National College of Ireland

Decision Making Framework for Purchasing A Cloud Instance

MSc Research Project
Cloud Computing

Piyush Narkhede
Student ID: 17151538

School of Computing
National College of Ireland

Supervisor: Dr. Sachin Sharma

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Piyush Narkhede
Student ID:	17151538
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Dr. Sachin Sharma
Submission Due Date:	20/12/2018
Project Title:	Decision Making Framework for Purchasing A Cloud Instance
Word Count:	5907
Page Count:	28

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	27th January 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Decision Making Framework for Purchasing A Cloud Instance

Piyush Narkhede
17151538

Abstract

Buying an instance on the cloud is a decision taken by user. Those decisions are based on reviews, features and service of the cloud. Before investing in cloud instance, user should know about some factors of that instance like utilization rate, transmission speed and many more. Perfect decision always based on an exact proof, that proof is provided by our research which gives an idea about various instances running on different regions. We are also predicting bid price for Amazon spot instance. Providing an assurance to user whether instance will be reliable and the network transmission speed in the region will be faster than other regions by using real-time values. We are using ARIMA machine learning time series model to predict bid price. Amazon CloudWatch is used to monitor the instances and collecting real-time values for our model. Also, recommendations are provided which helps user to select perfect instance from different regions. As per our implementation, we successfully provided decisions based on different time and region of an instance. In future, we are expecting to add real-time bidding price of the spot instances. Also, comparison with various cloud service providers will be additional benefit for the user.

Contents

1 Introduction **2**

2 Related Work **3**

3 Methodology **4**
3.1 Project Components 4
3.2 Algorithms 7
3.3 Sequence Diagram 7

4 Implementation **8**

5 Evaluation **12**
5.1 Recommendations at 9AM: 13
5.2 Recommendations at 1PM: 14
5.3 Discussion 15

6 Conclusion and Future Work **15**

Appendices **18**

A Configuring AWS Instance: **18**

B ARIMA Model **25**

C Application UI **25**

1 Introduction

Cloud Computing is continuously growing technology for delivering computing services such as storage, memory, server, etc. While cloud service providers (CSP) are taking care of the servers so that their services should be reliable all the time. Also, they are responsible for delivering flexible resources with moderate cost to a user over the cloud. A user can compare various CSPs and make decision which is perfect for their company. For this decision, they can follow reviews of previous users as well as cost and features of the instances so that they plan a budget over a good quality instance. For migrating from traditional way to modern IT world, industries started using cloud as an option for growing their business. It is a successful business strategy to use cloud as an online IT resource for running industry within low budget, secure and reliable Marston et al. (2011).

For using cloud, there are many CSPs available in market such as Amazon Web Services (AWS), Microsoft Azure, SAP, Google Cloud, IBM, Oracle etc. All of these providers are providing reliability, security, fault tolerance and many more features Wang et al. (2010). But all those features user can use and then decide whether these are working properly or not. But what happens if they get any framework which will give idea about features and cost before using them? User can see the exact working and cost of an instance in advance and decide whether purchase that instance or not. So, that user can fix their budget as per frameworks prediction as well as they will get assurance for QOS provided by CSP.

By taking this motivation, our research topic is based on a prediction of cost and performance of an instance of cloud. We are using AWS for our research because as per Dignan (2018), 68% users using AWS as a cloud platform for their companies. AWS is using bidding strategy to sell their instance. It is better to reserve the instance before time or in advance. Spot instance can allocate by on demand and advanced manner. On-demand service cost more for an instance than advance booking. The customer has to bid on specific instance so that they can buy an instance in advance.

As we can see, in stock market, a user used to see predictions for future investments. Same for electricity bills and housing rates, the user seeing the predicted values and then, plan for saving in the future. For spot instances, user want to bid over that instance, but they has to decide that price, which they should start for bidding. So, instead of depending on time bidding, user can use our framework to see expected price of a spot instance. Also, some features like showing availability and bandwidth will be helpful to buy an instance whether in same or different zone. That analysis is showing in terms of graphs so that you can easily take decisions over spot instances. Motive of this framework is to save the budget for companies and providing exact recommendations for an instance.

My research question is: "Can the framework's predictions enhance users' decisions to buy cloud instance by checking cost and network data transmission speed of an instance?".

It is deeply explaining the framework or system which is proposing for recommendations. This proposed system will enhance the users' decisions for purchasing a cloud instance.

For predicting spot price, we are using ARIMA time series model and one of the best service provided by Amazon CloudWatch[1], using for monitoring network logs and CPU performance in the cloud. We are predicting cost for EC2 instance but whether that instance is really reliable? Can we use all memory allocated by cloud service provider (CSP) for that instance? Which instance has faster transmission speed whether from same region or different? For these questions, we need a better solution. So, we are using Amazon CloudWatch to monitor the network packets and CPU Utilization of an instance. By watching all these results, user can make decisions whether they should buy that specific instance or not. Also, by combining all results, we are providing some excellent recommendation which will help user to finalize their decision.

The thesis report is explained as follows: Section 2 shows historical research done over predictions based on machine learning for spot prices which helps to find-out challenges. Also, discussed papers on verification and reliability of CSP. Section 1 gives a basic idea about the research methodology and algorithm/tools using for implementation. Section 3 shows how the system will look like. It also consists of flow of implemented framework of the proposed system. Section 5 is showing results got after running the proposed framework with showing key challenges present in model. Finally, Section 6 is concluding this research in brief with the future work.

2 Related Work

Prior work has done over cloud service monitoring on different CSPs Modi et al. (2018) . In that research author implemented an approach to monitor cloud service to see whether CSPs are providing QoS as per SLA or not. Author deployed that model in between user and CSPs so that if SLA violation is detected. By this research, we took an idea of keep monitoring service. But, instead of monitoring service, we are going to check whether all available memory is usable or not. The main factor with Modi et al. (2018) is they are just brokers so user has to invest their money in cloud service and after that broker start its work. Our research gives a predicted idea whether user should take this service or not? Before investment of money, we are providing initial stage for taking decision about cloud service.

Toosi et al. (2016) and Fabra J and A (n.d.) shows bidding strategy is more profitable than a fixed price. It makes maximum profit for providers revenue without any prior knowledge. Results said that, savings by using spot instances are 88% than another fixed type of instances. So, AWS cloud service will be better for our project than other. AWS is offering some reservation feature for EC2 instances. This feature is more cost effective and flexible for any company.

Various techniques used for forecasting and predicting spot prices during runtime or using historical data. We can see all the methods and some machine learning methods used for prediction. Khandelwal et al. (2018) used the Regression Random Forest (RRF) model to predict spot values one day/week ahead. It is helpful for cloud service providers to minimize execution and instance bid failure cost. But challenge for this project was they want more dataset for evaluating results. Agarwal et al. (2017) and Baughman et al. (2018) proposed a method for predicting spot prices using Recurrent Neural Network

[1] `https://aws.amazon.com/cloudwatch`

(RNN) and Long Short-Term Memory (LSTM) networks. But as per their results, time complexity required for this algorithm is more than time series, they proved. Alkharif et al. (2018) used time series analysis technique to predict spot prices. The author used the SARIMA model for prediction and finally conclude that SARIMA is 17% better than other time series models. But for our research, we just had limited time dataset and as per SARIMA model, it needs more attributes in our dataset so that we can add seasonal data and do the predictions. By looking towards dataset and other algorithm we decided to use ARIMA model for prediction. In Contreras et al. (2003), Auto-Regressive Integrated Moving Average (ARIMA) technique is used to predict next-day electricity prices. ARIMA technique has accuracy and good statistical soundness. Our dataset contains time series data which require for ARIMA model and using it for spot price prediction.

Modi et al. (2018) explains QoS and service availability of an instance to be as a broker. Also, Lee et al. (2014) and Ibrahim et al. (2016) implemented SLA checker model for end users. They used SLA and network metrics monitoring feature to give run time SLA verification to user. They created their own strategy to monitor metrics. Motivating by this research, we started to work towards instance performance and network metrics monitoring. Instead of making own strategy and wasting of time, we used CloudWatch to monitor real time performance of an instance and based on that recommendations will be provided. As per Stephen et al. (2018), the instances monitored by CloudWatch monitoring gives twice the range of network data transmission than the others.

By reading all previous work, we finalize our model towards AWS cloud service monitoring system. Instead of working as a broker, we are trying to make a framework which will help user to make decision. This decision will be helpful for user to use cloud service by comparing predicted spot price and real time network metrics of an instance.

3 Methodology

As per previous work, our motive is to give recommendation based on real time values for user decision for investing in cloud instance. We will go through detailed architecture of our framework in this section.

Out framework consist of three important factors which results recommendations for user. First is user interface and based on that instance reliability. User can enter his instance regions and time. Based on that inputs system will check network bandwidth, CPU utilization and network packet rate by using CloudWatch. CloudWatch will monitor two instances from different regions and will provide values based on real time operations. Second important factor is future spot price prediction. We know that, Amazon using on demand and bidding strategy for their instance. But for better investment, spot instances are cheaper that on demand. So, we are providing future bid prices for user, so they can bid exact amount given by this section. And at last, recommendation section will compare all real time values from both sections and give exact advice that user can able to take further decisions before purchasing an instance.

