AI-based approach to enforce budget restrictions on Azure RG's for large enterprises

Harihara Vinayakaram Natarajan[1], Kunal Sunil Kasodekar[2]

Wipro Limited, Bangalore

[1]*harihara.natarajan@wipro.com,* [2]*kunal.kasodekar@wipro.com*

Abstract

Microsoft Azure is a cloud computing platform that integrates cloud services for development, deployment, testing, and application management [1]. An Azure Resource Group (RG) is a container that holds related resources (Instance of a service) required for a solution. This logical division provides a way for easy control, monitoring, and billing of the resources required for the solution [2].

Many large-scale organizations have a single azure billing account but a multitude of resource groups with many resources, making it difficult to monitor, audit, and limit expenditure. This inability to effectively manage RG's, results in groups within enterprises overshooting their pre- allocated budgets with no means of granular control. Natively azure does provide a REST-based budget API that informs the user when the RG crosses a specific threshold, although for the longest time it suffers from a bug wherein the response has a missing payload. The RG name is missing from the payload hence a brute-force search is required to check if the consumption is below the allocated limit. To tackle this issue we had implemented a naive mechanism that stores budgets in a database to query the actual RG using Azure's budget API. The RG's are queried periodically (once a day) and then stored in the database. However, this brute force method is computationally expensive and prohibitive (Especially in the case of a large number of RG's).

Hence, we propose a time-series-based prediction mechanism to enforce restrictions on Azure RG's for large enterprises by forecasting which RGs maybe near their budget limit and need to be checked/alerted/stopped, effectively replacing a brute force search. The forecasting models will be supplied with a feedback loop to ensure continuous learning and a FaaS architecture for scalable and modular serverless deployment. This prescient approach provides organizations with a proactive solution for better budget monitoring and consumption prediction.

Keywords: Granular control, Bruteforce, TimeSeries Forecasting, FaaS, Feedback loop

1. Introduction

The last few years have seen a massive surge in cloud-based enterprise applications. Cloud computing provides enterprises the ability to focus on solution design and workflow by off-loading infrastructure and deployment concerns to cloud-based services. This has resulted in a significant rise of FaaS and IaaS based cloud services. Microsoft Azure is a leading cloud computing platform preferred by such enterprises for their numerous solutions. To accommodate these many solutions, groups within the organization are stuck with a myriad of RG's and resources without fine-grained control. This results in budget overruns and ultimately revenue loss. To top it off as mentioned above Azures budget API has a long-standing bug that prevents effective budget control. To solve these issues we had created a naive solution as follows:

(a) An automated azure runbook stores budget details in a database during RG creation.
(b) Periodically another runbook uses brute-force search to query all the RG's in the database using Azure's budget API.
(c) The query returns the consumption percentages of all the RG's and those above the threshold are detected.
(d) For these detected RG's a notification mechanism is triggered and appropriate action is taken.

Enterprises are rapidly moving towards Low Code/No-Code platforms and the Cloud providers are providing the necessary building blocks to make this possible. Thus, in this scenario, it is very likely that bugs will exist in the building blocks and it may not be the highest priority for the provider (Azure in our case) to fix it. However, these are the features that will make/break the customer offerings provided by such organizations. For our organization (Wipro) the feature of stopping an experiment within the allocated budget was very crucial for our ProtoLab service offering and hence we explored the easiest way to build on top of existing services. However, as this naive approach is not scalable and time-consuming for large organizations we have proposed a forecasting based solution with continuous learning to predict when the RG's will be near their consumption threshold. We also realized that the model can be further used even if Azure fixes the bug to provide accurate predictions. This will help organizations to proactively monitor, manage and limit expenditure on a granular level. The prediction models will be deployed on Azure using FaaS.

2. Related Work

As per our knowledge, we were not able to find domain-specific (cloud-computing) work related to our current study. Most of the studies we found in other domains were related to budget forecasting rather than management and monitoring. None of them considered creating a production-level pipeline with a feedback loop to account for maintaining accuracy on real-life dynamic data. We reviewed this budget forecasting work in other

domains ranging from Highway Construction and Underground water management (water budget) to generalized budget forecasting mechanisms.

In 2018, M. Birylo et al. [3] used ARIMA as a Time-Series forecasting method to forecast water budget (underground water level) at three different locations in Poland. They were able to find out the key parameters affecting the future time-steps and thus their model was able to find a high correlation between actual and predicted values. However, the authors do not: include a feedback loop to account for future data-variance, provide solutions for effective budget management, find the best hyperparameters for the model, and experiment on other statistical/ML models.

