Containerization and Resource Prediction for Autoscaling Applications using Time Series Analysis

Project/Dissertation (AMC10801) report submitted to

Indian Institute of Technology (Indian School of mines) Dhanbad

in partial fulfilment for the award of the degree of

Integrated Masters of Technology

in

Mathematics and Computing

by

Anshuman Singh

(15JE000969)

Under the supervision of

Prof. S. Gupta

Department of Mathematics and Computing

Indian Institute of Technology (Indian School of mines) Dhanbad

Winter Semester, 2019-20

-

DECLARATION

The Dissertation titled **"Containerization and Resource Prediction for Autoscaling Applications using Time Series Analysis"** is a presentation of my original research work and is not copied or reproduced or imitated from any other person's published or unpublished work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions, as may be applicable. Every effort is made to give proper citation to the published/unpublished work of others, if it is referred to in the Dissertation.

To eliminate the scope of academic misconduct and plagiarism, I declare that I have read and understood the UGC (Promotion of Academic Integrity and Prevention of Plagiarism in Higher Education Institutions) Regulations, 2018. These Regulations have been notified in the Official Gazette of India on 31st July, 2018.

I confirm that this Dissertation has been checked with the online plagiarism detector tool Turnitin (http://www.turnitin.com) provided by IIT (ISM) Dhanbad and a copy of the summary report/report, showing Similarities in content and its potential source (if any), generated online through Turnitin is enclosed at the end of the Dissertation. I hereby declare that the Dissertation shows less than 10% similarity as per the report generated by Turnitin and meets the standards as per MHRD/UGC Regulations and rules of the Institute regarding plagiarism.

I further state that no part of the Dissertation and its data will be published without the consent of my guide. I also confirm that this Dissertation work, carried out under the guidance of Dr. S. Gupta, Professor, Department of Mathematics and Computing, has not been previously submitted for assessment for the purpose of award of a Degree either at IIT (ISM) Dhanbad or elsewhere to the best of my knowledge and belief.

Anshuman Singh
Integrated M.Tech (Mathematics and Computing)

Department of Mathematics and Computing

Admission No.: 15JE000969

(Forwarded by)
Dr. S. Gupta
Professor
Department of Mathematics and Computing

Synopsis

With increasing number of devices joining the internet, the number of "clients" are increasing exponentially and so more servers are getting deployed. Naturally, Cloud Computing is gaining popularity because of features like flexible resource allocations on-demand.

As applications are becoming cloud-native, containerization technology is making it possible for organizations to scale better and to develop and deploy easily. As more and more computing resources are being used, we try to develop a resource prediction strategy which can help us scale better by implementing technologies like Autoscaling and optimize our computing resources cost by effectively using the cloud resources with reduced overhead.

We try to propose a strategy of resource prediction using Time Series Forecasting. We propose a way to generate a dataset for any server and compare forecasting models on such datasets. We find Facebook Prophet Model to be most flexible and relevant for this case.

Acknowledgements

I would like to express my gratitude to my thesis advisor Prof. S. Gupta.for his patience, trust and constant support.
I would like to thank Mr. Sanat Talwar, my colleague from whom I got to learn so many things which will stay with me forever.
I would also like to add a few lines thanking my parents for their selfless support.
Finally I would like to thank my seniors, batchmates and juniors, all of whom have supported, helped and held me to high standards.I shall forever be grateful.

Contents

List of Figures

List of Tables

Abbreviations

K8s	**K**ubernetes
QoS	**Q**uality of **S**ervice
ERU	**E**ffective Resource Utilization
LAN	**L**ocal **A**rea **N**etwork
ARIMA	**Auto Regressive I**ntegrated **M**oving **A**verage
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
API	**A**pplication **P**rogram **I**nterface **P**rotocol
CI/CD	**C**ontinuous **I**ntegrayion/ **C**ontinous **D**elivery
HPA	**H**orizontal **P**od **A**utoscalar
RMSE	**R**oot **M**ean **S**quare **E**rror

Chapter 1

Introduction

Before the development of cost-effective and powerful microprocessors and efficient networks, computing tasks had to be done on a single machine. If the user wanted to execute a task which is not possible for this machine computationally, the only option was to execute it on a bigger, more powerful, and costly supercomputer. Nevertheless, with cheap microprocessors increasing the supply of inexpensive machines, and Local Area Networks(LANs) promoting fast intercomputer connectivity, a modern form of resource-intensive computing developed. A set of independent computers work together as one entity of resource to execute any particular task. It is the principle behind distributed computing. [29]

When more and more people got connected to the Internet, this saw a great surge in the number of ”clients” for web servers. Many operations were computerized through laptops, cell phones tablets, and various IoT devices. Subsequently, vast quantities of computer resources relevant to such activities came into demand, to store the data and execute the computational tasks at such a fast pace. It is hard to imagine this becoming a reality unless multiple numbers of computers work together.

Distributed computing is used in a variety of ways: Cluster Computing, Grid Computing, etc. This dissertation focuses on an example of cluster computing, which is being adopted by the industry rapidly. This happens with the use of a *cluster manager* which accepts, schedules, stops and monitor the applications running of a group of computers running together as a single unit of resource. We will do that using Kubernetes, an open-source cluster manager developed by Google.[10]

Cluster Manager focuses on different factors as a metric of success. One big part of that is Effective Resource Utilization (ERU). Essentially, ERU refers to the percentage of cluster resources that applications currently use for computational or storage purposes. A good question to ask is how many devices can be eliminated from the cluster, while still operating the total workload load of the cluster comfortably. This measure is particularly important as more and more services are utilized by the cluster manager, and less the cluster costs, it is more viable to adopt cluster computing is for the end-user. Another metric to measure the performance is Quality of Service(QoS). It is the ability of a cluster to run an application at the desired performance level even after lots of changes happening in the cluster. Almost every organization keeps searching for the right balance among these two.[31]

With this background in mind, it is not hard to imagine why every application provider is trying to move to cloud-native.Cloud computing provides three types of services to say lucidly: First one PaaS, SaaS and IaaS which provides Platform, Software and Infrastructure as services respectively. [27] see figure 1.1. Cloud storage is the aggregation of minimal resources, automation tools, and networks, which is provided when requested. Cloud infrastructure is best designed for deploying web applications because of its ability to scale on-demand elastically. More and more businesses are increasingly running online services on the cloud, which saves the need for maintaining data servers and also helps not waste resources on the server. It has helped reduce costs dramatically.

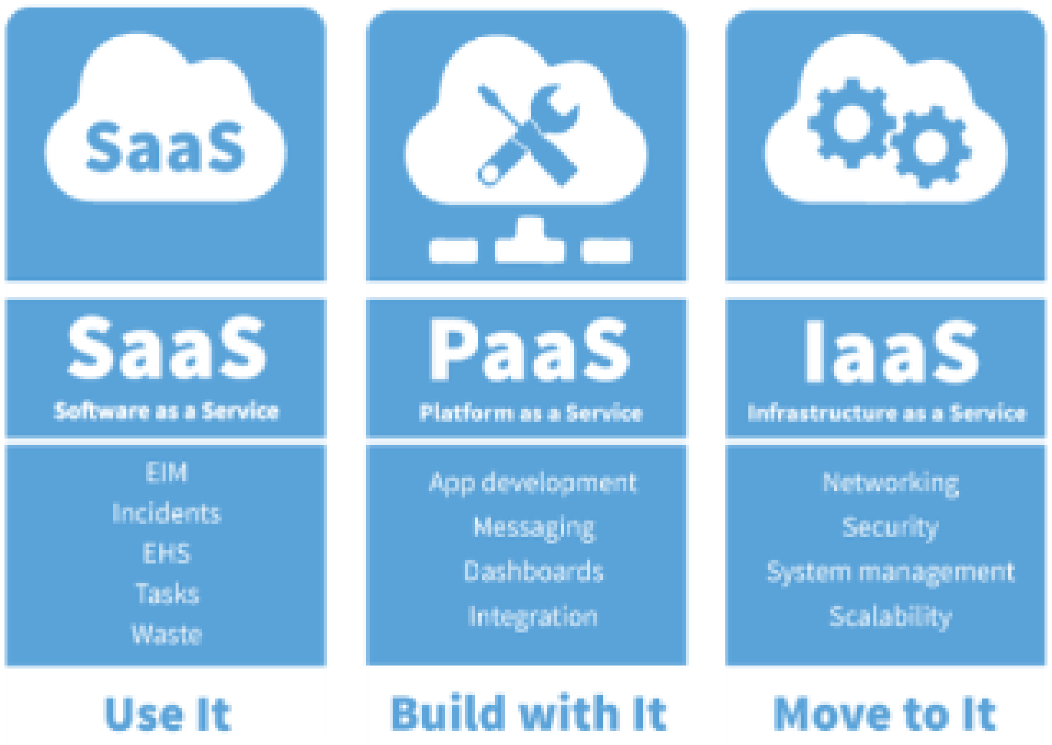


FIGURE 1.1: Differences between SaaS, PaaS and IaaS [27]

1.1 Motivation behind Objective

1.1.1 Need for virtualization

Virtualization is an essential part of modern cloud infrastructure.[32] Many cloud storage data centers nowadays operate hypervisors over their physical computers. A hypervisor is a piece of a computer program that allows virtual computers and operates them. For such hypervisors, and the virtual machines operating on them, network administrators may automate the usage of physical resources accessible and use the most efficient form of the application infrastructure that suits them.

Through virtualization, resources may be used more efficiently than traditional bare-metal systems, which require physical hardware to separate various sections of network infrastructure. However, this can still be improved.Through virtualization, resources may be used more efficiently than traditional bare-metal systems, which require physical hardware to separate various sections of network infrastructure. However, this can still be improved. A hypervisor uses multiple kernels on a single physical computer. Thus it is difficult to separate programs and processes. Mills[4] estimated how 1,500 terawatt-hours of electricity were used annually to fuel cloud storage data centers, which is around 10 percent of the world's energy use, and even that estimate is growing. Hence improvement in this area creates a significant impact.

1.1.2 Why Linux Based Containers

With new advancements around Linux Based Containers, they have become a great alterna.tive for hypervisors. Linux Containers (LXC) is a kernel technology capable of operating several processes, each inside its isolated environment. This is container-based virtualization. Mathijs [1] also compared the performance of Hypervisor and CoreOS and found the LXC containers to be better performing in terms of request-response rate. All this goes to prove that Containerization is the way the industrial infrastructure is going to take shape in the future as it is easier to create and maintain, less costly, and better performing.

Containers can be regarded as lightweight wrappers over a single Unix process. As such, containers need significantly fewer memory and can be started, stooped, and restarted in a matter of seconds. However, such ephemeral bene ts come at the price of reduced isolation between two containers operating on the same physical host. The most significant advantage is also the ease of scalability these provide.

We shall dive into the scalability in great detail in this dissertation.

1.2 Objective

As we have established the advantages of containerization technology, this dissertation aims to provide direction to anyone who wants to containerize their application as well as run it in production with features like scalability and cost-effectiveness. This aims to exhinit how to use Docker for containerization and Kubernetes for managing the containers over the cluster. We study autoscaling in great detail and go into existing algorithms for its implementation in the industry right now.
The main objective is to propose a forecasting method that can be used over historical data by anyone who wishes predict resources in advance, which can be used to autoscale the application as well as nodes in the cluster and. It can help greatly for reducing costs by effective use of cloud resources. We propose to collect Time Series Data and perform forecasting for the same.