3.1 Project Components

1. **Amazon CloudWatch:**

   As per Figure 2, Amazon CloudWatch is a instance monitoring service provided

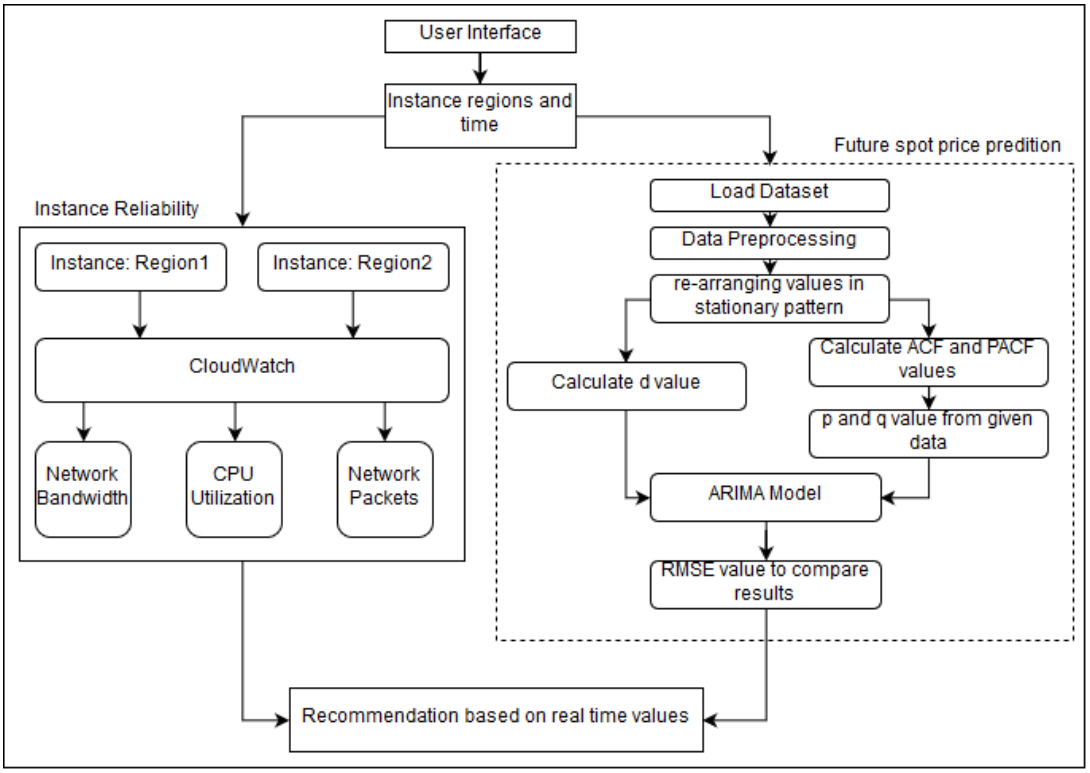


Figure 1: Architecture Diagram of our proposed model

for developers, managers and system operators by Amazon. Using CloudWatch, user can monitor health, utilization and performance of an instance. It collects real time data and shows in the form of logs, metrics and events so that user can discover insights, solve troubleshoot issues and take decisions over an instance. In our project, we are using CloudWatch to collect real time metrics of an instance so that we can display those values on user interface. We enabled deep monitoring feature provided by CloudWatch and started collecting CPU utilization, Network Packets In/Out and bandwidth. These values are accessed by using CloudWatch APIs on front end view.

We created two spot instances with different regions and deploy some workload on it. CloudWatch monitors those instances and gives real time values to our framework via APIs and those values in form of numbers and metrics. These results based on CPU Utilization, Network packets and bandwidth. Those metrics we are showing to user and passing to our recommendation model for further calculations.

2. **Prediction Model:**

For predicting future bid price of Amazon spot instance, we used ARIMA model as per Figure 3. ARIMA model consist of steps for predictions such as visualization of time series, stationarize series, plotting of ACF/PACF values with optimal parameters, ARIMA model and finally predicted results. These steps are responsible for predictions and gives predicted spot price for bidding to user. Our dataset has values based on time series. So, we can use that for this model. First, we load that dataset and removed all unwanted things like spaces and N/A values. Preprocessing contains removing of extra columns and rows which are not useful like in our case,

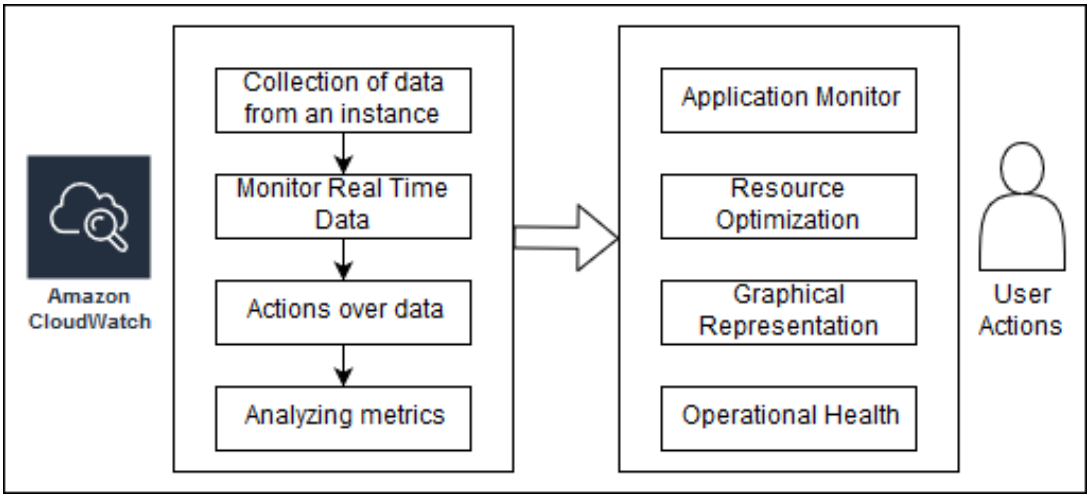


Figure 2: CloudWatch features and user actions

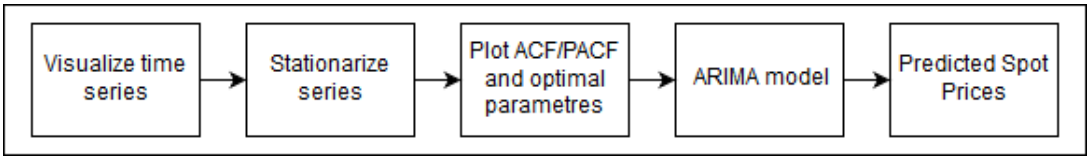


Figure 3: ARIMA Spot Price Prediction Model

we are working on LINUX platform so rows which having Windows and SUSE as a operating system type, removed from our dataset. And data is filtered as per requirements which contains date, time and instance cost.

After pre-processing of data, we re-arrange data as per pattern of stationary graph. So that we can easily calculate (p,d,q) values. These values are performing major role in ARIMA model. We can easily calculate d value from that given data but for p and q value we need plot of ACF and PACF graph. These values are important to select input parameters for ARIMA model.

P and q values can easily determine from ACF/PACF plots. After getting all (p,d,q) values, we passed those values to ARIMA model and calculate predicted future values. In our case, here, we got future spot price for an instance. But, Are these values accurate or not? To see this, we compared those values by calculating RMSE value. RMSE is the part of model which checks performance of deployed model and checking predicted values with actual values. Finally, giving status whether our predicted values are accurate or not.

3. **Recommendation Model:**

   Recommendations based on network packets and its transmission speed with CPU utilization rate of different instances. We proposed algorithm for giving recommendations for selecting perfect instance. We run some workload on instances on different regions. And, simultaneously, by taking help of Amazon CloudWatch, we get inputs for our algorithm. Those values we compared by using our proposed algorithm and as output, recommendations was provided so that user can decide

from which region, they should buy an instance.

3.2 Algorithms

- Recommendation Model: The proposed recommendation model is implemented as per following algorithm. Table 1 describes the variables used in algorithm.

Algorithm 1 Recommendation Algorithm

Result: Region of an Instance
while *reg1 != reg2* **do**
  if $reg1.nw \geq reg2.nw$ **then**
    if $reg1.nwpackets \geq reg2.nwpackets$ & & $reg1.cpu < reg2.cpu$ **then**
      return reg1
      break
    else
      return reg1
    end
  else
    return reg2
  end
end

Table 1: Variables used in algorithm

Variables	**Description**
reg1	Region 1
reg2	Region 2
reg1.nw	Internet speed for region 1
reg2.nw	Internet speed for region 2
reg1.nwpackets	Number of packets transferring to region 1
reg2.nwpackets	Number of packets transferring to region 2
reg1.cpu	Percentage of CPU utilization for region 1
reg2.cpu	Percentage of CPU utilization for region 2

3.3 Sequence Diagram

Sequence diagram of our proposed framework shows working of our implemented model. User is providing inputs for our framework in terms of details of an instance such as instance type, regions, time, etc. System will forward these inputs to CloudWatch so that system can integrate real time metrics on system UI. Also, time and dates are forwarded to ARIMA model by system. It will give predicted price for same date and time back to the system. After getting all values, system will use our recommendation algorithm and give recommendations to the user. That sequence diagram is shown in Figure 4.

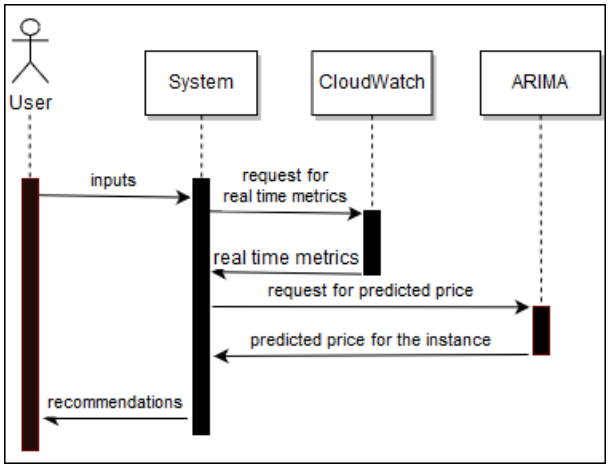


Figure 4: Sequence Diagram of proposed model

4 Implementation

This section explains proper steps used during implementation of our model. Detailed code is available at Narkhede (2018). Following topics will give brief idea about implementation and steps with figures:

1. **Application GUI:**

   In order to implement our framework, we used C#.net as a web application development platform, output shown in Figure 5. In C#.net, we can easily integrate API in our application. We needs the Amazon CloudWatch metrics for our recommendations. Amazon CloudWatch is providing API's for various metrics. So, we are going to use their maximum values of NetworkIn, NetworkPackets and CPU Utilization rate so we can process with those inputs for our recommendation model. As per our prediction model, price for the instance at 9AM is $0.0495 which is shown in Figure 5.

2. **Amazon EC2 Instance:**

   For testing the performance of a AWS EC2 instance, we created two instance over two different regions, such as Europe-central and Asian Pacific-South shown in Figure 6. We created same type of instance on different region. For results, we run same workload on both server at same time. Monitoring is present at bottom side of the page of instance shown in Figure 6. So, that we can also see our cloud metrics on AWS EC2 instance dashboard.