In 2009, Wichan Pewdum et al. [4] trained an ANN-based forecasting model to predict the final budget and time duration for a highway construction project on two routes in Thailand during the initial construction stage. They found out the key factors that affect the budget and trained a neural net with these parameters as the input variables. This AI- based method provided an accurate model with insights to construction managers regarding budget overruns and delays enabling early corrective action. This work also suffers from the same problems as above as it lacks a multi-model pipeline-based approach along with a feedback loop. Also, the authors have not experimented on different forecasting models and treat it as a toy problem rather than a production one capable of handling real-world dynamic data.

Thus in our study, we are trying to bridge the gap between an experimental model and a real production scenario by incorporating a multi-model forecasting pipeline along with a feedback loop to prevent concept drift and maintain model accuracy across various organizational hierarchies. Also, unlike the other literature, we are handling the case of simple linear budgets and varying non-linear costs by using linear and dynamic models respectively to reduce operational costs.

3. Research Methodology

3.1. Synthetic Data Preparation:

Cost consumption datasets for azure with genuine time-series data are not publicly available. For experimentation, we require multiple resources with varied data points based on many different scenarios. As such data is not available publicly as well as within our organization we resorted to creating synthetic datasets. We have tried to create data points that mimic real-world scenarios. Such synthetic data will certainly affect the model accuracy but to counter this we have set-up a continuous learning-based framework to incorporate real data. This mechanism will certainly improve the accuracy after each learning cycle. For the data preparation stage, we have tried to model after some real azure resources. Based on our initial analysis most (Around 75%) of the resources have a fairly linear cost consumption trend. Hence for the first iteration of our solution, we have created a synthetic data generator that creates a data frame with 1 linear and 2 non- linear cost consumption resources. Thus, for prediction, we are considering an RG with one

linear resource and two non-linear resources. We have made the following considerations for our models and data:

Time Interval: The time sequence is generated for one month of data with an hourly frequency.

Prediction Mechanism: Data is trained on various models wherein the best model is selected for deployment. Once selected, the model periodically makes forecasts with a feedback loop for continuous learning and to check for inconsistencies in prediction.

3.1.1. Azure resource with Linear Datapoints

To create a resource with a linearly increasing accumulated cost trend an Azure Cosmos Db [5] with a provisioned output is considered. All the data points for the month have the same cost with some random cost additions due to increased storage. We create a linear dataframe for this Resource by considering a base level service with 100 read units/second (u/s) and 10GB storage with 1 table. Minute aberrations in the data are due to spikes in storage costs.

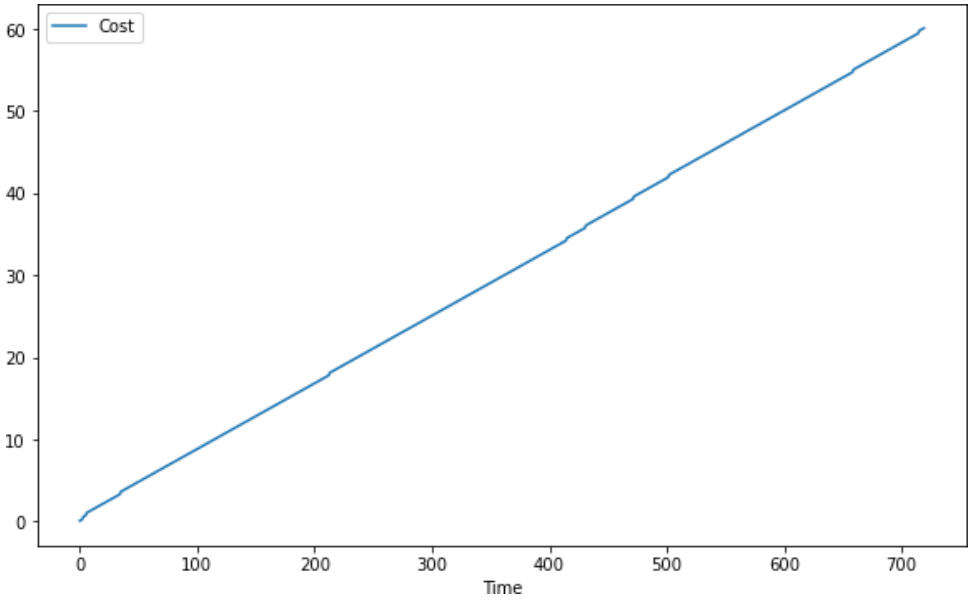

Figure 1: Graph for resource with linear data points

3.1.2. Azure resource with Non-Linear Datapoints

We create two non-linear data frames with different data distributions during data generation. The dataset for the Logic app has weekly seasonality with randomized noise added during each step of data production. The other dataset for the storage account has data- points with high variance during each time-step and no overall seasonality.