1.3 Research Question

1.3.1 Limitation of Kubernetes Autoscaling Strategy

In the current automatic scaling service method for Kubernetes, the Horizontal Pod Autoscaler(HPA) will make sure that it is working in accordance with the targets which are predefined while defining Kubernetes resources(Pods) which run any application. After this, it keeps getting continuous updates about the status of Pod Utilization for resources like CPU percentage usage and memory usage to calculate resource utilization of replica set. This value is compared to the target value, and appropriate upscaling or downscaling of the pod is performed by the replication controller. This follows the following simple equation[33].
Here E is the number of replicas of pods which is desired for smooth function, CU is Current Utilization Percentage, TU is target value set at the start and CR means Current Replicas.

$$E = \left(\frac{CU}{TU} * CR \right)$$

Let's try to see initilization of Pod in phases [33]

TABLE 1.1: Phases of pod initialization by Horizontal Pod Autoscalar.

Phase	Time Taken	Description of the process
1	t1	Trigger HPA and calculate total number of replicas to be created and notify the Replication Controller
2	t2	The Controller received results and decide if up scaling or down scaling is required
3	t3	The scheduler detects creation or deletion of new pods and finds appropriate node to run it to.
4	t4	Kubelet starts the new resource downloads the images and initialize new pods into the node

Hence the total time is given by the equation:

$$t_{total} = \sum_{i=1}^{4} t_i$$

that time t_{total} has already elapsed. In this time a large number of requests may get queued in the server and will lead to waiting in response time. This may lead response in service requests which can be a problem in cases where it is not expected. For example, a billing server or a video streaming service. Therefore, there is an ***scope of improvement*** which can be done if we can predict the user requests beforehand.

1.3.2 Cost Aspect

Cloud Infrastructure is provided by Amazon Web Services which is the biggest cloud provider. Google Cloud is also a popular one in the market. Each cloud provider has various pricing strategies, but they provide two sorts of products: **on-demand instances** of computation resource and **pre-reserved instances** of resources. The former are virtual machines that only generate costs when utilized. With full authority, a cloud user attaches and removes an on-demand instance. In comparison, reserved instances are computing resources allocated and charged, with an upfront charge, for a certain time. The second type requires a level of commitment and can, in fact, be much cheaper for the user if effectively used for long term utilization. [6] Cloud service applications ought to schedule wisely to prevent needless expenditures. Reserved instances are valuable for expense-savings on the one hand. On the other side, they generate unnecessary expenses if allocated instances are underutilized. In particular, evaluating the usage of computing resources and allocating both reserved instances and on-demand instances is crucial for optimal cost. A good balance between these two forms of services allows for cost reduction and efficiency.

1.4 Delimitation

This dissertation proposes a methodology that can be used by any organization in need of scalability and cost optimization. However, it is not a dynamic framework or software which can be deployed to achieve the same task.
This dissertation works on a static dataset to predict the future values of resource utilization. It proposes a predictive method that can better the resource prediction and help better the Horizontal Pod Autoscaling implementation of Google Kubernetes being used in the industry today.
This also assumes the in those cases that where per minute prediction from data is

not possible due to noise, we can predict over hours and later spread it over minutes either by taking ratios over minutes and then seconds.

1.5 Related Works

A lot of work has been done in order to adopt containerization technology effectively. Also, the area of effective autoscaling is the area of great research over the years. Al-Haidari [19] studied the impact of setting the upper CPU utilization threshold and the size to which it can be scaled for the effective performance of cloud resources. Barret E. [21] tried to use temporal difference, reinforcement learning algorithm for better scaling strategies. Since then, a lot of research has been done in the effective prediction of resources in advance. Xie[5] tried triple exponential smoothing in historical data and tested it over the Google dataset. Anqi Zhao[cite 8] tried to bring the complexity of data itself into consideration, and he proposed a strategy of combining empirical modal decomposition as well as ARIMA models to predict the load on running pods in order to scale up or down the number according to this prediction.

In this dissertation, we try a way to obtain a realistic dataset from any web server over a period of time and try to apply Time Series methods including Moving Average, Exponential Smoothing, Holt's Models, ARIMA models and finally propose Facebook's open-source Prophet Model. We analyze this and later propose the use of Facebook's Prophet Model for prediction because of its performance and flexibility of use over any dataset.

Chapter 2

Background

In this chapter, the background needed to discuss the problem of resource prediction in aspect of containerization, is provided.
It should work as a comprehensive guide for anyone trying to containerize their application to run it on cloud.

2.1 Traditional Application (Monolithic Code)

Traditional applications built-in monolithic style of code, share a single codebase used by several developers, and if developers decide to implement or modify features, they have to make sure that their systems can continue to function during and after that change. Complexity arises as more technologies get introduced which restrict corporations' ability to evolve with new versions and apps[12]. Furthermore, as new releases are introduced into development, the whole range of systems is restarted, creating poor experiences into consumers who use them.
A monolithic implementation is often a single point of failure meaning the whole application falls if one service is down. Big organizations running and serving applications on the web have faced the above challenges, utilizing various processes,

techniques, and innovations which has led to the development of micro-services architecture. [12]

2.2 Microservices Architecture

The principle behind microservice style is breaking our application into smaller independent parts or services, each service isloated and indepenndent of one other and communicating to each other over API requests. Hence a group of services together constitute one application. For example, in Uber, seraching cab and billing are 2 different services.
They can be customized according to requirements, and can be implemented by a wholly automated pipeline for delivery. There is an absolute minimum of manual control of these systems, which can be written in multiple programming languages and using numerous techniques for data storage.
Advantages:

1. If a small part of application fails, the whole application might not be effected.

2. Smaller codebases and team objectives allow quicker implementations, which also helps to leverage Continuous Delivery benefits. [16]

3. As the smaller services are segregated, from the whole program program, one can more conveniently scale up the more important ones at the correct times. This can have an effect in cost reduction significantly.

4. Microservices offer versatility as there won't be too many questions over dependency since it is handled by container, so it is even simpler to scale back improvements. There is greater freedom, with fewer programming in play.

This form of architecture is being adopted by lot of companies, most of which see heavy traffics and need to avoid system failure. Amazon [3] and Netflix [13] are great examples.

2.3 Software Development Point of View

To create, update and maintain this ”smaller” services as a part of software at a quick pace is the challenge that software developers face. A lifecycle of software development involves the design, implementation, testing, and release cycles. Traditional Deployment strategies used to view each as a separate site which used to be time taking task . Now fully automated pipelines have taken their place. One needs to select the environment to deploy software into; production or test environment and deploy it using an automated pipeline, which is Continuous Integration/Continuous Delivery Pipeline (CI/CD) [17].

2.3.1 CI/CD Pipelines and DevOps

Any organization that wants to adopt a CI/CD pipeline needs to adopt Continous Integration(CI) and then Continous Delivery(CD). The purpose of this is to reduce the manual steps needed to deploy a piece of software in a production or any other environment.
Continuous Integration- It is a how modern developers collaborate. It is way of develop and integrate changes quickly without causing issues like system downtime. [20] Continuous Delivery means the ability to make changes in features or configuration and make it reach the end-user quickly and sustainably. [17] It requires very close collaboration between Development and Operation teams. The boundary between the two is fading as we have DevOps(Development and Operation) teams working in organizations that are moving to the cloud.

2.4 Containerization an Application: Docker

Docker is an open-source platform that runs applications and makes it easier to create and distribute the software. The application is packed as a ”container,” which

has all the dependencies (OS, packages, etc.). These containers tend to operate in an autonomous way over the Kernel of the operating system instead of running their own Kernel like in the case of Virtual Machines.
Containerization Technology has been around for decades now. However, the recent development of docker has made it possible to reach it a scale of use, which it never could before.
The application can be packed into lightweight containers, and these containers can be shipped anywhere by hosting in a registry so that anyone from any part of the world can run the application as a container by downloading its ”image” from the registry.
Docker containers are fundamentally different to VMs. They run over host machine’s kernel. They do not consume extra resources as much as VMs tend to do.

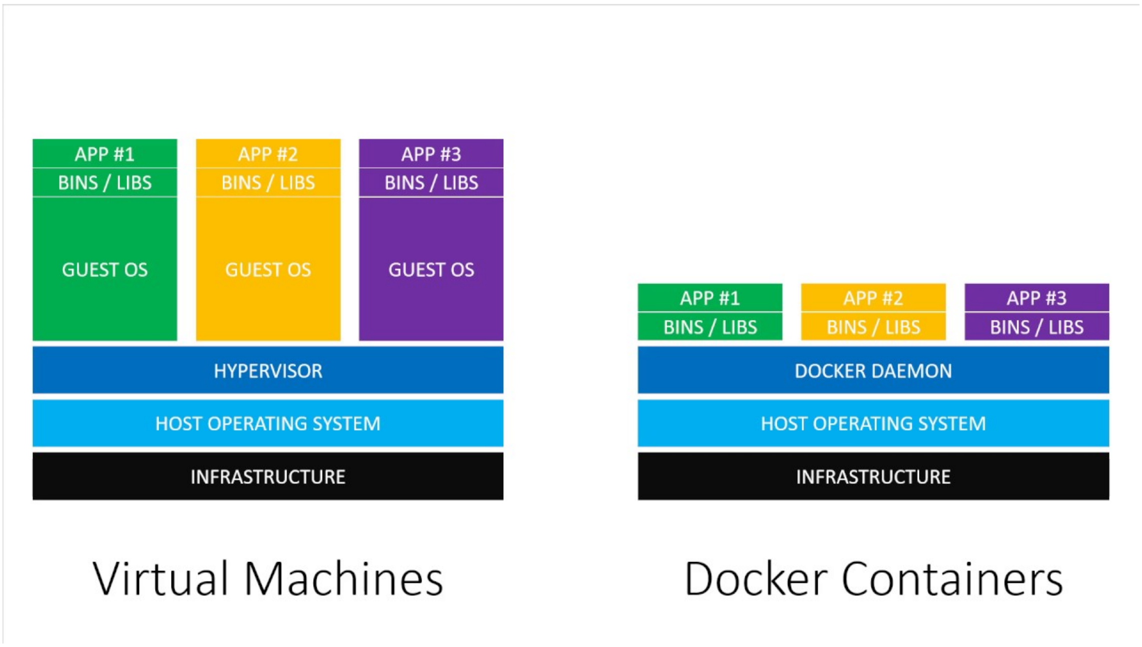


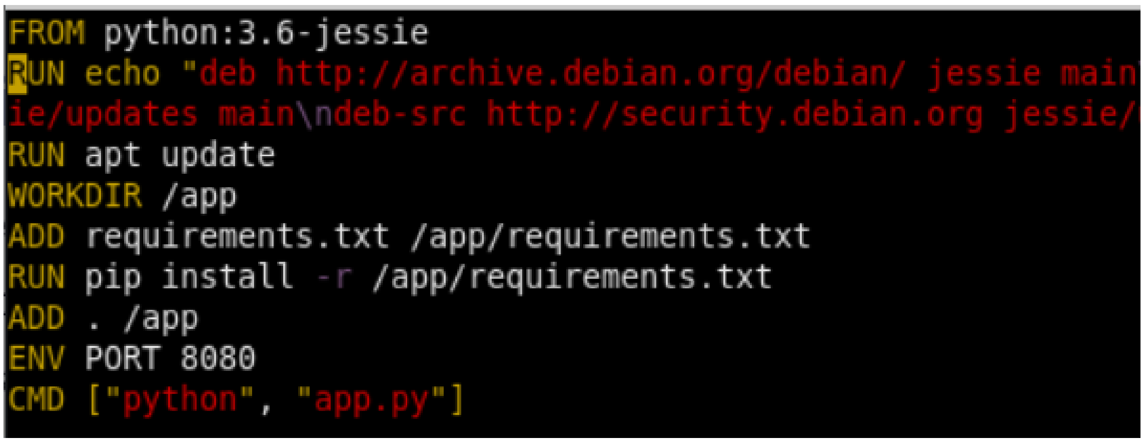
Figure 2.1: Virtual Images vs Docker Containers

Let’s try to understand a few components:

1. **Docker Client and Server:** The Docker server gets a request from the docker client, and it processes it. Docker provides us with complete RESTful(Representational state transfer) API as well as a client binary. Both the

server and client can run on the same machine, or we can use the client to request the server, which we choose to run in a remote machine.