3. **ARIMA Model:**

   As per Figure 3, we are using spot price prediction in our proposed framework. As per our literature survey, we decided to use ARIMA for prediction of spot prices. ARIMA model is implemented by python. Our dataset contains just 3 months pricing history of the spot instances. So, we applied ARIMA model for forecasting values of the spot instances and by taking average values, we calculated spot prices as per region, time and date.

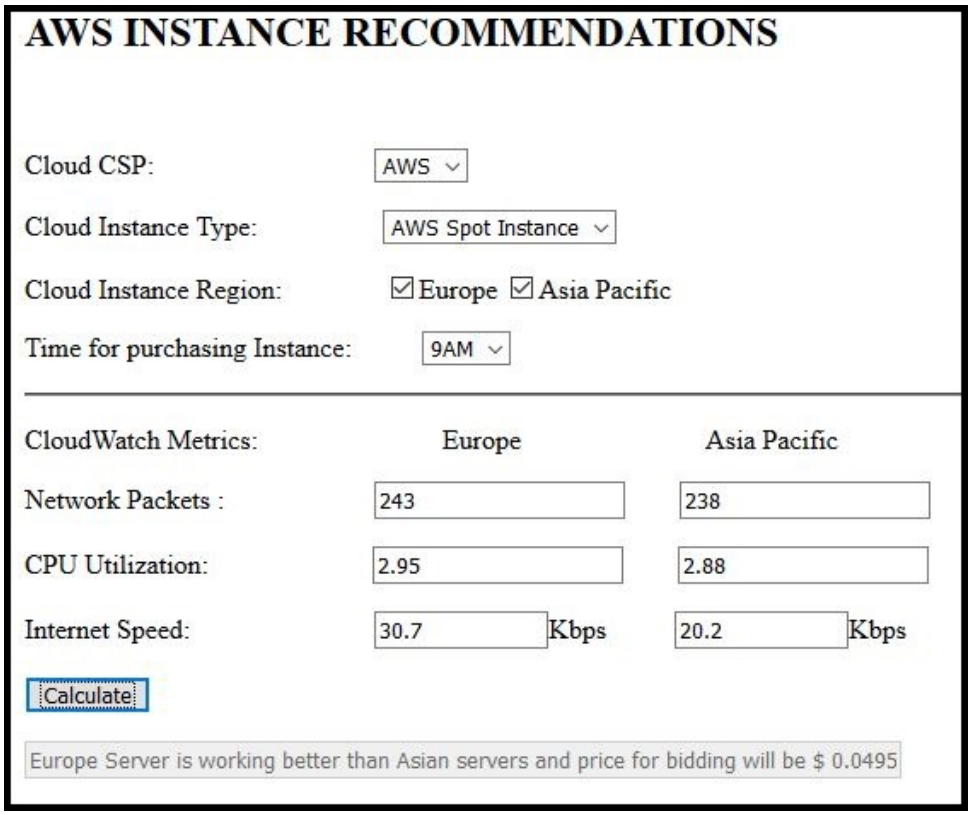


Figure 5: Application GUI

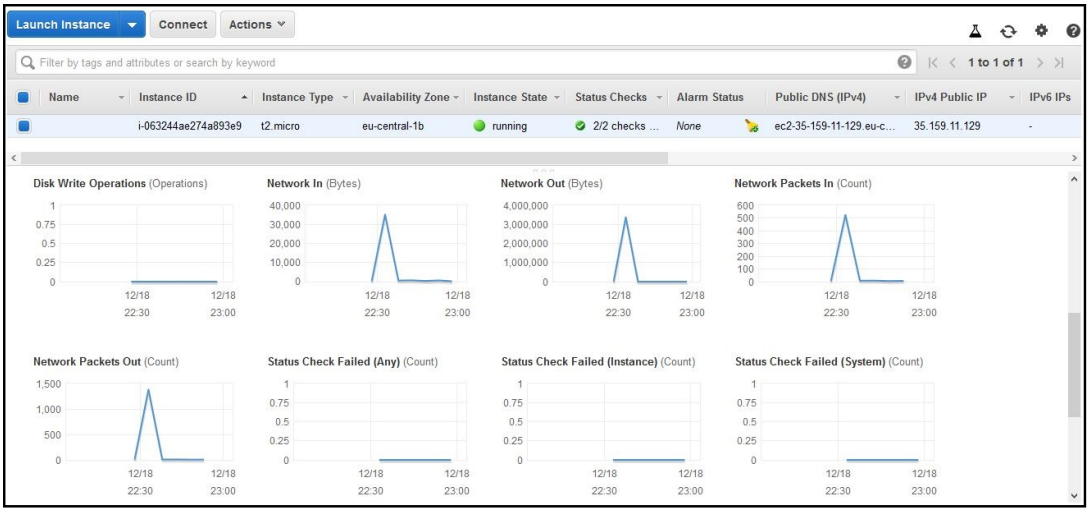


Figure 6: EC2 Instance Dashboard: EU-Central Region

- Data Pre-processing: In data pre-processing, we divided our data as per time zone. For example, our first test held at 9AM of Europe time zone so we filter our dataset as per time and region of the server. Also, we are giving predictions with help of instance type, so we again filtered our dataset with instance type. Now, we got the data which is for 3 months and with specific region, time and instance type.
- Re-arrangement of values in stationary pattern: For ARIMA model, we need to check stationarity our data by using some tests such as plotting rolling statistics and Dickey-Fuller Test. These tests gives assurance whether our data is stationary or not. Our data was not stationary, so we tests their trend and seasonality which made our data stationary by transforming logs in our data. Results after shifting logs is shown in Figure 7

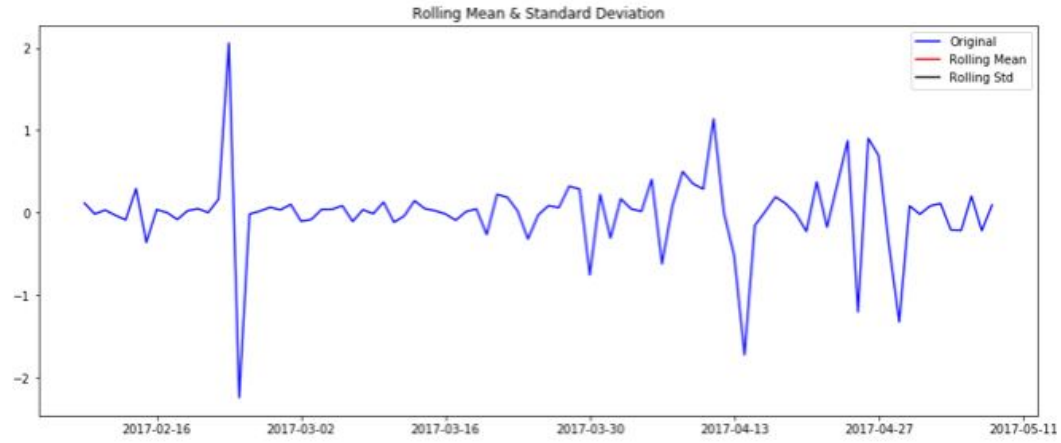
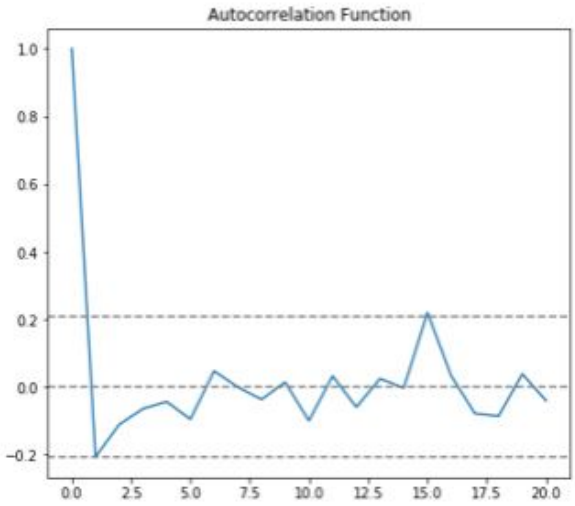
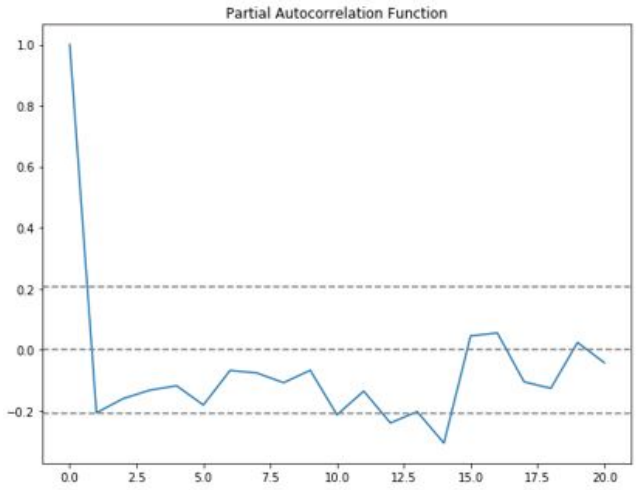


Figure 7: Dickey-Fuller Test after transferring logs in data

- d value calculation: Decomposition technique is used to maintain trend and seasonality of our data. As per model, ether 0 or 1 is used to calculate results. If we want to pass first order difference, then d=0 and to pass original value, d=1 should be proper value. Finally, both values will print same results. In our case, we took d=1 and passed original values for our values.
- ACF/PACF Values: Auto-correlation function (ACF) is used to calculate q value and Partial Auto-correlation function (PACF) used to calculate p value which are required by ARIMA model. In our case, our both values are 1, as per Figure 8
- ARIMA Model: ARIMA consist of combination of three graphs/values. Auto-regressive(AR), Moving Average(MA) and integration of both AR and MA together. For MA and AR graphs, we put input (p,d,q) as (1,1,0) and (0,1,1). So, after integrating both, ARIMA model created by values (1,1,1). Model forecast values for our dataset and predict the spot prices which are shown in Figure 9



(a) ACF, q = 1



(b) PACF, p = 1

Figure 8: ACF/PACF values

Figure 9: Integration of AR and MA model as ARIMA model

- RMSE Value: Root-mean-square-error (RMSE) is the error value which helps to findout accuracy of our deployed model. Here, our data is compared with predicted values and error value is comes as output with which we can easily predict that our results are accurate or not. Here, in our case, Figure 11 shows RMSE value of our results, RMSE = 0.1000 showing that error value is less so our results are accurate as per Grace-Martin (2008).