3.1.3. Accumulated Costs

For this iteration of analysis, we are looking to predict which resources accumulated cost crosses the threshold set by the administrator. However, we can easily predict the daily cost and then calculate the accumulated cost by using the same set of models/pipelines. While plotting the accumulated cost graphs and observing the datasets for the non-linear resources closely we found some interesting insights. It was clear that the accumulated costs will show an additive increasing trend. However, we expected some aberrations and seasonality. However even after decreasing and increasing the randomness/deviation of the data-points the graphs were somewhat linear with a very small number of aberrations and changepoints despite high variance among data points. This lack of non-linearity was surprising and we will investigate this further in the future. Despite the linear nature of the accumulated costs in this case the models and pipeline are created for highly irregular, seasonal, time-series data for a generalized prediction mechanism.

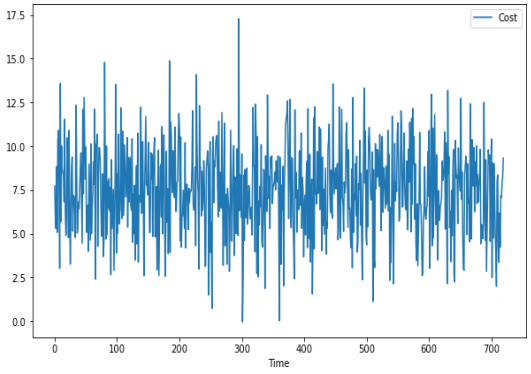

Figure 2: Daily costs for azure logic app

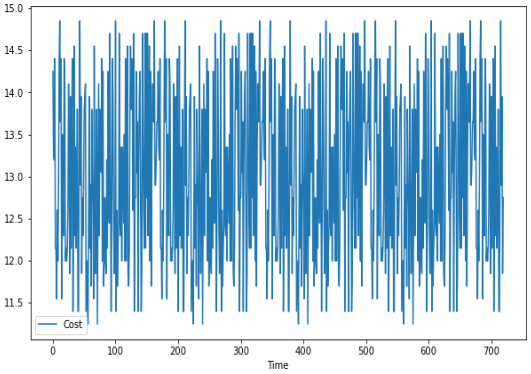

Figure 3: Daily costs for storage app

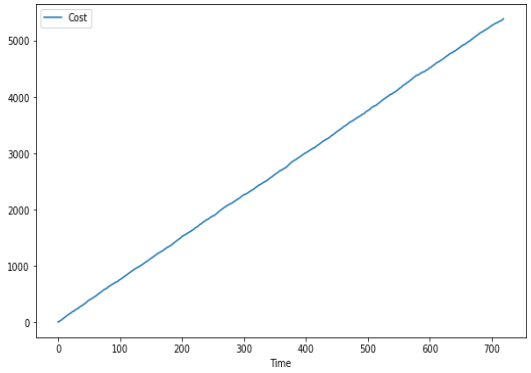

Figure 4: Accumulated costs for azure logic app

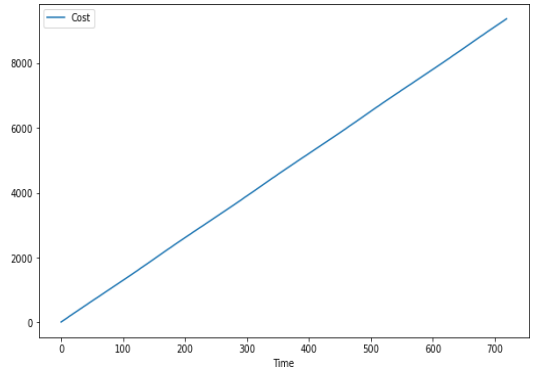

Figure 5: Accumulated costs for storage app

4. Solution Architecture

In this section, we explain our solution architecture and workflow followed:

4.1. Prediction Mechanism

We will segregate the resources in the RG based on the nature of the data points. Linear and non- linear data are separated to apply different sets of applicable Machine learning (ML) models to them. This was done because a large number of the RG's had costs that followed a linear trend. Hence applying irrelevant forecasting models on linear data is avoided. Once separated these data points will undergo transformations as required by each model before training. Then the models are trained, validated and the best model is selected for deployment using Azure ML Workspace. This workflow is carried out periodically whilst procuring historical time-series data for prediction. Data is ingested by the pipeline via an Azure Event Hub stream which will periodically procure the data points of the RG's.

4.2.Models used for Linear Pipeline

(a) Linear Regressor[6]

(b) MLP Regressor (MultiLayerPerceptron)[7]

(c) Random Forest Regressor [6]

4.3.Models used for Non-Linear Pipeline

(a) Auto-Arima [7]

(b) Exponential smoothing [7]

(c) Fb Prophet[10]

(d) Hierarchical time-series forecasting [7]

4.4.Solution Outline

The simplified outline for our current solution is a follows:

1. Initially, for generating our training data an automated azure runbook queries budget consumption of all RG's and stores them in CosmosDb. This will be our historical time-series data used for training our models.