2. **Docker Images:** A packed version of any application can be thought of as an image. To create a Docker image, we need to create a Dockerfile which has the concept of layering. Meaning we can start with any publicly available image as a base layer(usually OS) and build on top of it. The Docketfile has commands as shown in the example, which can perform tasks of putting different layers of required dependencies whenever required. After the creation of a Dockerfile, the client sends a "build" command to the Docker server, which creates an image based on the content of a Dockerfile.



```
FROM python:3.6-jessie
RUN echo "deb http://archive.debian.org/debian/ jessie main
ie/updates main\ndeb-src http://security.debian.org jessie/
RUN apt update
WORKDIR /app
ADD requirements.txt /app/requirements.txt
RUN pip install -r /app/requirements.txt
ADD . /app
ENV PORT 8080
CMD ["python", "app.py"]
```

FIGURE 2.2: An example of Dockerfile

3. **Shipping the image(Docker Registry):** Docker images are placed in registries. Images can be pushed and pulled from it. It is like a central hub for images for availability throughout the web. They can be classified into public or private registries. Docker Hub is one such public registry where anyone can pull images from or push their images. There is no need to create your file from scratch each time. Such registries are used by organizations that are aiming for continuous integration as it allows the availability of images irrespective of team or location.

4. **Docker Containers:** A Docker image in runtime is a docker container. It can be thought of as an independent instance running on the host with no knowledge of other instances or the fact that it is not a virtual machine itself but a container. Hence we can run multiple containers from the same image and balance the load to scale it.

2.5 Container Orchestration and Management

Any Container management software automates the creation, destruction, deployment, and scaling of containers. Each week, Google claims to launch more than 2 billion containers [9]. Such an extensive scale made Google develop tools for container management.
Based on its experience in working with Linux containers, Google launched Kubernetes as an open-source container management tool. Kubernetes has made its place in the industry as almost every big organization, as well as small startups, are shifting towards using it.

2.5.1 Kubernetes Architecture

Master Node: The master node is in charge of running the Kubernetes cluster. It is effectively the point of entry for all administrative processes. The master node hosts control processes that are necessary for the entire system. Kubernetes services run on the master node, and they are listed below [26]:

1. *API Server:* There is a server that allows external users to request Kubernetes using commands. We talk to API server using predefined commands.
2. *Controller Manager:* It is responsible for running these processes on Kubernetes master.

3. *Scheduler:*This service ensures that any new resource created in the cluster is scheduled somewhere in a node in the cluster.

4. *Etcd:* This service maintains a key-value pair, which helps all the nodes in the cluster to remain synchronized about the state of the cluster

Worker Nodes: Other than the master, which is used to schedule the processes, the worker nodes are actual machines that run the containers in their kernel. There can be multiple worker nodes associated with one master node. It has 2 main parts in it.

1. *Kubelet:* Kubelet is like the captain, which is in charge of its worker node. It listens to instructions from the master node and performs the required tasks in worker nodes.

2. *Kubeproxy:* This service ensures that every component running in our cluster is available. The controller manager receives instructions from scheduler and give instructions to kubelet. Scheduler is controlled by API Server directly which takes actions according to your requests.

2.5.2 Running Application(Pods)

There are many nodes in Kubernetes, and it manages all of them in the cluster. Kubernetes run containers on these hosts, and it the smallest unit of application management in Kubernetes. We call this a Pod. It is necessary to replicate and run multiple pods for scalability purposes. In pods, there can be more than one container making up an application. The storage required can be internal or as a pod extrernally. Kubernetes manages all such pods created (using YAML file); they are monitored and stopped or started whenever necessary.

2.5.3 Exposing your application(Service)

After having our application running as Kubernetes pods, we need to make it available for users to access. The Pod in Kubernetes is a mortal resource, and it can restart if required. Hence the IP address of a POD is not the best way to make it available as it might be changed when pod recreation happens. The abstract way of exposing a Pod is using another Kubernetes resource called service. [10]

There are different kinds of Services available in Kubernetes object.

ClusterIP: For those pods which only need communication by objects inside the cluster, we give service, an IP that is local to the cluster.
NodePort: For exposing our application to the outside world, we can use a publically available node and expose the service in one of its static ports. The service and hence the application is reachable to the external user using Node IP:Nodeport
LoadBalancer: These days, many cloud providers give an option to use a load balancer service which automatically generates a cloud provider's IP address so that the service is reachable to the outside world.

2.5.4 Scalability and Autoscaling

Once we start using Kubernetes to run our application, we can modify and ship the changes faster than ever before. [10] But what happens when the app becomes even more successful than we had anticipated. Suppose we have a business which sees a lot of traffic during the day but low traffic in the night. We would want our services to dynamically scale automatically instead of needing manual help every time it's needed.
Kubernetes allow pod autoscaling based on different metrics. Two of the most widely used ones are:

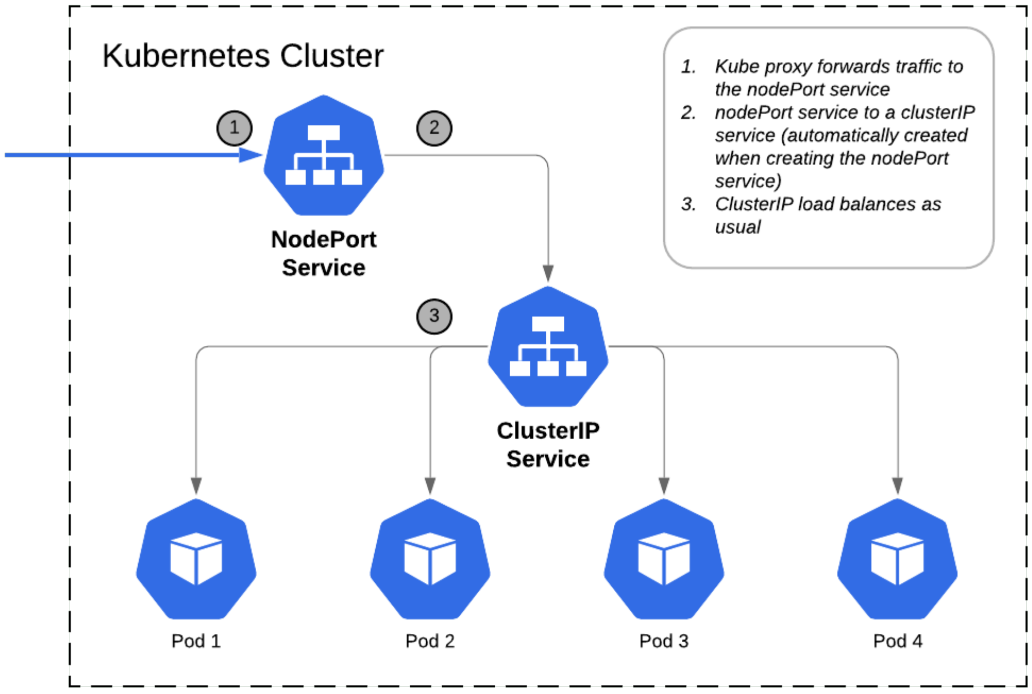

Kubernetes Cluster
1
2
3
1. Kube proxy forwards traffic to the nodePort service
2. nodePort service to a clusterIP service (automatically created when creating the nodePort service)
3. ClusterIP load balances as usual
NodePort Service
ClusterIP Service
Pod 1
Pod 2
Pod 3
Pod 4

FIGURE 2.3: Exposing Pods using Services in Kubernetes [7]

2.5.4.1 Vertical Pod Autoscaling

Vertical pod autoscaling (VPA) allows us to not worry about what values of CPU utilization or memory should we assign in our pod. The autoscaler will prescribe CPU and memory requests as well as limit values. When the assigned limit gets exhausted, the VPA can automatically scale the limits of resources and helps pod function effectively.

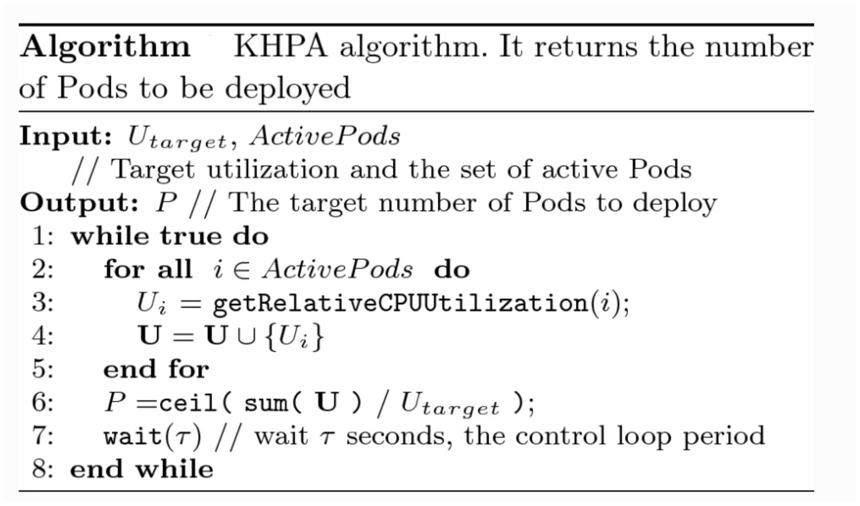
There are many benefits to using VPA. Because the pod only uses what it needs, the nodes in our cluster are used efficiently.

Pods have enough resources available all the time, and we don't have to worry about running tests to know the "correct" value of CPU and memory utilization. Hence maintenance time is significantly reduced.

Let's dive deeper into Horizontal Pod Autoscaling.

2.6 Horizontal Pod Autoscaling

Since we want to keep the entities like Pods lightweight, we have the option of using multiple replicas of the same instead of increasing the size of Pod, as we do in VPA. Kubernetes allow us an option to deploy a Horizontal Pod Autoscaler(HPA) for our application which takes the "correct" values and limit values of CPU and memory utilization beforehand and tries to maintain a target percentage utilization throughout all the replicas of the application pods running.
It deploys a service called Metrics Server, which periodically checks for the current consumption and notifies HPA, which decides if it needs to change the number of replicas and notify the replication controller about the same [10]
For ease of study, we assume that each Pod has a single container. The algorithm currently used is as follows:



Algorithm KHPA algorithm. It returns the number of Pods to be deployed

```
Input: $U_{target}$, $ActivePods$
    // Target utilization and the set of active Pods
Output: $P$ // The target number of Pods to deploy
1: while true do
2:    for all $i \in ActivePods$ do
3:       $U_i$ = getRelativeCPUUtilization($i$);
4:       $\mathbf{U} = \mathbf{U} \cup \{U_i\}$
5:    end for
6:    $P$ =ceil( sum( $\mathbf{U}$ ) / $U_{target}$ );
7:    wait($\tau$) // wait $\tau$ seconds, the control loop period
8: end while
```

FIGURE 2.4: HPA current Autoscaling Algorithm [15]

HPA takes as input the target CPU Utilization and the current number of Pods in the cluster. Then using metrics server, it checks for the status of utilization(every 30 seconds) and calculates the new number of replicas required to maintain the desired CPU Utilization.

2.6.1 Limitation in HPA(our area of focus)

As discussed in Introduction chapter 1.5, this algorithm has a few limitations.
From the time it takes for the metrics server to run (which is 30 seconds in the worst case) to Pod initialization request triggered by HPA, t_{total} amount of time passes. Please refer to the table above. 1.1
We use Time Series Forecasting to predict the load in the server in advance so that the replication controller can be notified in advance to maintain a certain number of Pods running for the next minute or hour. We can find lower and upper values of prediction, and we can juggle between focus given to maximum availability or cost optimization by choosing QoS or ERU, respectively.
Next, we focus on the time series forecasting and the results we have derived from them.