Date	Cost
2017-03-01	0.043584
2017-03-02	0.044208
2017-03-03	0.046994
2017-03-04	0.051741
2017-03-05	0.055909
2017-03-06	0.059323
2017-03-07	0.060653
2017-03-08	0.065083
2017-03-09	0.068739
	...

Figure 10: Forecasted values by ARIMA model

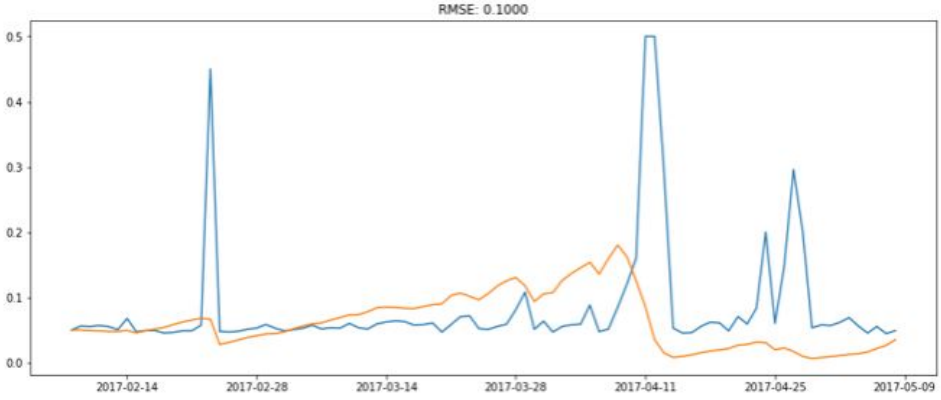


Figure 11: RMSE Value

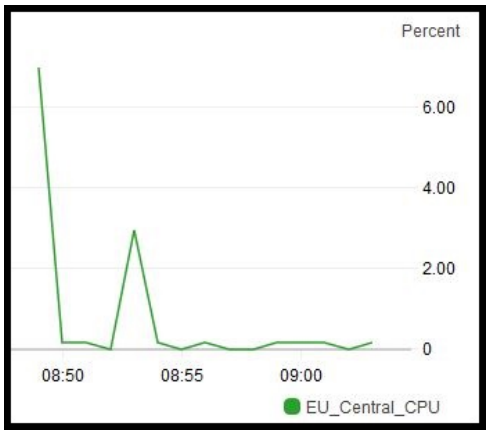
5 Evaluation

Evaluation has been performed based on our CloudWatch metrics. We created two instances, one in EU-Central region and other in AP-South region. After running those instances by using PuTTy[2], we created same workload on both instances. That workload consist of C++ program of shell sort which is having time complexity as O(n.log n) and compiled that program on both instances. That workload executed on both instances at the same time. Simultaneously, looking to Amazon CloudWatch and took real time cloud metrics. From all those metrics, we only selected three important types such as CPU Utilization, NetworkIN and Network Packets which are the key inputs for our recommendation model.

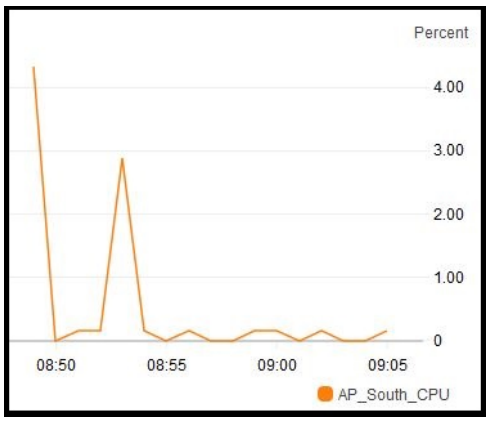
These result were taken two times in a day. First result has been taken by nearly 9AM and other at 1PM. Results are shown in Subsection 5.1 and Subsection 5.2.

[2] `https://www.putty.org`

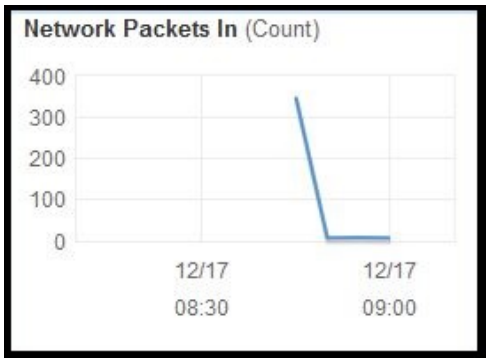
5.1 Recommendations at 9AM:



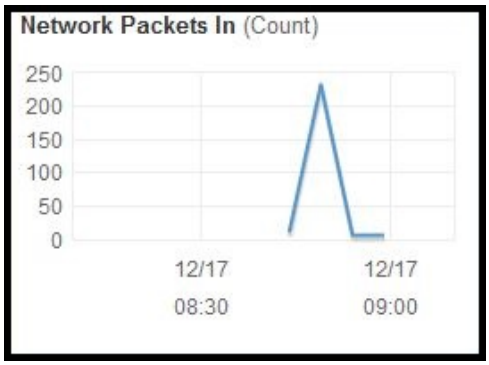
(a) CPU Utilization of EU Instance



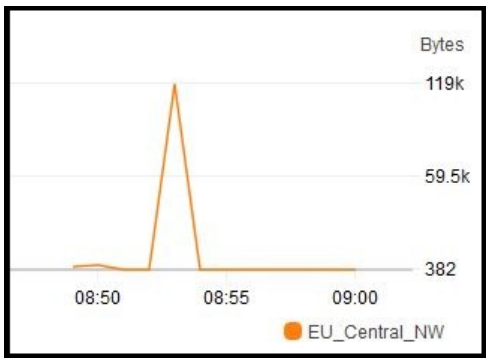
(b) CPU Utilization of AP Instance



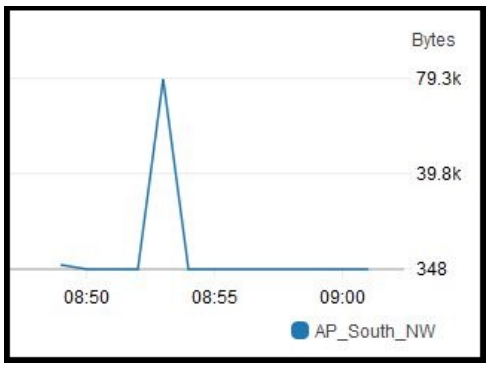
(c) Network Packet Transmission at EU Instance



(d) Network Packet Transmission at AP Instance



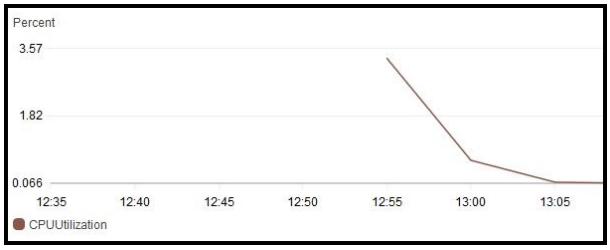
(e) Internet Speed for EU Instance



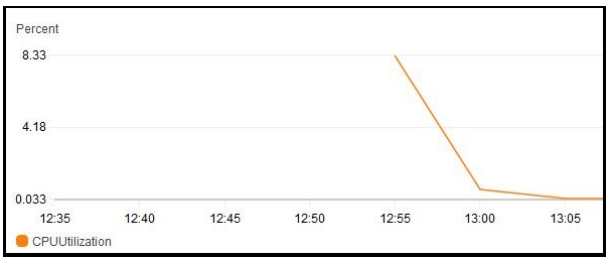
(f) Internet Speed for AP Instance

Figure 12: Metrics showing comparison between Europe(EU) and Asian Pacific(AP) instance at 9AM.

5.2 Recommendations at 1PM:



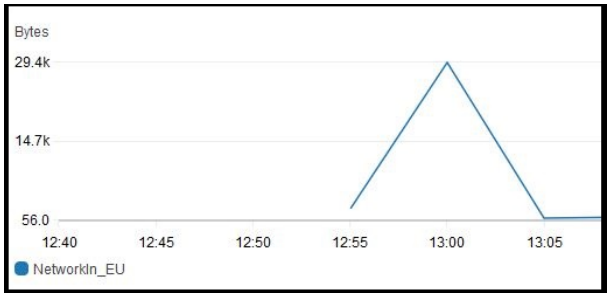
(a) CPU Utilization of EU Instance



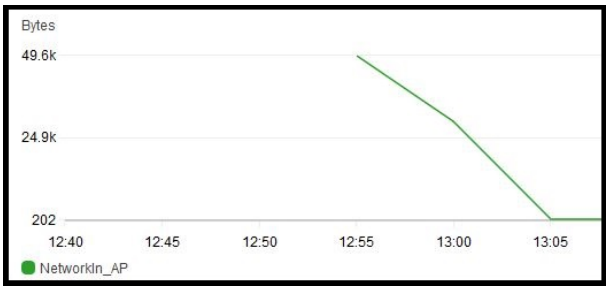
(b) CPU Utilization of AP Instance

(c) Network Packet Transmission at EU Instance

(d) Network Packet Transmission at AP Instance



(e) Internet Speed for EU Instance



(f) Internet Speed for AP Instance

Figure 13: Metrics showing comparison between Europe(EU) and Asian Pacific(AP) instance at 1PM.

5.3 Discussion

Two evaluations has been done by using Amazon CloudWatch as results shown in Figure 12 and Figure 13. Both results were taken at different time and, by result, we got to know that instance performance varies time by time. As per Figure 12, EU-Central region has transferred 243 packets with the speed of 30.7 Kbps network speed and the CPU Utilization was 2.95% of the allocated memory. If we can see, AP-South region transferred 238 packets with the speed of 20.2Kbps which means for the same workload Europe region performs faster than Asian servers at the time of morning. Whereas, as per Figure 13, AP-South transferred 443 packets with the speed of 29.9Kbps speed and EU-Central transferred just 429 packets with 29.4Kbps speed. So, now we can see the difference that Asian servers performing more better than Europe servers.

Finally, our results justifies that the morning time of Europe servers were not busy but at the afternoon time, there were more traffic on servers in Europe than Asian servers. Our results shows the real time metrics with differentiating load of different servers.