2. We train the models in our pipeline using a published Azure ML workspace pipeline. Azure Event Hub, a streaming service streams data points for ingestion to our ML models using a REST endpoint with various triggers.

3. Once the models are trained they are registered in the Azure ML workspace and the inference configuration is setup. The entry script for the environment will ingest data and forecast future data points. These results are returned as JSON responses to CosmosDb

4. An azure runbook compares the top "X" predictions with the ground truth (actual RG con- sumption) acting as a feedback loop. In case of a high RMSE error, the models are retrained and redeployed for the next cycle.

4.5. Feedback Loop

Any machine learning model created to emulate real-world scenarios will suffer from decreasing accuracy over a period of time. This is because the relationship between the independent features and dependent target variable may change over some time or some hidden dependencies may become dominant. This issue is predominant in deployed models with no continuous learning and is called concept drift. Thus we implement a simple feedback loop to counter this issue. Currently, we don't have a full-fledged feedback loop. During each cycle of prediction, we will compare the RG's presumed to have crossed the budget with the ground truth. Based on the judgment error we will re-train models with a high error percentage on new historical data.

5. Results and Discussion

For the prediction mechanism, a dictionary of linear and non-linear resources of the RG is created. The linear and non-linear groups are subjected to a different set of estimators and transformations. The Pipelines are as follows:

5.1. Linear Pipeline

Initially, the data is preprocessed to include the previous time-lag and lag-difference to increase the number of features for better prediction. Now a set of Scikit-learn pipelines [8] are created with estimators known to perform well on linear data. The estimators used are a linear regression model, Random forest, and Neural Network. The Pipeline iteratively trains the data points on these models and then models pertaining to each of these pipelines are evaluated to find the model with the lowest RMSE. This model will be the model that will be deployed for prediction. The linear regressor and neural network are highly accurate and give a very low RMSE error of 0.0291 and 0.0296 on our validation dataset. Surprisingly the random forest regressor gives a constant (wrong) output during prediction. We concluded this was because a random forest is an ensemble of decision trees that answers/predicts based on the decision of each tree. These decision trees are created based on training data, hence they cannot predict/regress on new/unseen data and therefore all the trees give a wrong answer which is averaged out for each prediction. Once all the predictions are completed the costs of all the resources belonging to the RG are summed up to calculate the total RG consumption.

5.2. Non-Linear Pipeline

We have two sets of competing pipelines running simultaneously and the pipeline having the model with better accuracy during deployment will be considered for the next cycle of prediction. These sets of competing pipelines use hierarchical time-series forecasting on one hand and staple forecasting models on the other. Theoretically, the hierarchical time-series model is better suited for our current data due to the inherent grouping of resources and RG's. However, we have two competing pipelines because we want to test pipeline performance on real data in production and accordingly choose the model. Initially, the data is preprocessed to create a univariate time series sequence. Then the pipelines work as

follows:

5.2.1. Non-Linear Pipeline A

(a) Auto-Arima

A pmd.arima pipeline is created with a BoxCox transformer[12] and an Auto-Arima model. The box cox transformer converts the data to a normal distribution. If not used the model accuracy is drastically affected. The Auto-Arima model is fitted and the best combination of p,d,q is picked by the model itself. The model finds the best values of p,d,q as 0,1,2 and 1,2,2 for the logic app and storage account respectively. Seasonality if any is set based on week-wise data. The advantage of using a pipeline is that the encoding and decoding for the transformations are done by the pipeline itself, streamlining the whole process. Also, model fitting, prediction, and evaluation are simplified. The Auto-Arima model gives an RMSE value of 210.2071 for the azure logic app and 393.2108 for the storage account.

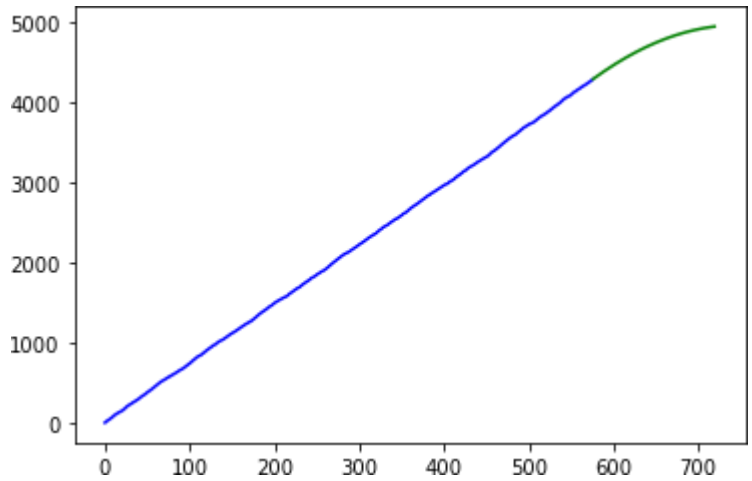

Figure 6: Auto-Arima Prediction for logic app (Green line implies Predicted values)