Chapter 3

Method Proposal and Dataset Generation

3.1 Standard Inferences from Autoscaling Used Cases

As we try to find a dataset to apply our time series analysis, we need to be aware of the feasibility of this prediction in newly generated datasets. So we try and see two use cases where companies have adopted autoscaling and try to infer the commonality in their used case scenario and any new server where we want to apply this prediction.

1. **Netflix:**
   Netflix streams in billions of devices throughout the planet and becoming a thing that is in everyone's pocket.
   "On-demand video provider Netflix documented their use of autoscaling with Amazon Web Services to meet their highly variable consumer needs. They

found that aggressive scaling up and delayed and cautious scaling down served their goals of up time and responsiveness best"-[2]
Netflix adopted a microservices architecture very early [13] and is part of the reason for it to be a market leader. Any streaming service will obviously get more traffic during evening hours as compared to morning hours in the usual days. Also, with the introduction of new content, the number of users is supposed to grow exponentially. We can also assume more traffic on weekends as compared to weekdays.
We can infer three things from this: An increasing trend (exponential growth of users), cyclic pattern(weekly or yearly) as well as a few outliers (holidays or day new shows are introduced)

2. **Facebook:** According to a blog post by Facebook [11], "Facebook reported a 27 percent decline in energy use for low traffic hours (around midnight) and a 10-15 percent decline in energy use over the typical 24-hour cycle"-[2] This was achieved using historical data and analyzing the pattern of Facebook traffic during different hours of the day. Ofcourse as Facebook was growing (trend), they were able to utilize patterns in hourly data for prediction (cyclic pattern).

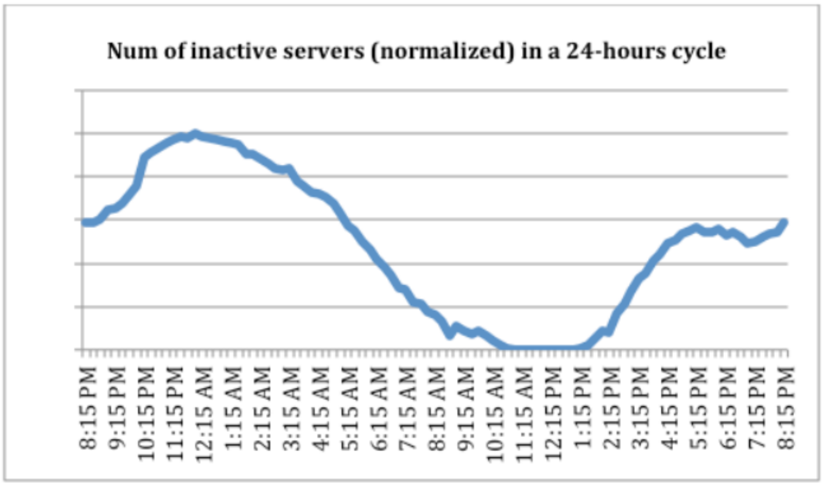


Figure 3.1: "Normalized number of servers in a web cluster put into inactive mode by Autoscale during a 24-hour window"- [11]

3.2 Common Hypothesis Generation

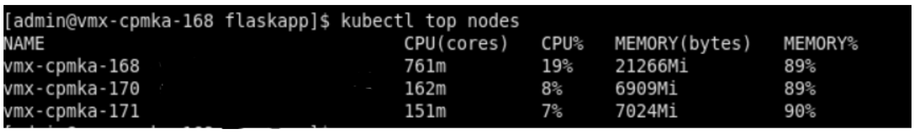
We can see in most of the used cases; there is going to be a trend as well as cyclic nature in the pattern of data. They together are components of a time series itself. We cannot apply simple linear regression or similar methods because here, the data is dependent on time, which violates the underlying assumption of the regression model. Along with these advantages, Time Series Analysis also gives us various methods to take trend, seasonality as well as handling the noise in the data.
We try to generate a common Hypothesis, i.e., we try to list the factors which will affect the observations. We can safely assume the following factors:

a) The load in the server will go up as the years pass by.
b) Traffic may fluctuate periodically, i.e., on a particular group of months or weeks, etc.
c) Traffic will be high during the peak hours(day or night depending on the business)
d) There are going to be certain days which will be outliers in the observation.

3.3 Datset Generation from any server

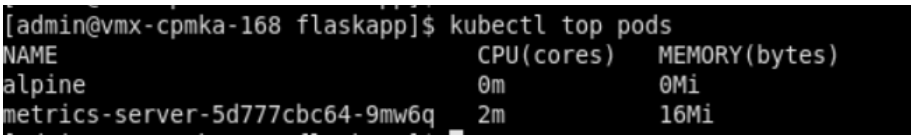
There is a lot of data we can collect from the cluster running our server.

We can generate datasets by using the most widely used open-source systems monitoring called Prometheus, which is usually already present in any Kubernetes cluster working in production.
According to prometheus website [14], "Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength".



```
[admin@vmx-cpmka-168 flaskapp]$ kubectl top nodes
NAME                               CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
vmx-cpmka-168                      761m         19%    21266Mi         89%
vmx-cpmka-170                      162m         8%     6909Mi          89%
vmx-cpmka-171                      151m         7%     7024Mi          90%
```

(a) Node Usage Stats



```
[admin@vmx-cpmka-168 flaskapp]$ kubectl top pods
NAME                               CPU(cores)   MEMORY(bytes)
alpine                             0m           0Mi
metrics-server-5d777cbc64-9mw6q    2m           16Mi
```

(b) Pod Usage Stats

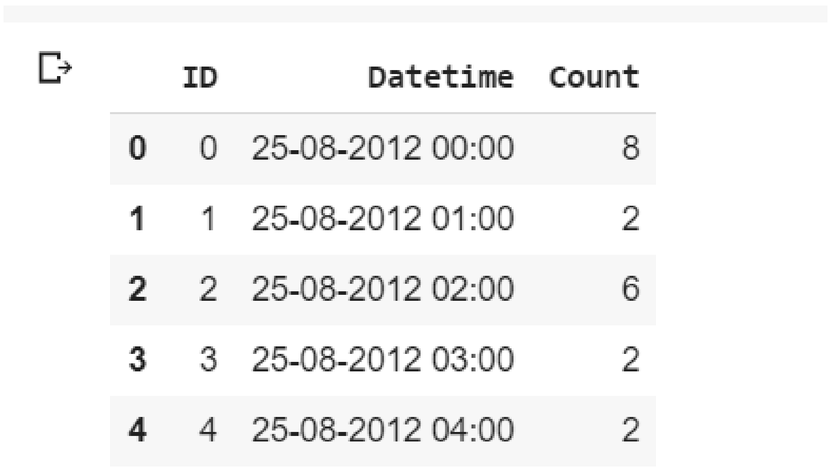
Figure 3.2: The Usage metrics being collected by metrics server in Nodes in the cluster and Pods running on them.

We can ship the metrics in any form necessary and feed it to InfluxDB, which is an open-source time-series database. It takes the metrics from Prometheus and creates a time-series database with fields like CPU Utilization percentage in Node, Number of pods in cluster, etc. along with the timestamp.
According to Matt Asay [18], this new practice of recording everything that is happening in the system, along with the timestamp, is very powerful. It helps one quantify the progress: to understand whether something has changed in the past and why, and also to track whether something in the current is changing, to foresee any event that may occur in the future.

3.4 Dataset

We use a similar kind of dataset for our analysis. We are provided with a dataset that has the total number of traffic on a train per hour, every day for about two years. Since the dataset is very similar in terms factors we discussed above, we use this dataset, apply Time Series in it and validate it after splitting it 80:20.

The dataset has two columns having TimeStamp in DateTime column in dd-mm-yyyy H:M format and count of visits for each timestamp. The dataset consists of 18288 rows.



	ID	Datetime	Count
0	0	25-08-2012 00:00	8
1	1	25-08-2012 01:00	2
2	2	25-08-2012 02:00	6
3	3	25-08-2012 03:00	2
4	4	25-08-2012 04:00	2

FIGURE 3.3: The content of time-series dataset used for analysis

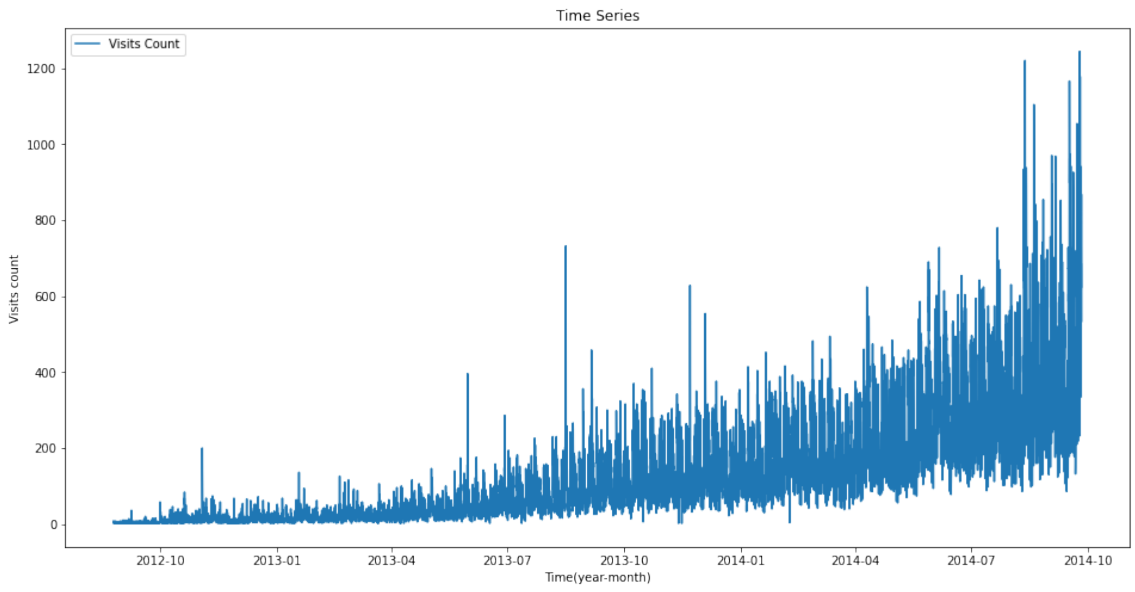


FIGURE 3.4: Dataset plotted over entire time

3.5 Evaluation Metric

We have used the Root Mean Squared Error(RMSE) as a performance metric. The reason for choosing it over a strategy like Mean Absolute Error(MSE) because we do not desire to have large errors. RMSE uses the difference is squared terms; hence errors will reflect better in RMSE. RMSE is the difference between predicted and observed values first squared and then taken the mean of and lastly taken the square root of.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(\frac{p_i - o_i}{n}\Big)^2}$$

where p_i, o_i and n means predicted, observed and number of observations respectively.

Another intangible performance metric to consider is **ease of implementation** and adaptability to other datasets. We will choose a method that can better fit the factors we have considered in our hypothesis.

Chapter 4

Methodology and Implementation

All the domains use time-series from finance to economics as it a great way to measure the change occurring in the state of a system over time. The reason for this is that it can be used to rigorously find patterns in the state with respect to time so that future changes can be predicted.
Definition: A sequence of observations, taken over equally spaced intervals of time. The observations collected are in proper chronological order.
If the observations of only one variable are recorded and analyzed, the time series is Univariate. If more than one observations are being analyzed, we call it multivariate. The time-series that we have worked upon is a univariate time series. The dataset is also an example of a discrete-time series as the observations are recorded in an equally spaced period of one hour.