6 Conclusion and Future Work

The proposed model is a real-time recommendation model for the users. It works with Amazon CloudWatch for real-time values and providing proper outcomes as recommendations to user. User can easily get an idea about which region server they should purchase. Recommendations contains the server which will be more faster and reliable than other servers. We successfully proved that if a user from Europe region, wants to take Europe server in morning time, then, that would be a better decision. But, if a user think about more traffic over Europe servers at afternoon time because of working period and so purchasing an instance other than home region, will be the better option for a user. Also, our system is providing some predictions of Amazon spot instance price which a user can use while bidding. If a user is planning to buy a spot instance which is cheaper than other instances, our system will helps the user to give proper bid price for proper choice of server. Those recommendations are based on server's region and purchase time of an Instance.

Our system will be helpful for the users who are thinking before the investment in a cloud instance. Instead of purchasing any cloud instance, they can check the performance by using our system and then take their decisions. Recommendations of our system will be helpful for the user in their further working on that cloud instance. As per our research question, surely, our framework will enhance the user decisions' over purchasing a cloud instance.

However, the future scope for this system is to add more CSP and instance type with their real time metrics. So that, user can compare performance of the instances of different CSPs as well as will decide to go with proper CSP and instance type. Also, our dataset for ARIMA is too old, so that our prices might not be sufficient for all type of spot instance. So, if we get real time price history for predictions, then results of our system will be more accurate. Our system is showing instance availability for real time situation, but if we store all real time data and use machine learning, then we can predict future availability of an specific instance.

References

Agarwal, S., Mishra, A. and Yadav, D. (2017). Forecasting price of amazon spot instances using neural networks, *International Journal of Applied Engineering Research* **12**: 10276–10283.

Alkharif, S., Lee, K. and Kim, H. (2018). Time-Series Analysis for Price Prediction of Opportunistic Cloud Computing Resources, *in* W. Lee, W. Choi, S. Jung and M. Song (eds), *Proceedings of the 7th International Conference on Emerging Databases*, Lecture Notes in Electrical Engineering, Springer Singapore, pp. 221–229.

Amazon (2018a). Create your ec2 resources and launch your ec2 instance.
URL: *https://docs.aws.amazon.com/efs/latest/ug/gs-step-one-create-ec2-resources.html*

Amazon (2018b). Getting started with amazon cloudwatch.
URL: *https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/GettingStarted.html*

Baughman, M., Haas, C., Wolski, R., Foster, I. and Chard, K. (2018). Predicting amazon spot prices with lstm networks, pp. 1–7.

Contreras, J., Espinola, R., Nogales, F. J. and Conejo, A. J. (2003). ARIMA models to predict next-day electricity prices, *IEEE Transactions on Power Systems* **18**(3): 1014–1020.

Developers (n.d.). Shell sort - sorting algorithm animations.
URL: *https://www.toptal.com/developers/sorting-algorithms/shell-sort*

Dignan, L. (2018). Top cloud providers 2018: How aws, microsoft, google, ibm, oracle, alibaba stack up.
URL: *https://www.zdnet.com/article/top-cloud-providers-2018-how-aws-microsoft-google-ibm-oracle-alibaba-stack-up/*

Fabra J, Hernandez S, A. P. E. J. R. A. and A, M. (n.d.). A History-Based Model for Provisioning EC2 Spot Instances with Cost Constraints | springerprofessional.de.
URL: *https://www.springerprofessional.de/en/a-history-based-model-for-provisioning-ec2-spot-instances-with-c/12497328*

Grace-Martin, K. (2008). Assessing the Fit of Regression Models.
URL: *https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/*

Howtech (2013). How to Use Putty with SSH Keys on Windows.
URL: *https://www.youtube.com/watch?v=1wQ8wQfa7lw*

Ibrahim, A. A. Z. A., Kliazovich, D. and Bouvry, P. (2016). Service level agreement assurance between cloud services providers and cloud customers, *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 588–591.

Jain, A. (2018). Complete guide to create a time series forecast (with codes in python).
URL: *https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/*

Khandelwal, V., Chaturvedi, A. and Gupta, C. P. (2018). Amazon ec2 spot price prediction using regression random forests, *IEEE Transactions on Cloud Computing* pp. 1–1.

Lee, J., Kim, J., Kang, D., Kim, N. and Jung, S. (2014). Cloud service broker portal: Main entry point for multi-cloud service providers and consumers, *16th International Conference on Advanced Communication Technology*, pp. 1108–1112.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. and Ghalsasi, A. (2011). Cloud computing The business perspective, *Decision Support Systems* **51**(1): 176–189.
URL: *http://www.sciencedirect.com/science/article/pii/S0167923610002393*

Modi, K. J., Chowdhury, D. P. and Garg, S. (2018). An Ontology-Based Approach for Automatic Cloud Service Monitoring and Management, *in* E. B. Rajsingh, J. Veerasamy, A. H. Alavi and J. D. Peter (eds), *Advances in Big Data and Cloud Computing*, Advances in Intelligent Systems and Computing, Springer Singapore, pp. 1–16.

Narkhede, P. (2018). Decision making framework for purchasing a cloud instance.
URL: *https://github.com/psn30595/AWSInstanceRecommendations*

Stephen, A., Benedict, S. and Kumar, R. A. (2018). Monitoring iaas using various cloud monitors, *Cluster Computing* pp. 1–13.

Toosi, A. N., Vanmechelen, K., Khodadadi, F. and Buyya, R. (2016). An Auction Mechanism for Cloud Spot Markets, *ACM Trans. Auton. Adapt. Syst.* **11**(1): 2:1–2:33.
URL: *http://doi.acm.org/10.1145/2843945*

Wang, L., von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J. and Fu, C. (2010). Cloud Computing: a Perspective Study, *New Generation Computing* **28**(2): 137–146.
URL: *https://doi.org/10.1007/s00354-008-0081-5*

Appendices

A Configuring AWS Instance:

This section consists of installation of an Amazon Web Services (AWS) Instance. For our project, we need two instances with two different regions. As an example, we took Europe-Central and Asian Pacific-South regions for our research.

Table 2: Instance Configuration

Component	**Configuration**
EC2 Instance	t2.micro
Instance Region	EU-Central(Frankfurt) and AP-South(Mumbai)
Instance CPU and RAM	Default
Instance real-time metrics	Amazon CloudWatch
SSH for the Instance	PuTTY

1. **Amazon EC2 Instance**[3]**:** We created a trial account on AWS which gives free access for 12months and 750hrs of the EC2 instance. First, we select our region from the top right corner as shown in Figure 14. By clicking on **"Launch Instance"**, we can create and launch our instance. Configuration we are selecting default so after clicking on that button just gives next and finally launch our instances Amazon (2018a).

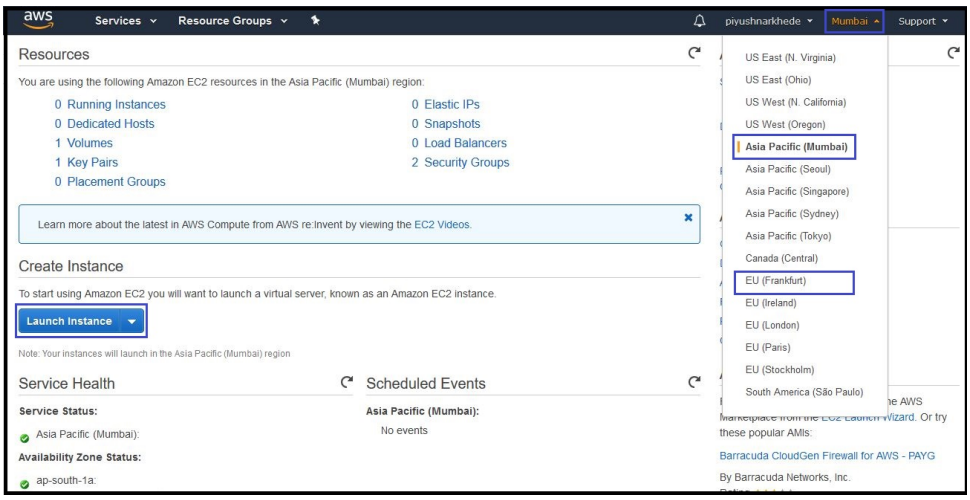


Figure 14: AWS EC2 Dashboard

For our project, we created 2 instance with different regions. Figure 15 shows the AWS EC2 instance launched on Europe server and type of that instance is **t2.micro**. Figure 16 shows another instance we launched on Asian server with type of **t2.micro**.