(b) Exponential Smoothing (ETS)

A double exponential smoothing model with an additive trend is trained to compensate for the trend of increasing accumulated costs. Due to lack of seasonality in the data, a triple/Holts-Winter model is avoided but a parameter grid search can be implemented in the future to automate parameter selection for this smoothing. Arima and ETS are similar because both use autoregression but, ETS reduces the weights of the time lags exponentially. The ETS model gives an RMSE value of 36.2866 for the azure logic app and 3.0673 for the storage account.

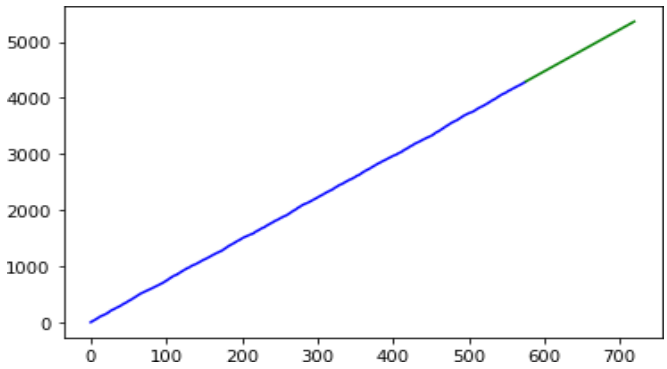

Figure 7: ETS Prediction for logic app(Green line implies Predicted values)

(c) FbProphet (FBP)

FBProphet is a forecasting algorithm that uses a decomposable additive model that fits several linear and non-linear components. It has the capability to consider trends and holidays whilst deducing daily/weekly/custom seasonality. The data is initially normalized then the FbProphet model with default parameters is fitted on our data. Various observations relating to trends and seasonality are made on our data. The model gives an RMSE value of 0.4190 for the azure logic app and 0.1309 for the storage account. Thus as compared to Arima and ETS, FBP gives the best accuracy due to its very low RMSE value. As the default model fits several linear models for the time-series data and our datapoints are have a linear nature the model is highly accurate.

The RMSE results of each resource are stored in a dictionary. So finally the best model for each set of the Non-Linear RG is selected for further use. Once all the predictions are completed the costs of all the resources belonging to the RG are summed up to calculate the total RG consumption. Scikit Learn pipelines require estimators/models to belong to the same library hence best to our knowledge it is not possible to create pipelines using models from different libraries. Thus creating a common unified pipeline will be a future goal.

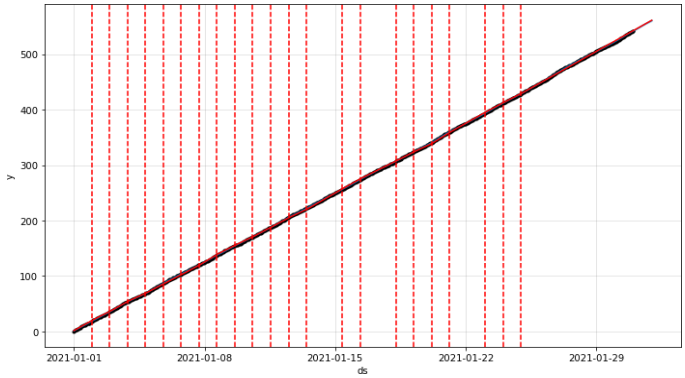

Figure 8: FBProphet Prediction for logic app (The red-lines indicate changepoints)

5.2.2 Non-Linear Pipeline B

In this workflow, a Hierarchical time series forecasting (HTS) is used. HTS is used when a time series can be naturally disintegrated into individual time series components. This is true in our case where we can disintegrate accumulated costs for a Resource Group into individual time- series components pertaining to the resources. This hierarchy can also be further utilized for grouping various RG's and then finally onto the subscription. A hierarchy of these time-series components is created based on which a summing matrix is created which maintains the equalities among the nodes. The condition maintained by the summing matrix is that the total costs at each hierarchial node (parent node) are the sum of the individual time-series (child node). Then a time- series model of choice (Arima/ETS etc) is fitted to each of these time sequences based on various conditions. We use HTS to create a time-series prediction model for our Resource group and resources individually instead of summing the individual time series manually. This provides for a faster, granular level of data prediction mechanism. Two methods of HTS are currently used:

(a) Bottom Up: An Auto-Arima model is fitted for all the children nodes (Resources). The data is then integrated up to the root node i.e the RG Costs.