Components:
Any time series consists of 4 components:
a) Trend: It is the movement of the observations in the time series in the long term, which tells whether the series is going upward, downward or it is getting stagnant. For example, in our cases, the number of visits to the train(and reservation website) should go uphill as years pass. It is an upward trend.
b) Seasonality: The patterns or fluctuations in the observations that repeat in one

year time. We will observe in our case that visits peak in certain months every year.
c) Cyclic Variations: The fluctuations in the observed pattern happening in a medium-term to longer frame of time. The cyclic variations durations can last to longer periods.
d) Residue: The irregularity in the data after taking the above factors out is called the residue.

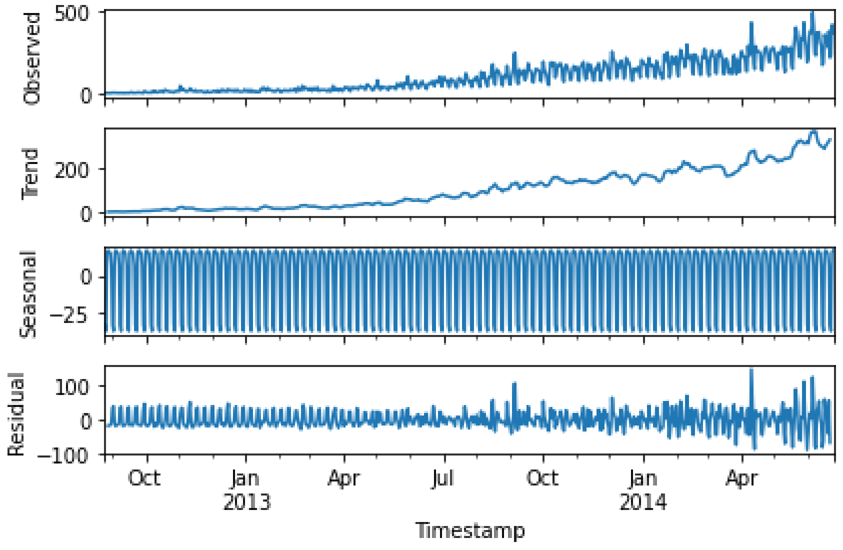


FIGURE 4.1: Components of time-series for our dataset

The Time Series prediction techniques can be viewed into two categories. The first type of approaches try to predict the future directly based on past few values while the second type tries to find a mathematical model that can fit the pattern. Moving Average Smoothing, Exponential Smoothing, ARIMA are examples of the first type. The Facebook Prophet algorithm we use is an example of the second category.

4.1 Preprocessing the dataset

We can group the time series by year, month, days or hours as needed to analyze it.

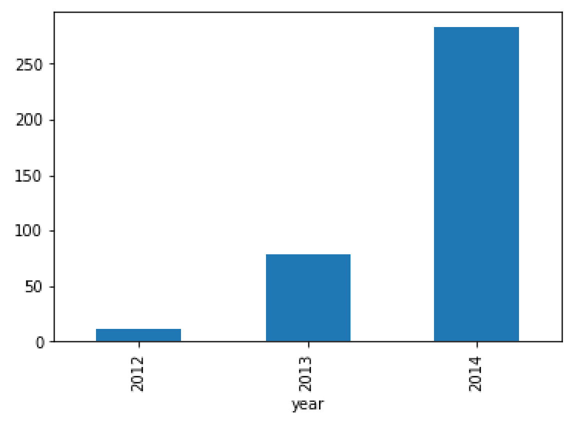


FIGURE 4.2: we can see an exponential increasing trend in time-series

As we saw in 3.4, the hourly data-set contains a lot of noise so we try to re-sample it by aggregating over days, weeks and months. The approach that we adopted is that we make predictions over time series in a one-day interval, which is created after we aggregate the observations for 24 hours of each day. We store the ratios of visits made per hour so that we can finally distribute the predicted observations over the one-hour interval in the same proportion. Next, we focus on the techniques and try to see if they are suitable for the dataset of our type.

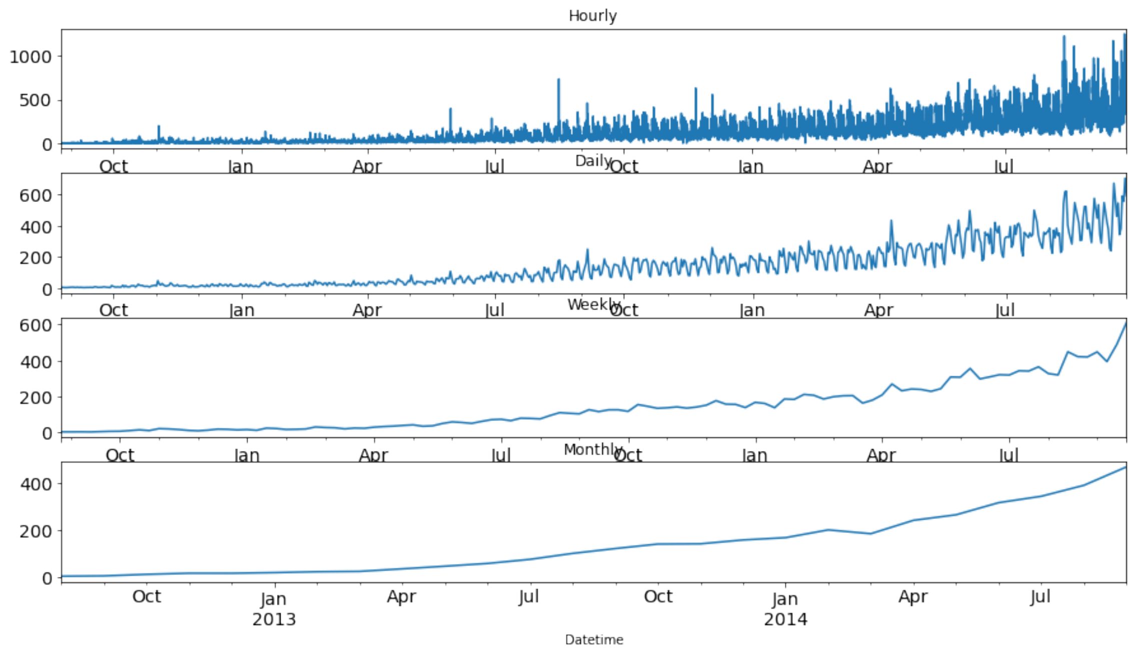


FIGURE 4.3: The resampled time series for days, weeks and months

4.2 Moving Average Smoothing Method

The moving average is one of the simplest methods for forecasting. As we move ahead in time in our observation, we want a metric that can take into account past values up to that point and also avoid getting lost into the noise that the data may have. The best way to do that is by taking an average as we move along. A simple Moving Average simply takes arithmetic means of last q observations and use it for the next forecast.

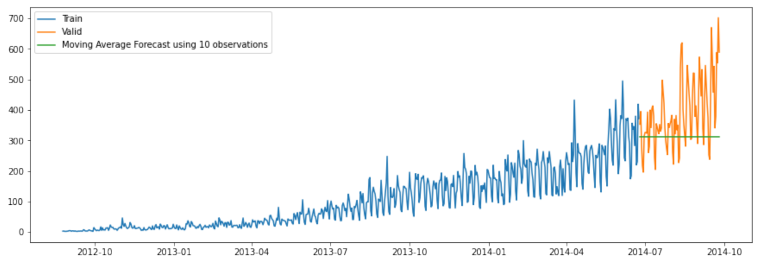
It can be useful in our cases if we have reached a stage in our server where the trend is almost stable. In that case, the next values depend a lot on values of past few observations, and taking an average of those might give good results. Since we can change the order of moving average, which in our observation means the days over which we are making observations, we can have some success.
We apply the moving average into our observations and validate it against the actual observations. We take period of 10, 20 and 50 days for smoothing.

The RMSE value collected for the 3 cases are as follows:
10 days- 134.3
20 days-130.44
50 days- 144.19

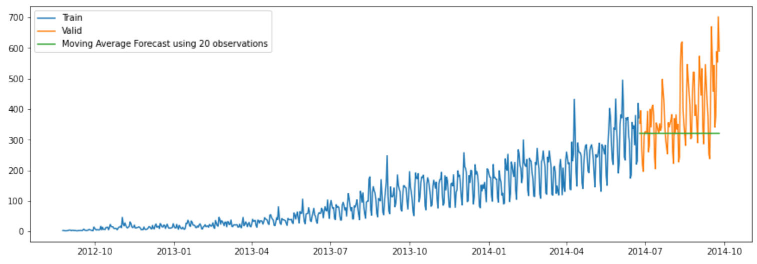
Is is not a very bad forecast but we can definitely do better.

4.3 Exponential Smoothing

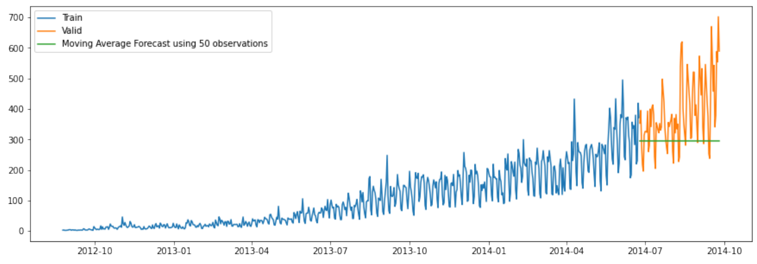
The Simple Exponential Smoothing (SES) method is generally used to forecast time series that has no definitive trend or seasonality. Just like we do in case of Moving Average, this method also considers past values to predict a future one but where it



(a) Moving Average smoothing: 10 days



(b) Moving Average smoothing: 20 days



(c) Moving Average smoothing: 50 days

FIGURE 4.4: Moving Average Smoothing applied over different orders.

differs is that instead of taking equal weights of a fixed number of observations, it takes all the observations of past in account, but it gives exponentially decreasing weights to observations as they move further in the past. Meaning the most recent observation gets most weight.

In SES, the following equation come into play,[23] which is used calculate the current smoothed value based on current observation and past smoothed value. We use this recursively.

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots \tag{4.1}$$

where $0 \leq \alpha \leq 1$ is the parameter used for smoothing in the time series $y_1, \ldots, y_T$. After a considerable amount of time, the trend is supposed to decrease and similar to Moving Average method, the past few values decide the future values with more correlation. SES is supposed to work better than Moving Average Method since it is not bound to distribute equal weights.

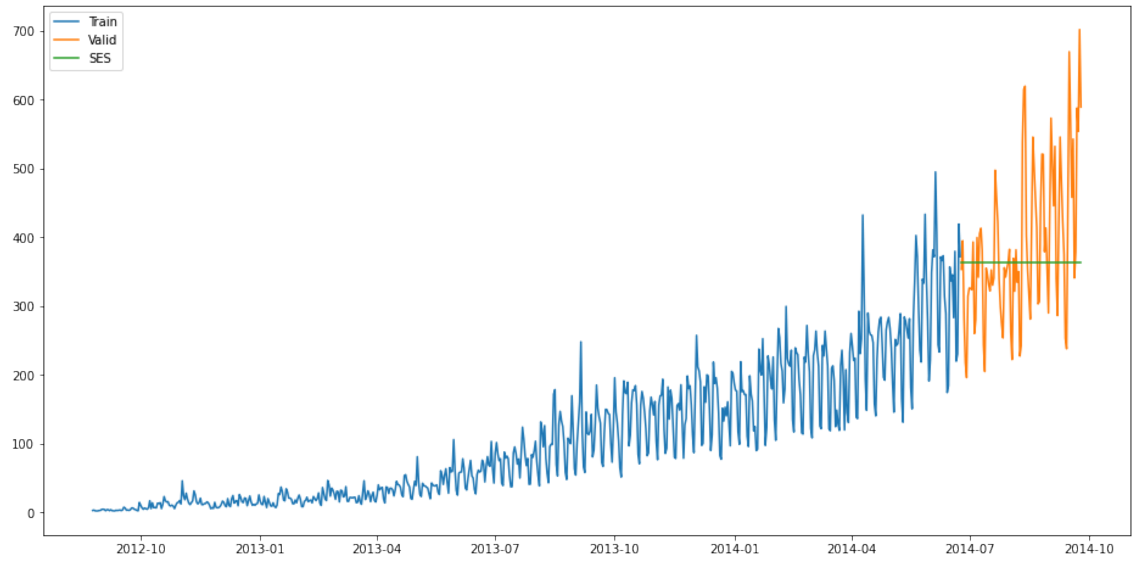


FIGURE 4.5: Prediction using Exponential Smoothing

As expected this method works better than Moving Average Smoothing with a RMS value of 113.43.