[3] `https://aws.amazon.com/ec2`

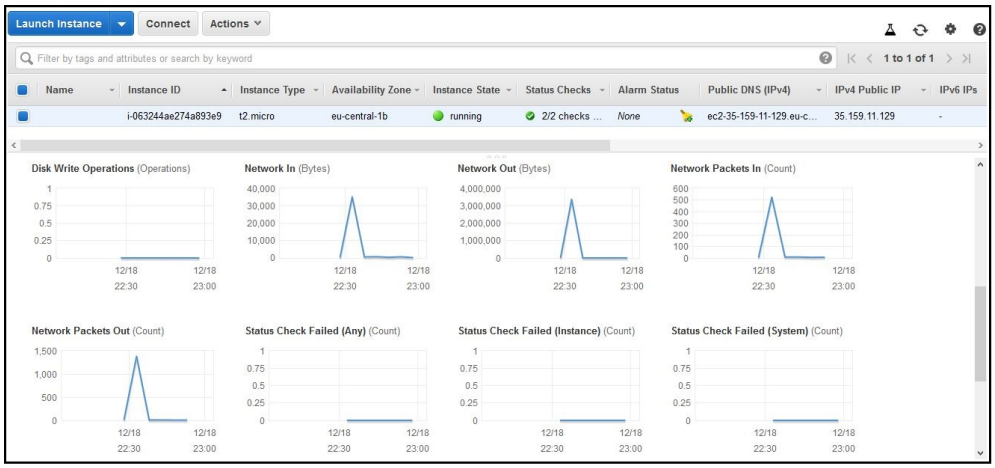


Figure 15: AWS EC2 Dashboard for Europe server

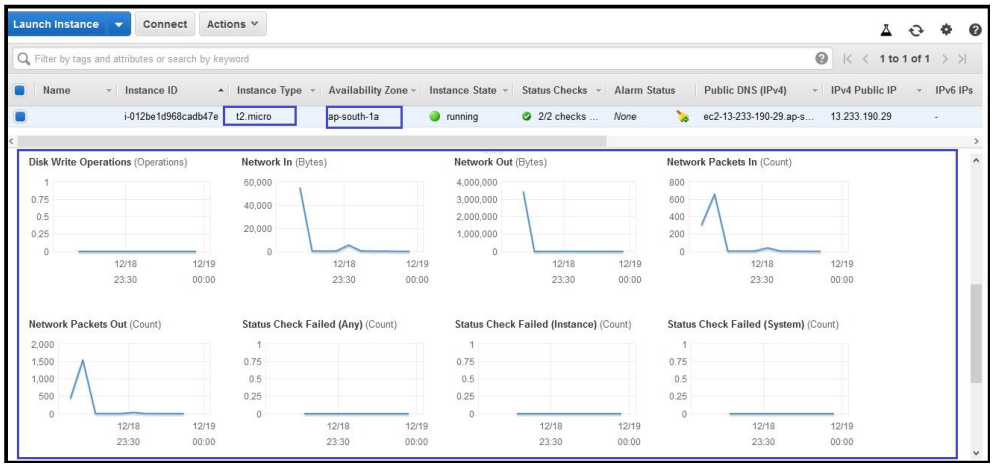


Figure 16: AWS EC2 Dashboard for Asian server

2. **PuTTy**[4]**:** PuTTy is used to run our EC2 instance by using SSH. For PuTTY, we need private key generated by PuTTy Key Generator Howtech (2013). Figure 17 shows the working of key generator. Click on **"Load"** to load private key, **.pem** file provided by AWS and converting into putty key. After successfully loading of key, it will give message as Figure 18 . We can save that key by using button **"Save private key"**.

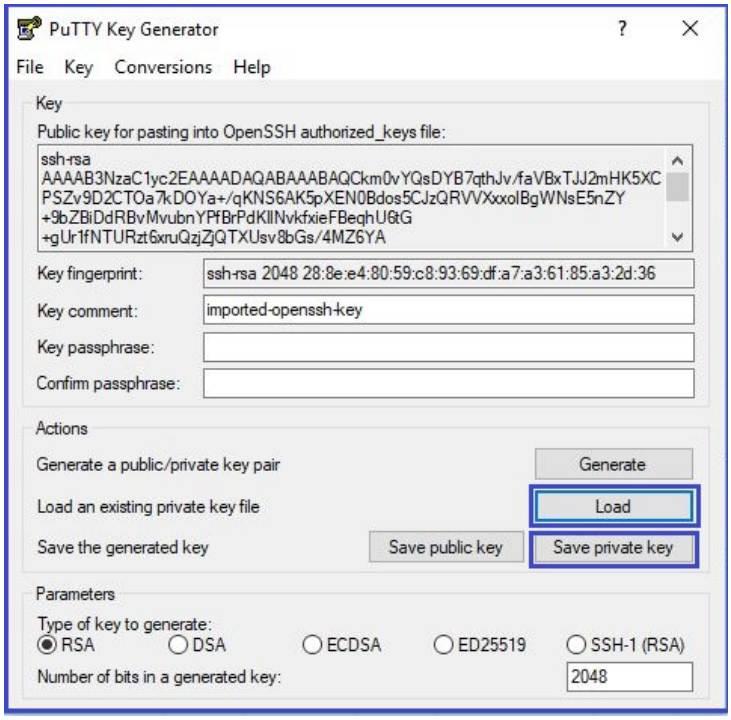


Figure 17: PuTTy Key Generator

Figure 18: PuTTy Key Generated Successfully

Now, for using EC2 instance we need to connect by using SSH. So, EC2 dashboard has one button **"Connect"**, just click it and Figure 19. It contains Host IP required by PuTTy as well as username and private key name on it.

Just copy those information and open PuTTy. In PuTTy, first we need to add our username in **"Data"** option of left side list. Click on Data and put our username

[4] `https://www.putty.org/`

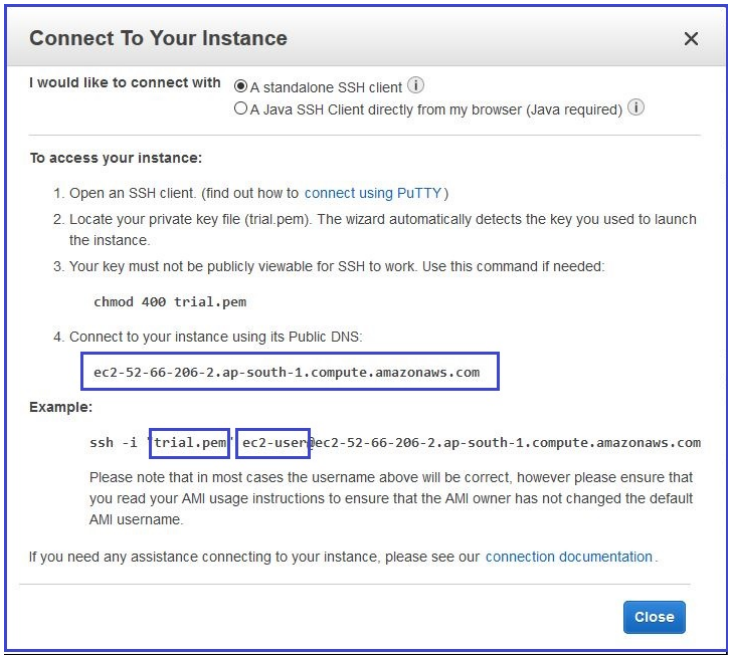


Figure 19: Connect your EC2 Instance

as per Figure 20. After this process, go to **"Auth"** section and upload private key generated by PuTTy Key Generator as per Figure 21

Then add host key in **"Session"** option on left corner as per Figure 22 and click on **"Save"** to save that details of an Instance. After setting up, click on **"Open"** to launch our terminal which is able to access EC2 instance as shown in Figure 23.

3. **Workload:** For our project, we want some workload which can run same time in both instances. So, we created one workload as a C program. That program contains code of shell sort as per which is having high time complexity Developers (n.d.). Workload is as Figure 24. That workload needs C compiler packages on EC2 instance. For installing those packages we used command: **"sudo yum install gcc"** which installs gcc packages which are required to compile our c program.

4. **Amazon CloudWatch** [5]**:** For getting real-time metrics, we are using Amazon CloudWatch Amazon (2018b). Figure 25 shows the dashboard of Amazon CloudWatch. In that, we can select metrics type as per our requirements as well as we can give statistics as per second or minutes. Also, we can get average of maximum values of real-time metrics. Here, we selected 3 types of metrics, such as **NetworkIN, NetworkPackets** and **CPUUtilization**.