(b) Top Down: An Auto-Arima model is fitted for the root node (RG costs) only. The data is then disintegrated to individual children nodes i.e the resources. For the diagram below A = Accumulated cost for the RG, CL = Linear Resource, and CN = Non-Linear Resource. The metrics and forecasts for these hierarchical model are as follows:

This way we create a time-series forecasting model for all the nodes whilst maintaining the hierarchy by either using a top-down or bottom-up approach. When using HTS we don't require

	Total	A	CL1	CN1	CN2
ME	32.3865232	32.3865232	0.08463164	31.9382229	0.363668666
RMSE	36.5677812	36.5677812	0.11313565	36.0703569	3.062743310
MAE	32.4820238	32.4820238	0.09279589	31.9982478	2.652005417
MAPE	0.2376405	0.2376405	0.17182987	0.6428234	0.030958874
MPE	0.2368384	0.2368384	0.15532970	0.6414262	0.005498103
MASE	1.5744794	1.5744794	1.11161748	4.2955802	0.202478125

Figure 9: Metrics (Bottom-up)

	Total	A	CL1	CN1	CN2
ME	32.3303225	32.3303225	-0.3570672	44.1093414	-11.4219517
RMSE	36.5157332	36.5157332	0.3653314	47.4275018	11.9691784
MAE	32.4270386	32.4270386	0.3570672	44.1093414	11.4219517
MAPE	0.2372336	0.2372336	0.6593742	0.8917329	0.1331644
MPE	0.2364213	0.2364213	-0.6593742	0.8917329	-0.1331644
MASE	1.5718142	1.5718142	4.2773680	5.9214247	0.8720553

Figure 10: Metrics (Top-down)

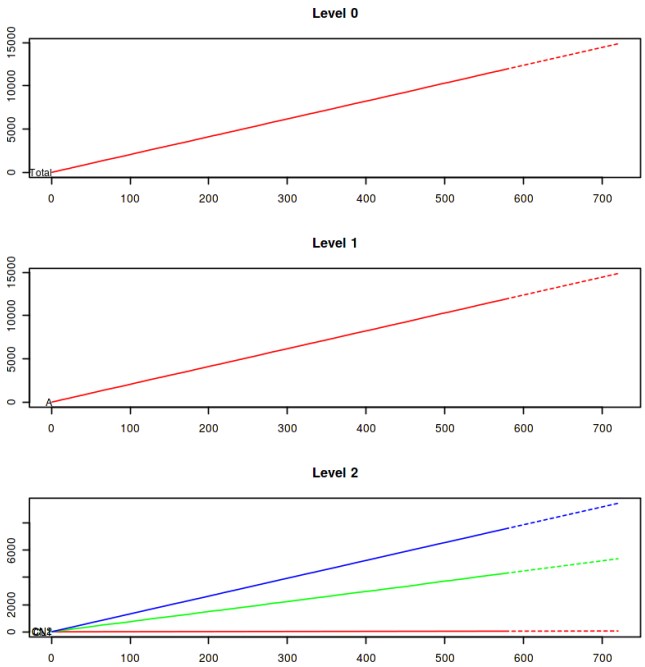

Figure 11: Forecast (Bottom-up)

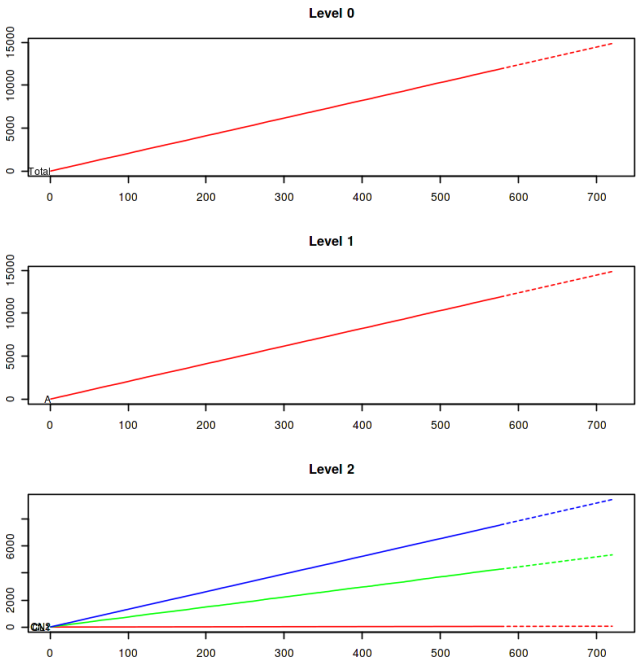

Figure 12: Forecast (Top-down)

to calculate the total cost's for the RG as the parent node has a time-series model for these accumulated costs.