4.4 Holt's Model

4.4.1 Holt's Trend Model

In the above two methods, we were not able to take the trend into consideration. Holt's trend model is a popular method for forecasting observations with a trend. How it achieves the same is by using three equations that help us predict future values. [28][8]

$$\begin{aligned}
\text{Forecast equation} \qquad & \hat{y}_{t+h|t} = \ell_t + hb_t \\
\text{Level equation} \qquad & \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
\text{Trend equation} \qquad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1},
\end{aligned}$$

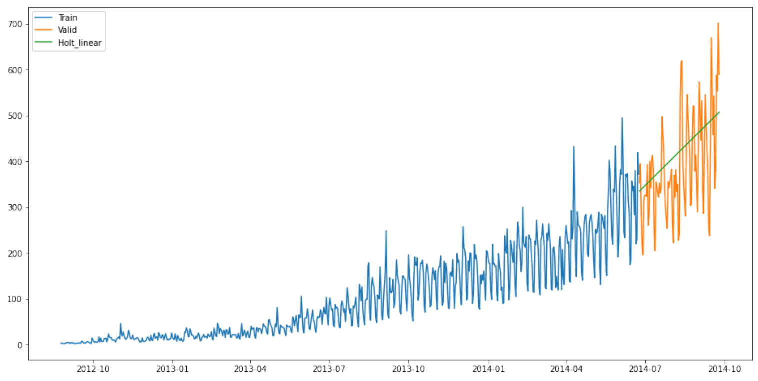
where ℓ_t is level of the equation at observation time t, b_t denotes the approximate trend which can be captured with a slope, α between 0 and 1 is a smoothing parameter and β^* is parameter the used for smoothing trend. [8]
As we already saw into the decomposed components figure above 4.1, our observations follow a trend. So we try to do forecasting using this method.

4.4.2 Holt-Winter's Seasonal Method

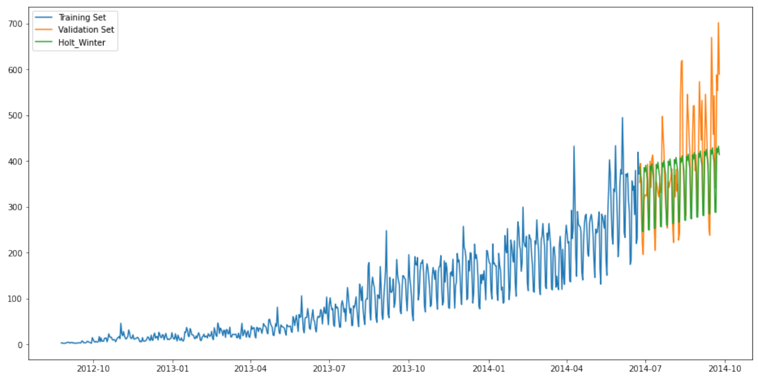
After successfully integrating trend into the forecasting, we try to use Holt-Winter's Model, which also include the effect of seasonality in the forecasting. We need to specify the seasonality frequency beforehand, which is 4 for quarterly or 12 for monthly, and so on. We have used the additive method which uses three following smoothing equations, to describe the components of level, trend and seasonality [25]

$$\begin{aligned}
\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\
\ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}$$

where k is Greatest Integer smaller than (h-1)/m. It just ensures that seasonality from only past one year is considered. $\alpha, \beta^* and \gamma$ are smoothing factors for level, trend and seasonality respectively. The RMSE results obtained are as follows:



(a) Holt's Trend Model



(b) Holt's Winter Model

FIGURE 4.6: Holt's Models

Holt's Trend Model: 100.20

Holt's Winter Model: 82.7

4.5 ARIMA Model for forecasting

The other type of method that works complementary with smoothing methods is the ARIMA Method. While the former method is formulated to work on-trend and seasonality of the data, the latter one tries to do forecasting using autocorrelations in the function.

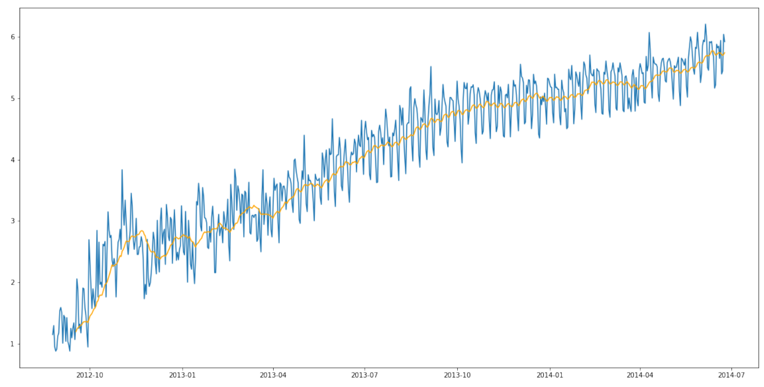
4.5.1 Stationary Time Series

A time series is called stationary if it's properties are independent of the time in which observations are based upon. [22]
This means that the trend and seasonality we have been trying to model makes a time series non-stationary. We cannot have seasonality, but in some cases, the cyclic time series is stationary because the cycles do not follow a fixed pattern, so we cannot predict an observation based on time. Because we have no predictable pattern, a stationary time series is usually horizontal, and it has constant variance roughly.

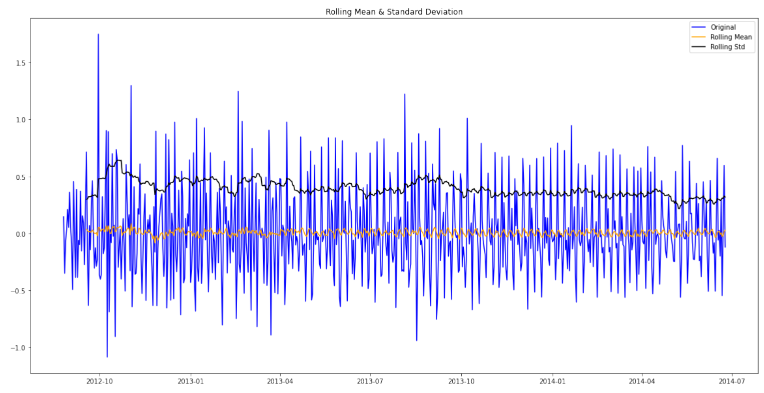
We can make a non-stationary time series, stationary using something called Differencing. We take the differences in consecutive terms in observations and try to get our curve almost horizontal. For the time series with a large number of outliers(significant variance), we try to stabilize the data and also ensure we don't lose any information. We usually take logs of observation, which penalizes big values more than small values. We can see the graph in 4.5.1

Dickey-Fuller Test:
How to know if our time series is stationary? This test assumes the null hypothesis that the series is non-stationary and gives us a bunch of critical values and test statistics. If our test statistics exceed the critcal value, we accept the hypothesis and state series to be non-stationary. Then we proceed to try and make it stationary.

ARIMA model uses both Autoregressive models as well as Moving Average models.
a) **Autoregressive model:**
Like we do in a regression model, we try to predict a variable as a linear combination of other independent variables. Here we try to predict the next term of observation as a linear combination of the last few terms. Thus, an autoregressive model of order



(a) Increasing Trend in Time-Series



(b) Time-Series after differencing

FIGURE 4.7: Making Time-Series Stationary

p can be written as [8]

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

where ε_t is the noise term.

b) **Moving Average:**

Instead of using past values of the observation to predict future values, as we do in regression, the Moving Average model tries to predict future values by using the errors in past predictions and their linear combination. The equation looks like [8]

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

where ε_t, θ are the noise term and error term respectively.

When we combine the moving average method with autoregressive method, it is called ARIMA. It also has a term for number of time we perform differencing. [8]

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{4.2}$$

where y'_t is the series after differencing. We are using linear combination of both past values(AR) and past errors(MA).
This is called ARIMA(p,d,q) model.
where p=order of autoregressive part, d =number of times differencing is done to make the model stationary and q=order for moving average part.

To get correct values of p,d, and q, we plot the Auto Correlation Function(ACF) and Partial Auto-Correlation function. ACF tells us about the autocorrelation of the observed values with its lagged values. PACF helps us get partial autocorrelations, meaning the correlation between a particular past observation and current observation after removing the effects of all those observations that lie in between. However, if both values of p and q are positive, the plots might not give the best values for p and q.
In our case, we try to apply AR and MA models separately, keeping q=0 in the first case and p=0 in the second case.
After applying both, we find that parameters (2,1,2) describe our model the best.
We try to fit the prediction to thw whole dataset and found the following fit: see figure 4.8 We achieve the RMSE value of 43.2 for next 93 days in validation set. The value is supposed to go up if we increase size of validation set.