[5] `https://aws.amazon.com/cloudwatch`

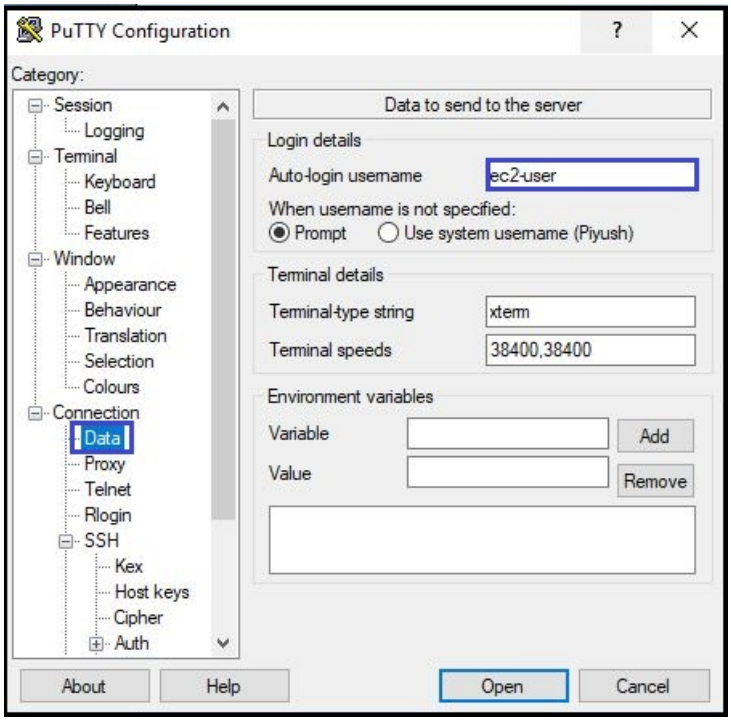


Figure 20: Putting Username in Data section

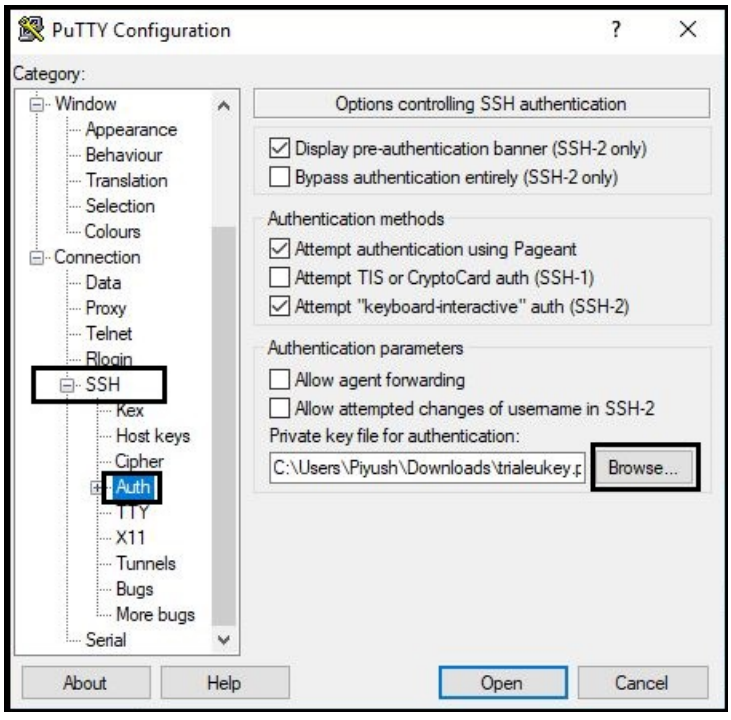


Figure 21: Upload PuTTy private Key

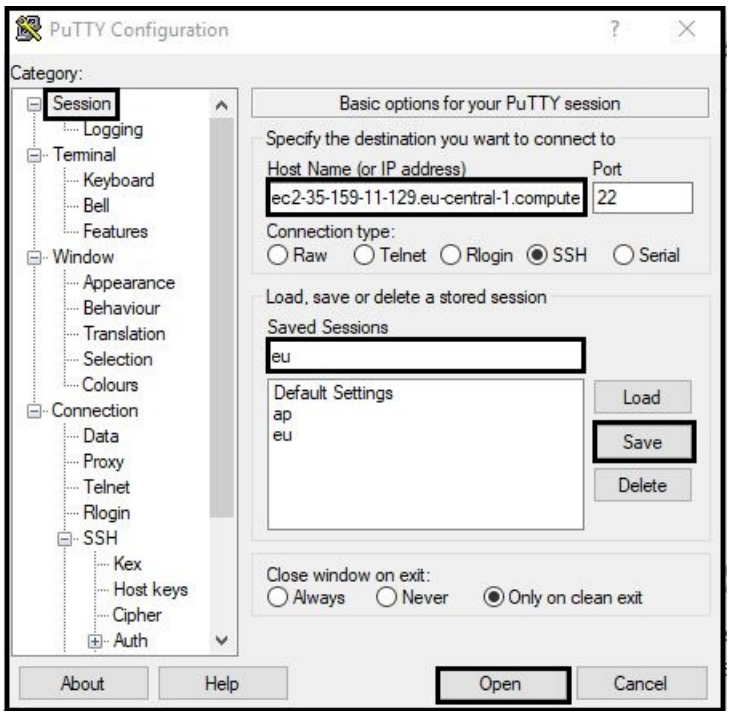


Figure 22: PuTTy Key Generator

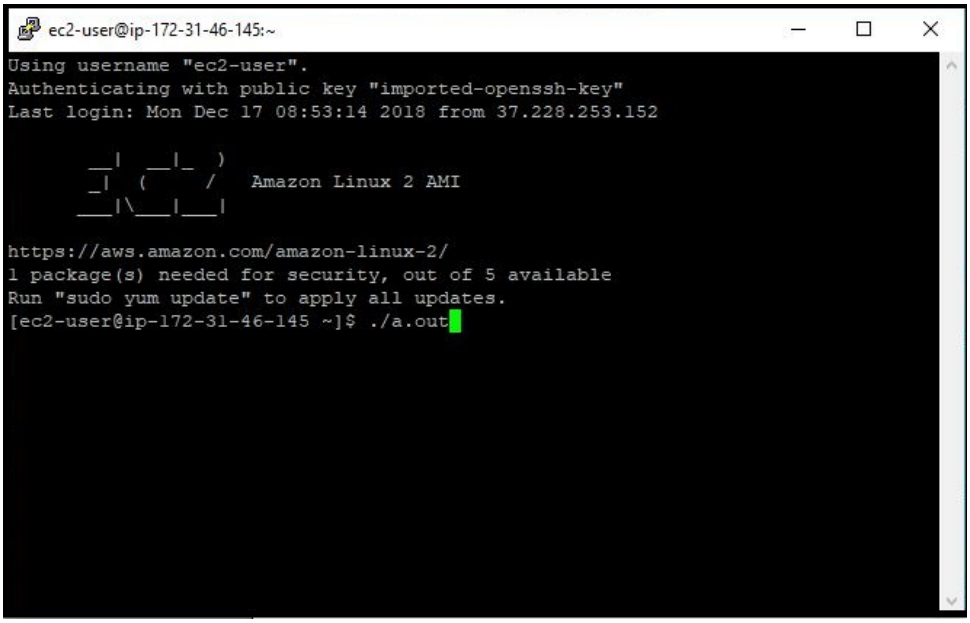


Figure 23: Logged In for an Instance using PuTTy

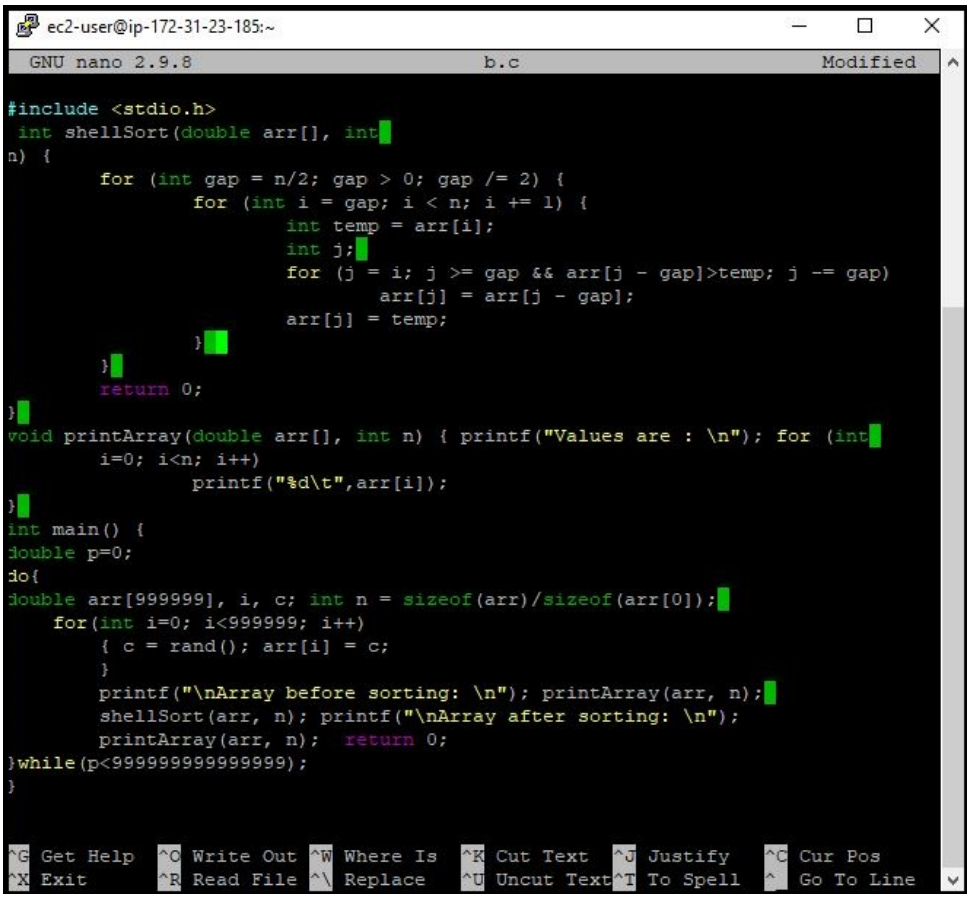


Figure 24: Workload running on both instances

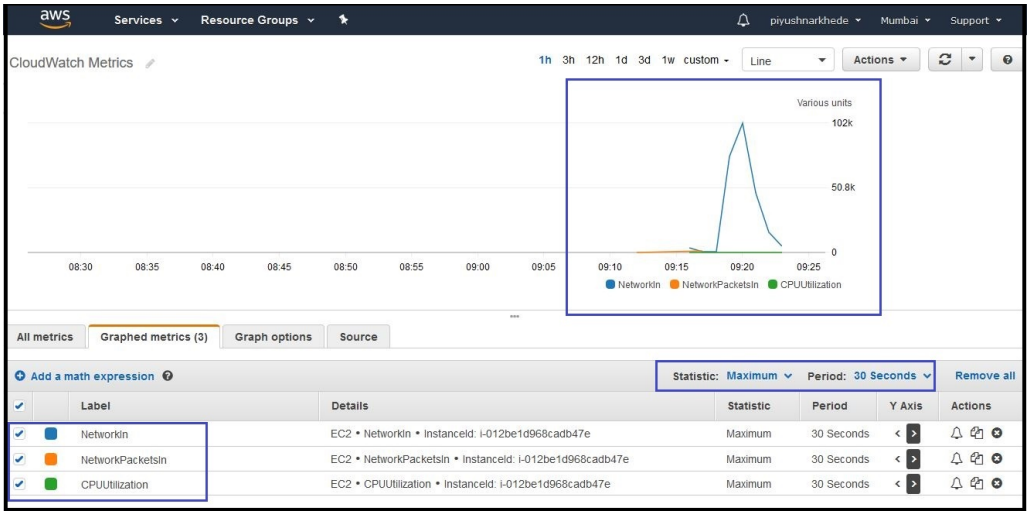


Figure 25: Amazon CloudWatch Dashboard

B ARIMA Model

For predicting spot price, we are using ARIMA model. We are using ARIMA model for forecasting day to day price of an spot instance Jain (2018). After getting that value, we took average of all three month value and prepared a common value dataset which contains values of an Instance for whole month. For using ARIMA model and for data pre-processing, we used Jupyter Notebook[6] for writing and executing code. So, that was easy to implement and generate output for our project. Figure 26 is showing UI of Jupyter Notebook. We can run our code in this notebook as well as we can get output in same window.

Figure 26: Jupyter Notebook

C Application UI

Our application UI is developed in Visual Basic 2017[7] with help of web form template. In which we imported some core libraries of AWS to integrate CloudWatch APIs and our EC2 Instance information in our project. With the help of NuGet Package Manager, we downloaded libraries for AWS such as AWS Core, AWS CloudWatch and AWS EC2. As per Figure 27, we downloaded toolkit as .msi file and installed for Visual Basic 2017. After successful installation, we got AWS explorer as per Figure 28

As per Figure 32, User has to give input as CSP, Instance Type and Time, Region of an Instance. By help of these data, our system will see real time metrics on CloudWatch and give recommendation as per our algorithm. All code developed for UI, workload and ARIMA model is available on Github [8].