5.3. Challenges

One challenge we are currently facing is managing RG's with multiple budget thresholds. Many RG's have actions set up for various thresholds (like 65%, 75%, 90%, etc.) rather than a singular restriction during any given cycle of forecasting. While creating the training dataset we can include a parameter for percentage budget consumption w.r.t to the initial threshold. RG's with a high percentage of budget consumption can be given preference for prediction. We can train an ML model to predict the cycle period for our forecasting mechanism. It will take into account features like RG budget consumption, feedback regarding missed RG prediction, and output the optimum cycle period. Also, we need a mechanism to prevent resources with persistent storage for code/important data to be removed from consideration for budget management/actions. Thus we are only looking to deprovision the compute. To prevent this, we are looking to flag these types of RG's during the RG creation stage so the runbooks will not fetch data from them. Also, these flagged entities will be ignored by our prediction pipelines. To reduce computation overhead we consider only the top "X" forecasted RG's for budget management. During each cycle, scripts will run to predict RG's that would have crossed the budget limit or are near it. Then the top results (based on accuracy) will be compared with the actual consumption (ground truth) of the RG's and accordingly further actions and feedback loops will be applied. However, what if RG's not belonging to these top results cross the threshold? Thus we need

to devise a mechanism to tackle this problem. One solution we propose is to predict the "X" number of RG's using an ML model that takes parameters such as accuracy, current consumption, prediction cycle length, etc into consideration. This way we can make sure all the RG's near the budget threshold are predicted. Another option can be to define an event listener that listens to the notifications after the predictions and finds out RG's that were missed. This data can be used as an input parameter for a feedback loop that dynamically changes the period of the prediction mechanism and encompasses the maximum number of RG's with budget overruns. Our final goal is to deploy our model in pro- duction, so rather than focusing on fast training with static data points (a methodology followed in ML research) we are looking to devise a workflow with dynamic data and fast inference with low latency. Our main focus is not creating a state of the art model but satisfying different objectives imposed by various stakeholders.