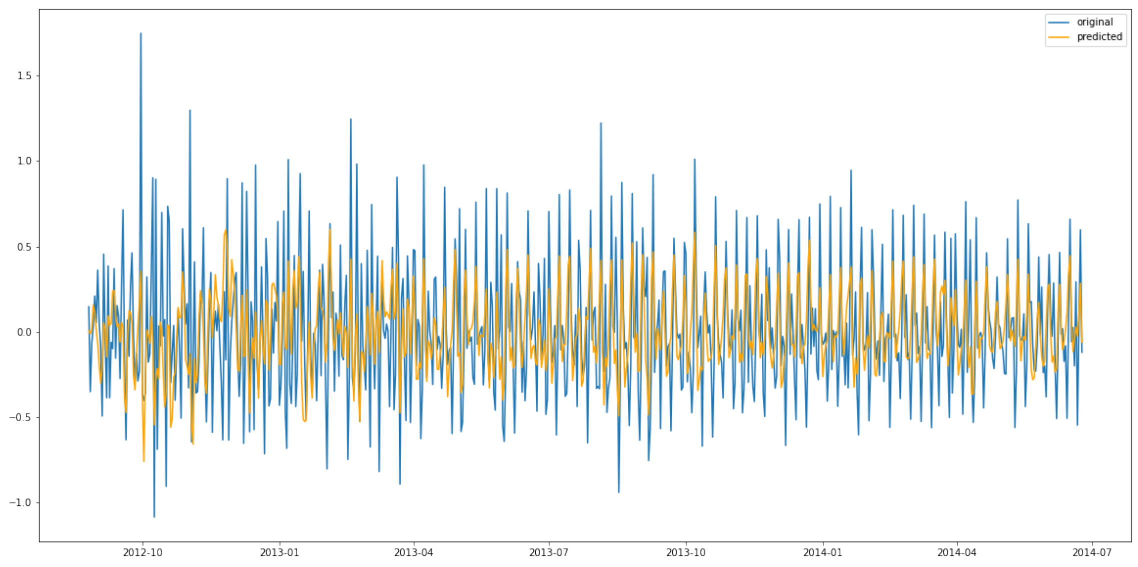


FIGURE 4.8: ARIMA Model over whole dataset

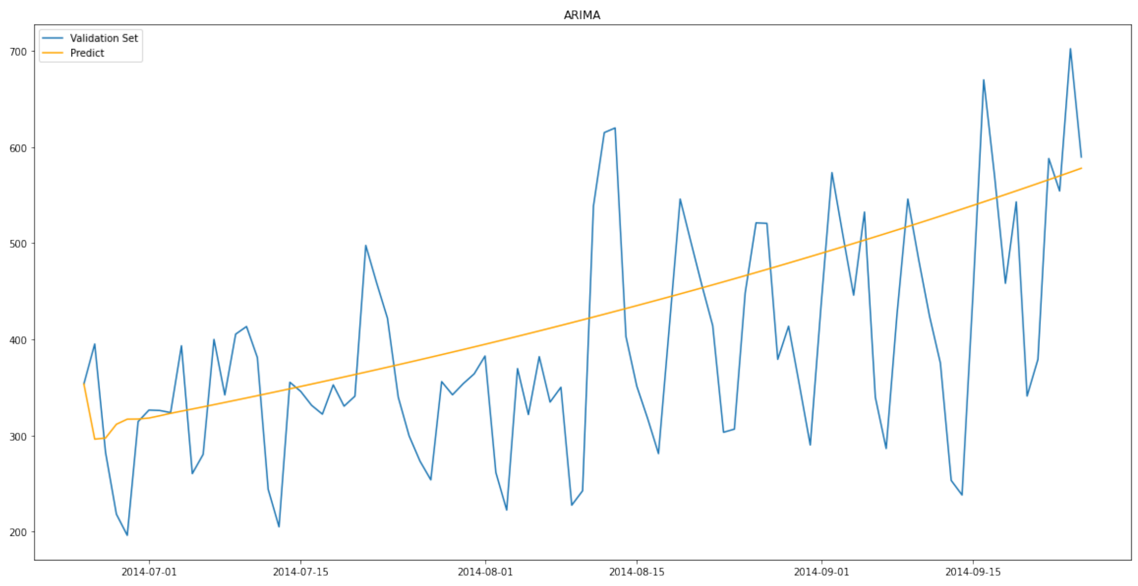


FIGURE 4.9: ARIMA forecast over validation curve

4.6 Facebook Prophet Model

Forecasting is used in every part of the industry, yet it is tough to single out a single model that can fit the needs of most analysts in the industry. We need a good grasp of time series analysis to be able to forecast different kinds of observations, which are not fixed in nature. Even our method in this dissertation excepts different kinds of datasets to be obtained with different patterns.
Taking all these into consideration, there is a need for a flexible model that could forecast at scale and could be easily tweaked by analysts according to their own needs. Especially when the historical data collected is very large, it's important that the machine does most of the tasks with less human intervention.

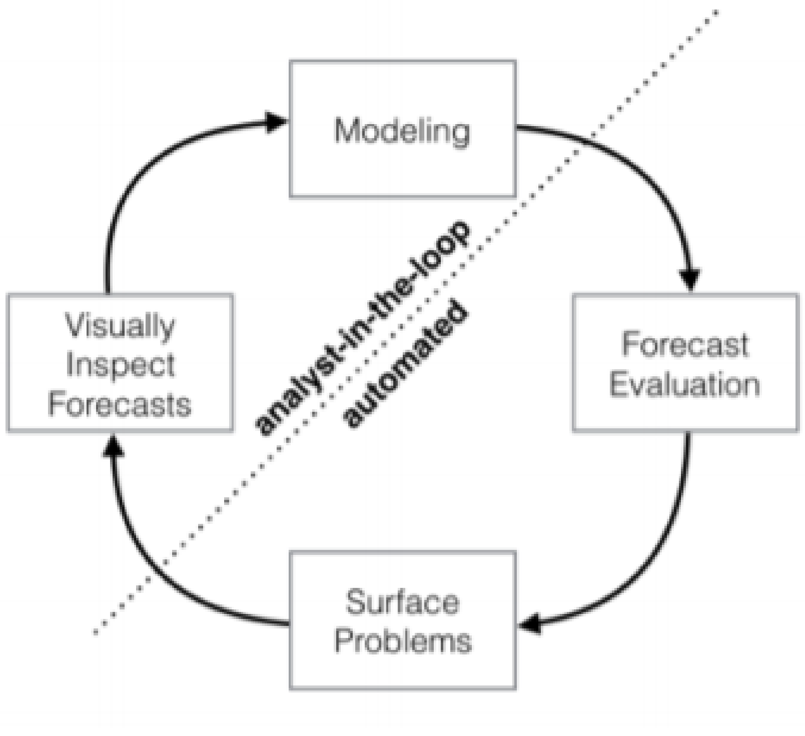


FIGURE 4.10: Keeping Analysts in loop for a forecasting method [30]

Facebook introduced a Prophet Model, which help us deal with all the commonalities in the dataset, as we discussed in previous sections. Weekly to yearly cycles, dips, or rise at certain days of the year, etc. The best thing this model does is that we can adjust it for these factors intuitively without knowing the underlying architecture of how this performs. Facebook open-sourced this model.
The Prophet intuitively is an additive model consisting of three main components: trend, seasonality, and holidays(outliers).

$$y(t) = g(t) + s(t) + h(t) + e(t)$$

g(t) part in the equation tries to fit the trends that observations follow.
s(t) part takes care of seasonality.
h(t) part takes care of sudden outliers or holidays that should be specially monitored by the model.
e(t) is noise or residual part.

Unlike predicting the very next value, the prophet model tries to fit the observation pattern according to the above equations. The fitting is very quick and the model is very adaptable.
As Sean Taylor analyzed in his paper [30], the prophet works superior to most models.4.11

We move ahead and tried to forecast our data using prophet model. see figure 4.12

When we try to fir this into our validation set, we see a very pretty picture: see figure 4.13

As discussed earlier, we focus on finding the right balance between Quality of Service(QoS) and Effective Resource Utilization(ERU).

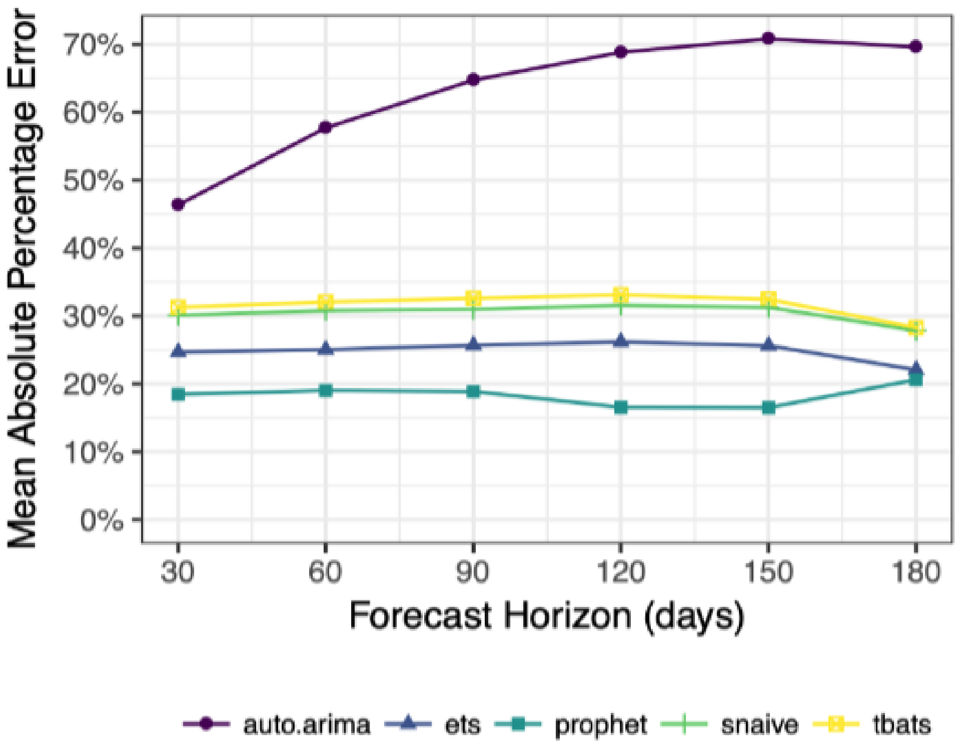


FIGURE 4.11: Comparison of various time series models [30]

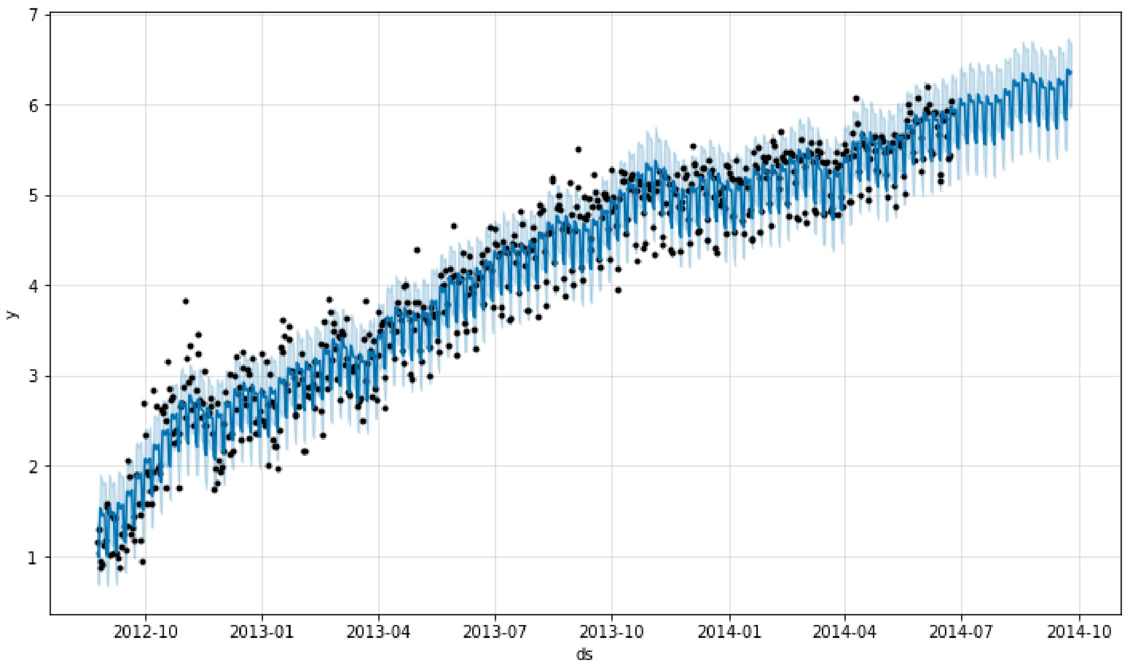


FIGURE 4.12: Prophet Model forecast over our dataset

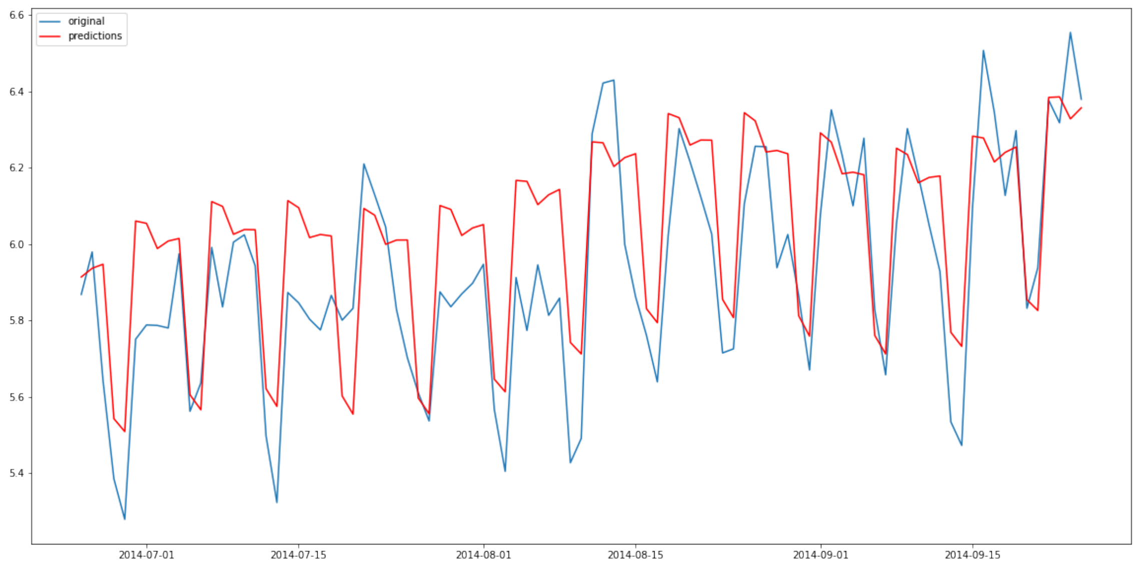


FIGURE 4.13: Prophet Model forecast vs validation set

We can try to predict the upper limit and lower limit after scaling the observations back to the original scale, by taking exponent. We observe that the original curve lies perfectly between the upper and lower limit. see figure 4.14