[6] https://jupyter.org
[7] https://visualstudio.microsoft.com
[8] https://github.com/psn30595/AWSInstanceRecommendations/

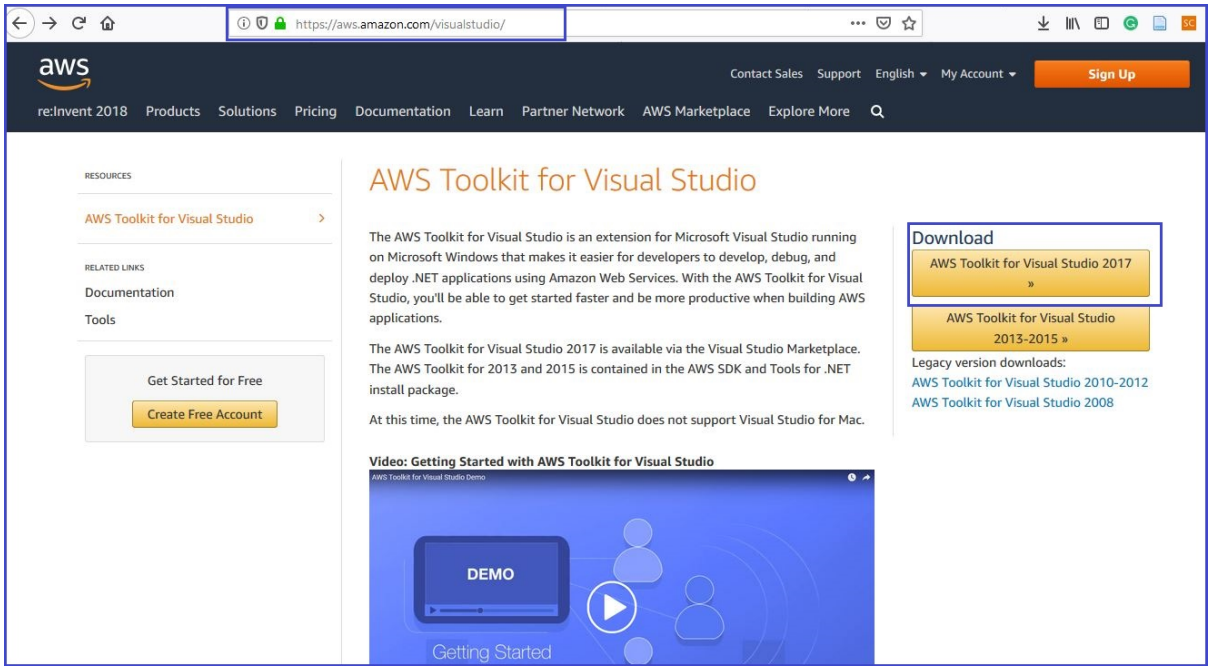


Figure 27: AWS toolkit for Visual Studio 2017

Figure 28: AWS Explorer

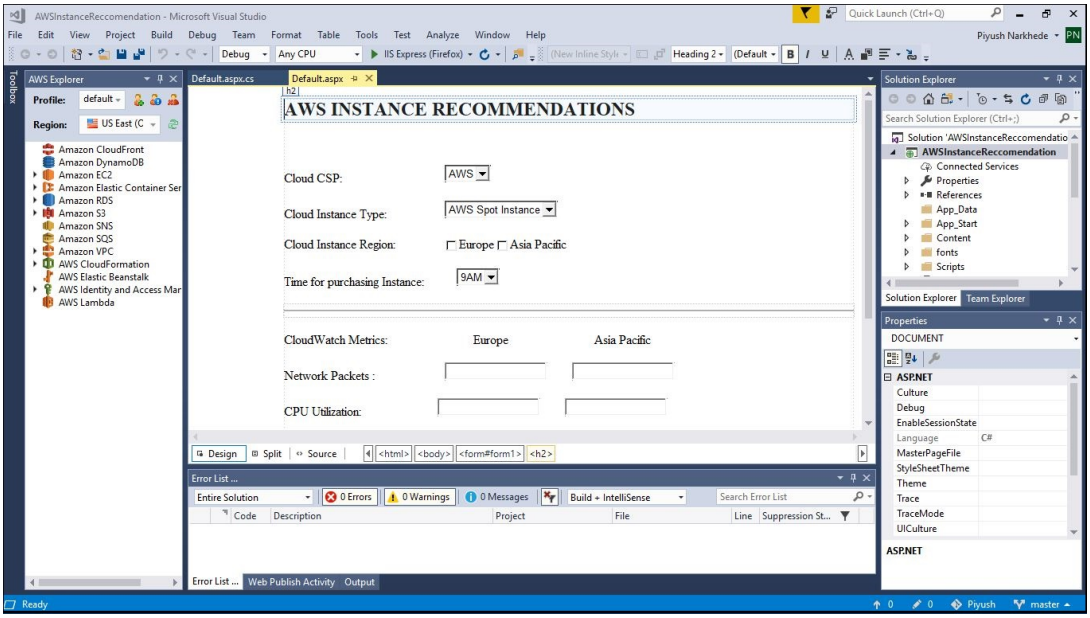


Figure 29: Visual Studio Coding Environment

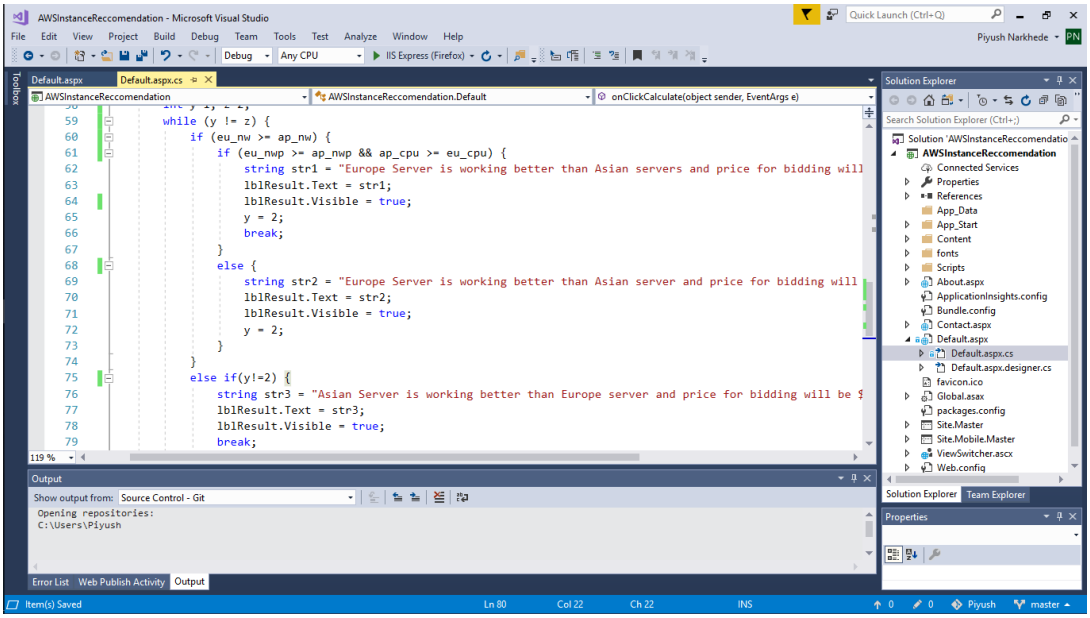
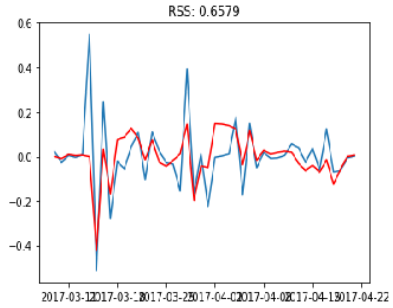


Figure 30: Algorithm for making predictions over an Instance

```
In [18]: #combined model
         model = ARIMA(ts_log, order=(1, 1, 1))
         results_ARIMA = model.fit(disp=-1)
         plt.plot(ts_log_diff)
         plt.plot(results_ARIMA.fittedvalues, color='red')
         plt.title('RSS: %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))

         C:\Users\Piyush\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:171: ValueWarning: No freq
         uency information was provided, so inferred frequency D will be used.
           % freq, ValueWarning)
         C:\Users\Piyush\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:171: ValueWarning: No freq
         uency information was provided, so inferred frequency D will be used.
           % freq, ValueWarning)
Out[18]: Text(0.5,1,'RSS: 0.6579')
```




```
In [19]: predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
         predictions_ARIMA_diff.head(5)

Out[19]: Time
         2017-03-09    0.001094
         2017-03-10   -0.009228
         2017-03-11    0.010686
         2017-03-12    0.005089
         2017-03-13    0.007265
```

Figure 31: Code for ARIMA model with values (p,d,q) as (1,1,1)

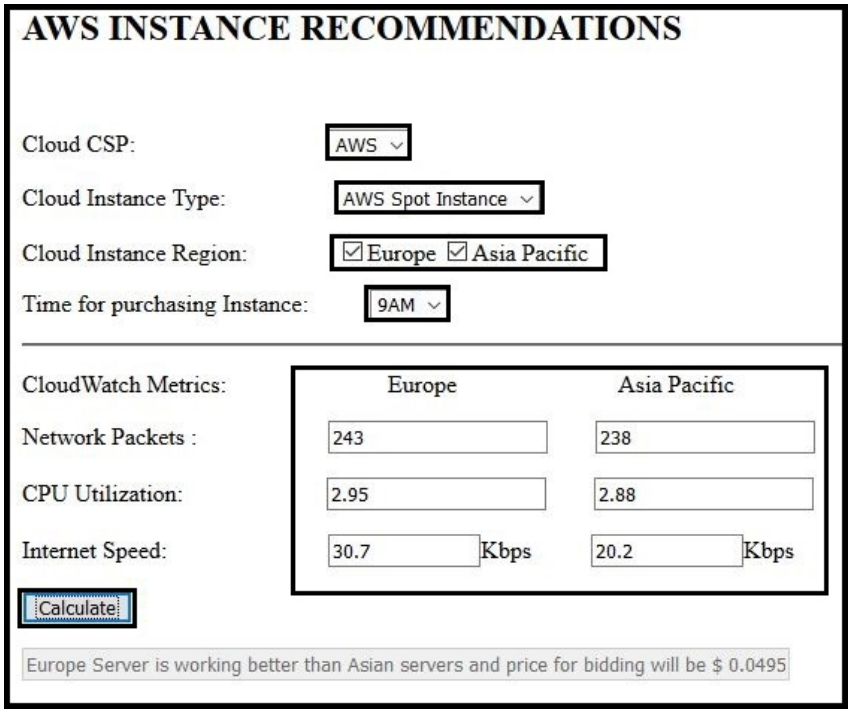


Figure 32: GUI of our system