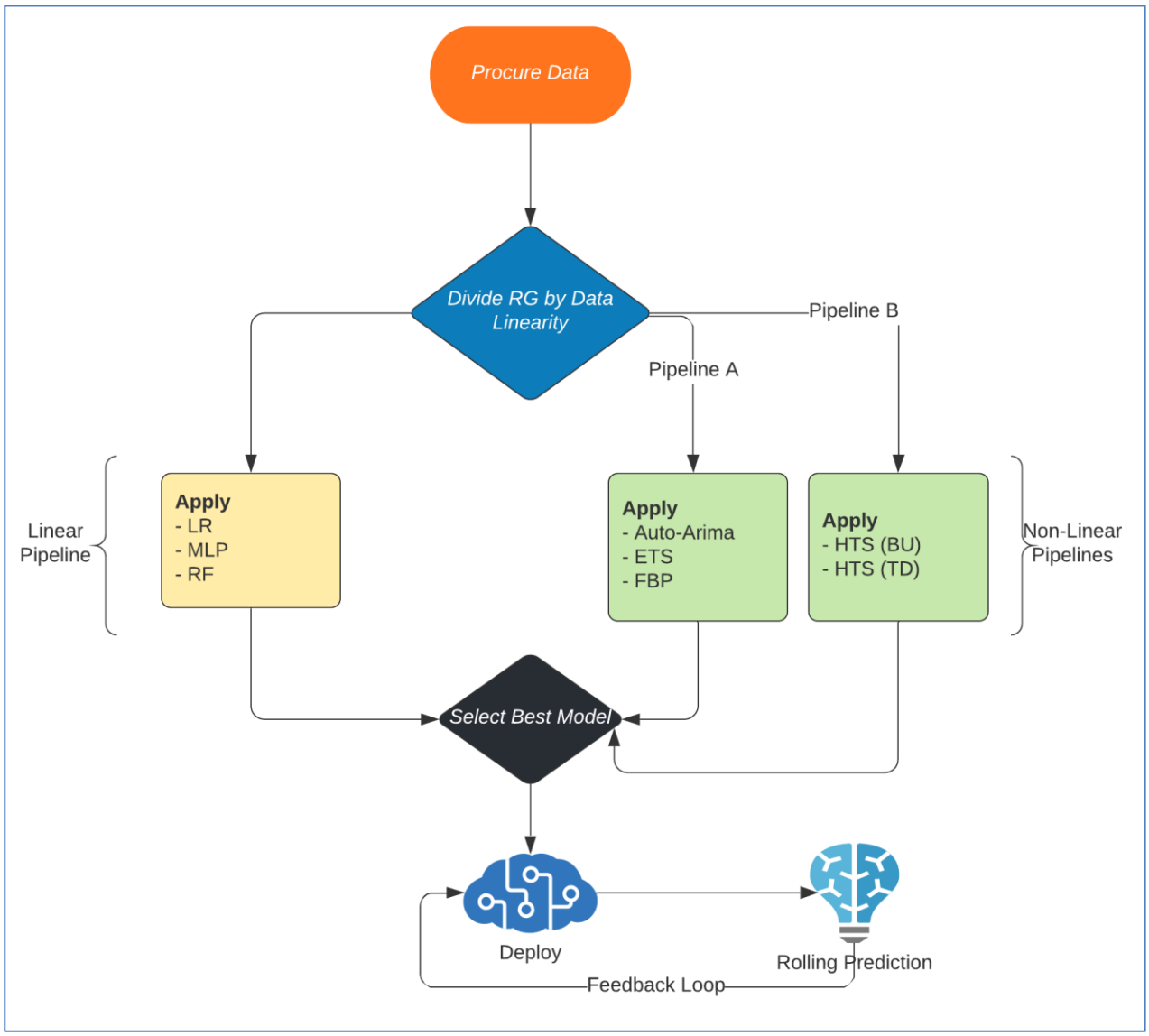

Figure 13: Model Workflow

Currently, we are using a simple feedback loop but we are looking to improve it in the future along with testing the solution on a large number of RG's with varied data points. We are also looking to use better deep-learning models (like LSTM) for more accurate time-series prediction and incorporate it by using an improved end-to-end pipeline.

6. Conclusion

Large scale enterprises are rapidly moving towards a cloud-based architecture due to its lucrative offerings like Low Code/No-code services. Microsoft Azure is at the forefront of this revolution, however, it lacks fine-grained resource control and suffers from a long-standing bug that prevents effective budget control resulting in budget overruns and revenue losses. It is imperative to develop a solution to solve these issues for such large scale organizations. To tackle these problems we have proposed a time-series-based forecasting solution that predicts the top "X" number of RG's that will cross the budget threshold. We have implemented a competitive multi-pipeline-based approach to select the best models pertaining to real data in production. We have also used a feed-back loop to ensure continuous learning to reduce model error over a period of time. We are using Faas architecture for a scalable, serverless, and modular approach. Thus our solution provides a proactive approach for better budget monitoring and consumption prediction using a multi-pipeline-based prediction mechanism.

7. Future Roadmap

In the next iteration of development we are looking to resolve the current issues plaguing the system along with:

1. Designing a fully functional end-to-end pipeline for prediction with an im- proved feedback loop
2. Calculating time saved by using our prediction model over brute force based search
3. Observing the effect of increasing the number of instances along with tweaking other system parameters.
4. Setting a limit on the number of instances for prediction along with budget management
5. Using deep learning-based models for time-series prediction
6. Testing the solution on a large scale with inherent feedback-loops and different types of synthetic datasets.

References

[1] Get started guide for Azure developers. (2019). Retrieved from https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide

[2] Manage Azure Resource Manager resource groups by using the Azure portal. (2019). Retrieved from https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/ manage-resource-groups-portal

[3] M. Birylo, Z. Rzepecka, J. Kuczynska-Siehien, J. Nastula; Analysis of water budget prediction accuracy using ARIMA models. Water Supply 1 June 2018; 18 (3): 819830. doi: https://doi.org/10.2166/ws.2017.156

[4] Pewdum, W., Rujirayanyong, T. and Sooksatra, V. (2009). "Forecasting final budget and duration of highway construction projects", Engineering, Construction and Architectural Management, Vol. 16 No. 6, pp. 544-557. https://doi.org/10.1108/09699980911002566

[5] Pricing calculator. (2021). Retrieved from https://azure.microsoft.com/en-in/pricing/calculator/

[6] Sklearn LinearRegression. (2021). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear model.LinearRegression.html

[7] Neural network models (supervised). (2020). Retrieved from https://scikit-learn.org/stable/ modules/neural networks supervised.html

[8] Ensemble methods. (2020). Retrieved from https://scikit-learn.org/stable/modules/ensemble. html

[9] Hyndman, R.J., & Athanasopoulos, G. (2018). Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2

[10] Quick Start. (2021). Retrieved from https://facebook.github.io/prophet/docs/quick start.html

[11] sklearn.pipeline.Pipeline. (2020). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[12] Stephanie Glen. (2015). Box Cox Transformation. Retrieved from https://www.statisticshowto.com/box-cox-transformation/