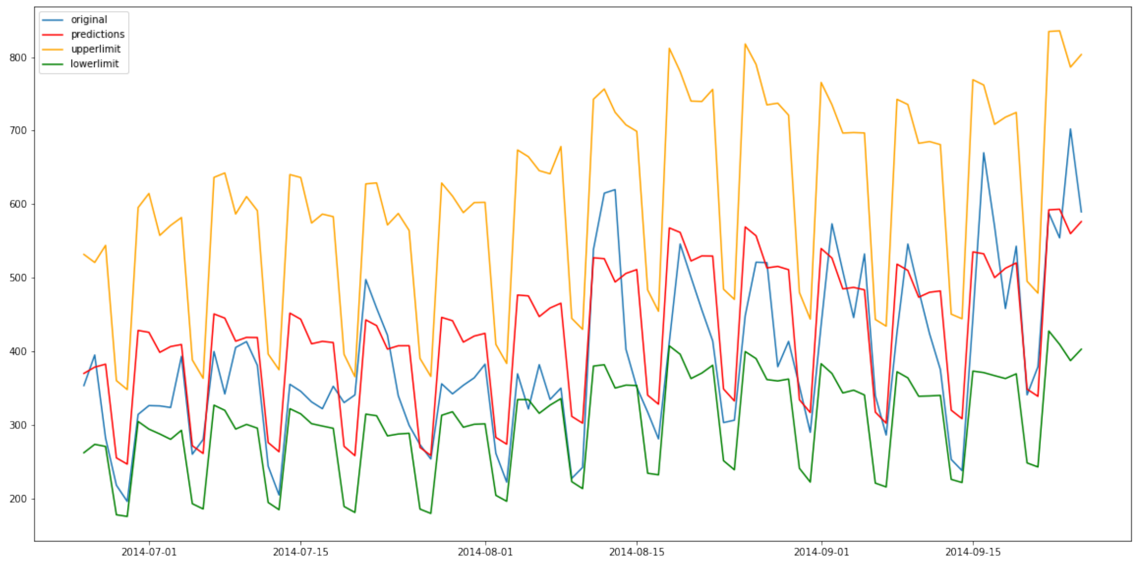


FIGURE 4.14: Prophet Model forecast vs validation set with upper and lower limits

We get an RMSE value of 74 which is surprisingly great considering how little we had to tweak with the model.

Chapter 5

Conclusion

We try to conclude a couple of things from this dissertation. The first part focused upon the shift happening in the industry to adopt containerization as a tool for running web servers and applications on the cloud. (see this 1.1.2)
Through various factors like ease of implementation and its lightweight, we observe how it is a desirable choice for deploying application servers. It helps us utilize cloud resources like CPU or memory efficiently, reducing the cost of buying those resources from the cloud.

In the second part, we observe the problem the current resource autoscaling algorithm being used in the industry has. (see this 2.6.1) Another non-trivial problem we observed was the cost of buying resources from the cloud without having an estimated idea of resources that we require in advance. 1.3.2

We proposed resource prediction in advance to solve the above two problems. We defined the use of sophisticated open-source tools to collect time-series data from a running web server.(see this 3.3)
Finally, we analyzed one dataset of the same nature and tried to use simple models

that could be extended over other datasets without much added manipulation.

Here is the list of methods we used:

1. Moving Average Smoothing and Exponential Smoothing:

These methods fit poorly for our observations because they were unable to capture the upward trend in data. However, for the web-servers which have been functioning for a long period of time and have gotten stagnant in terms of the number of users, they might be useful.

2. Holt's Models:

We tried to take the trend as well as seasonality into consideration using Holt's and Holt's Winter Model. We see them fitting better for our dataset as compared to previous models. However, there was a major drawback when we tried to fit seasonality in Holt's Winter model. It needed some time to fit the correct value of seasonality frequency. (see this 4.6(b))

3. ARIMA Method:

ARIMA method is actually a great method since it does not focus on seasonality directly but used autoregressive and moving average approaches for forecasting. We had to make the series stationary.

Although it gave us good prediction results with RMSE less than 10 percent, it also took the most time to implement.

Finding the correct values for parameters in the ARIMA model is not a hard task, but it needs human intervention each time to do it. And our focus is to find a model that makes the machine do most of the work.

4. Facebook Prophet Model: Facebook's Open-Source Prophet Model [30] is an additive model that takes into consideration many linear and non-linear functions of time and tries to fit the trend, periodic fluctuations, and special outliers as components.
The equation is simply

$$y = g + s + h + e$$

where g is trend component, s is periodic changes that can be weekly, monthly or yearly, h accounts for special points in time series, and e is for other idiosyncratic changes. They are all functions of time.
This kind of model defines an entity like a web-server greatly. Be it a railway reservation website or a shopping website, they will have trends, periodicity, and great rush on specific days like holidays.
We try to fit our data, and from figure (see this 4.13) we can see how well it fits our needs. The RMSE value is around 10 percent, and it captures any sudden spike with great similarity to actual values. In terms of accuracy for upper and lower limits, it is a clear winner. (see this 4.14)
It is also a model that required the least human effort and can be of great use when we try to make this prediction dynamic in the future. It wins in terms of flexibility for forecasting in datasets like ours.

Table 5.1: RMSE values for each prediction model against validation set of 93 days

Method	RMSE Value
Moving Average Smoothing	130.44
Exponential Smoothing	113.43
Holt's Trend Model	100.20
ARIMA Model	43.22
Facebook Prophet Model	74

I would also like to add that the prediction strategy we just concluded is not limited to just better the autoscaling feature of Kubernetes, but it is also equally useful in predicting the resource utilization in cloud nodes so that we can use more and more reserved resources(which cost less) and only required number of on-demand resources(which cost more). This will help in **cost optimization.**

5.1 Limitations and Future Scope

There are two fields of focus as we look forward.
First, this method is a static one. We will focus on making this a dynamic model, possibly a custom pod autoscaler which can be used directly over a web-server to autoscale application according to server load.
Second, we can also explore Deep Learning Techniques like Recurrent Neural Networks (RNNs). Especially, Long Short-Term Memory RNN shows a lot of promise. Also, the ensemble methods combining Machine Learning Techniques and Statistics Method, working with large time series, are taking the field of forecasting ahead. We will focus on them also. [24]

The predictive resource allocation is a vast field of study in the cloud world with a lot of scope and this is just the beginning. The deeper we dive in, the more robust our cluster can become.

Bibliography

[1] 539ae779eb69a.pdf. http://www.mmc.igeofcu.unam.mx/acl/femp/MaquinasVirtuales/VirtualizacionEnLinuxCon-Containers/539ae779eb69a.pdf.

[2] Autoscaling - wikipedia. https://en.wikipedia.org/wiki/Autoscaling#Amazon_Web_Services_(AWS).

[3] The biggest thing amazon got right: The platform – gigaom. https://gigaom.com/2011/10/12/419-the-biggest-thing-amazon-got-right-the-platform/.

[4] Cbc_final_compress. https://papierenkarton.nl/wp-content/uploads/2017/02/Cloud_Begins_With_Coal.pdf.

[5] Cloud resource prediction model based on triple exponential smoothing method and temporal convolutional network - middlesex university research repository. http://eprints.mdx.ac.uk/29525/.

[6] Cost optimization in cloud computing. https://core.ac.uk/download/pdf/80716572.pdf.

[7] Designing your first application in kubernetes, part 3: Communicating via services - docker blog. https://www.docker.com/blog/designing-your-first-application-kubernetes-communication-services-part3/.

[8] Forecasting: Principles and practice. https://otexts.com/fpp2/.

[9] Google cloud platform blog: An update on container support on google cloud platform. `https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html`.

[10] Kubernetes documentation - kubernetes. `https://kubernetes.io/docs/home/`.

[11] Making facebook's software infrastructure more energy efficient with autoscale - facebook engineering. `https://engineering.fb.com/production-engineering/making-facebook-s-software-infrastructure-more-energy-efficient-with-autosca`

[12] Microservices. `https://martinfowler.com/articles/microservices.html`.

[13] Microservices at netflix: Lessons for architectural design. `https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/`.

[14] Overview — prometheus. `https://prometheus.io/docs/introduction/overview/`.

[15] A study on performance measures for auto-scaling cpu-intensive containerized applications — springerlink. `https://link.springer.com/article/10.1007/s10586-018-02890-1#citeas`.

[16] What is continuous delivery? - continuous delivery. `https://continuousdelivery.com/`.

[17] What is continuous delivery? - continuous delivery. `https://continuousdelivery.com/`.

[18] Why time series databases are exploding in popularity - techrepublic. `https://web.archive.org/web/20190626143018/https://www.techrepublic.com/article/why-time-series-databases-are-exploding-in-popularity/`.

[19] Fahd Al-Haidari, Mohammed Sqalli, and Khaled Salah. Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources. volume 2, pages 256–261, 12 2013.

[20] S A I B Arachchi and Indika Perera. Continuous integration and continuous delivery pipeline automation for agile software project management. 05 2018.

[21] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[22] Denis Kwiatkowski, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1):159 – 178, 1992.

[23] Tania Lorido-Botrán, Jose Miguel-Alonso, and Jose Lozano. A review of autoscaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12, 12 2014.

[24] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54 – 74, 2020. M4 Competition.

[25] Md. Habibur Rahman, Umma Salma, Md Hossain, and Md Tareq Ferdous Khan. Revenue forecasting using holt–winters exponential smoothing. *Research Reviews: Journal of Statistics*, 5:19–25, 12 2016.

[26] J. Shah and D. Dubaria. Building modern clouds: Using docker, kubernetes google cloud platform. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0184–0189, 2019.

[27] Priyanshu Srivastava and Rizwan Khan. A review paper on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8:17, 06 2018.

[28] P. M. Swamidass, editor. *Forecasting models, Holt'sHOLT'S FORECASTING MODEL*, pages 274–274. Springer US, Boston, MA, 2000.

[29] Andrew Tanenbaum and Maarten van Steen. *Chapter 1 of Distributed Systems - Principles and Paradigms.* 03 2016.

[30] Sean Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72, 09 2017.

[31] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[32] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, April 2010.

[33] Anqi Zhao, Qiang Huang, Yiting Huang, Lin Zou, Zhengxi Chen, and Jianghang Song. Research on resource prediction model based on kubernetes container auto-scaling technology. *IOP Conference Series: Materials Science and Engineering*, 569:052092, aug 2019.