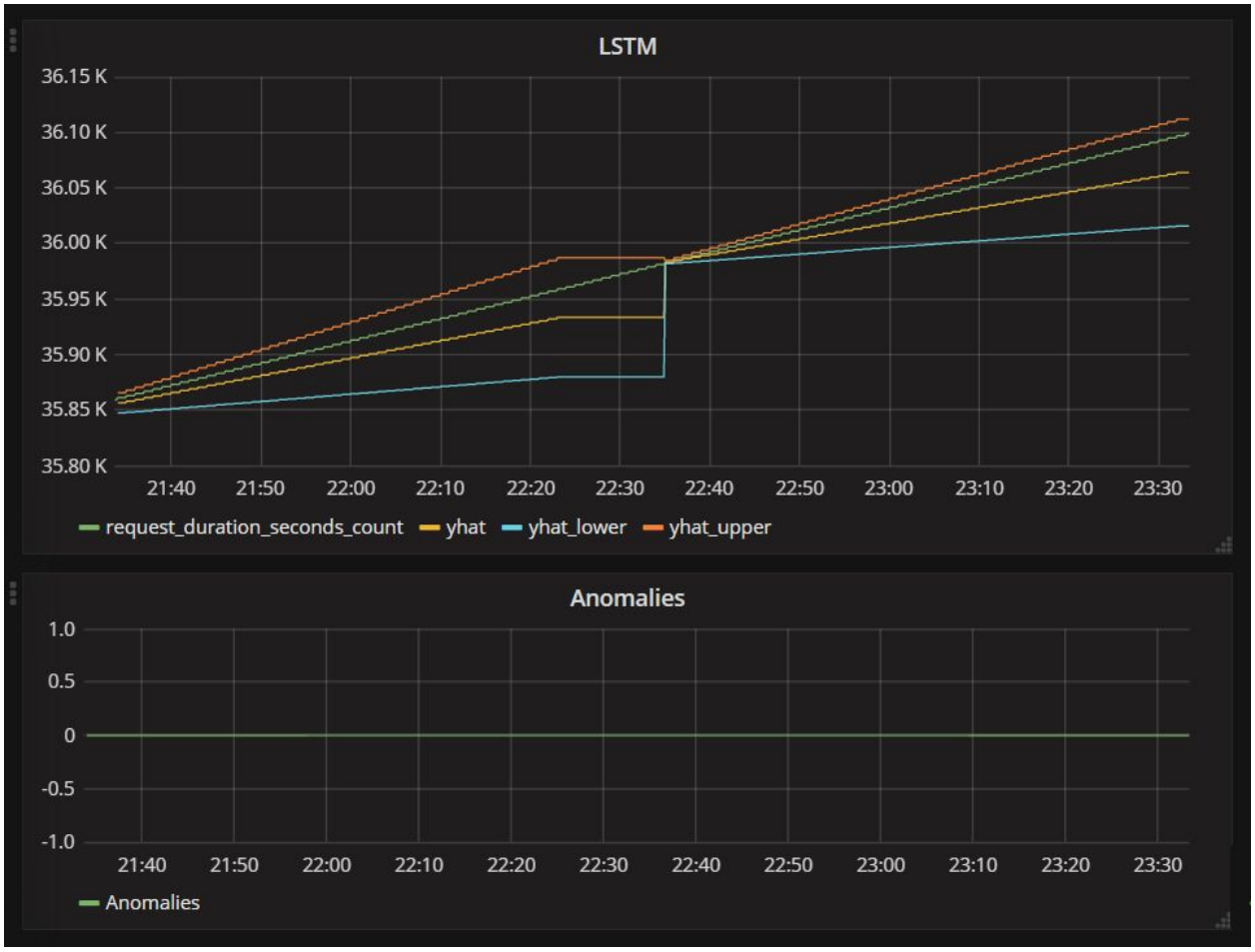


Figure 7 An image of the Grafana interface.

6 Conclusion

In this project, we successfully developed a machine learning model incorporating LSTM architecture for time series forecasting. We tested our model on a synthetic dataset (Numenta Anomaly Benchmark dataset) and implemented the model on real-world Prometheus data and improved the PAD forecasting performance. Next, we automated tuning our model's hyperparameters for a given metric to improve the overall performance. Furthermore, our model was encapsulated in a python file and integrated into PAD. At last, we deployed PAD on OpenShift with our model up and running properly.

7 Future work

More work can be done to further improve the performance of PAD. Firstly, we can implement additional time series forecasting models to strengthen the model-pool available for forecasting in PAD. Secondly, we can modify the existing anomaly detection algorithm to incorporate better thresholding of forecasted metric values. Thirdly, the model selection can be automated for a given metric by running all models in the available model-pool. Fourthly, PAD can be used as a service. At last, we can correlate multiple metrics to detect anomalies

References

[1]Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M., 2010. A view of cloud computing. Communications of the ACM, 53(4), pp.50-58.

[2] Giessmann, A. and Stanoevska-Slabeva, K., 2012. Business models of platform as a service (PaaS) providers: current state and future directions. JITTA: Journal of Information Technology Theory and Application, 13(4), p.31.

[3]Pousty, S. and Miller, K., 2014. Getting Started with OpenShift: A Guide for Impatient Beginners. " O'Reilly Media, Inc.".

[4] GitHub. 2020. Aicoe/Prometheus-Anomaly-Detector. [online] Available at: <https://github.com/AICoE/prometheus-anomaly-detector> [Accessed 3 May 2020].

[5]Medium. 2020. An Overview Of Time Series Forecasting Models. [online] Available at: <https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb> [Accessed 3 May 2020].

[6] Ghosh, B., Basu, B. and O'Mahony, M., 2007. Bayesian time-series model for short-term traffic flow forecasting. Journal of transportation engineering, 133(3), pp.180-189.

[7] Medium. 2020. A Quick Start Of Time Series Forecasting With A Practical Example Using FB Prophet. [online] Available at: <https://towardsdatascience.com/a-quick-start-of-time-series-forecasting-with-a-practical-example-using-fb-prophet-31c4447a2274> [Accessed 3 May 2020].

[8] 2020. [online] Available at: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/ https://www.citethisforme.com> [Accessed 3 May 2020].

[9] Salwerowicz, D.A., 2019. Design Proposal for a Software Tool for Speech Therapy. Modern Application Structure of Visual Speech Therapy App for Children (Master's thesis, UiT Norges arktiske universitet).