Model-Driven Machine Learning for Predictive Cloud Auto-scaling

Hanieh Alipour

A Thesis
In the Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Electrical and Computer Engineering) at
Concordia University
Montréal, Québec, Canada

May 2019

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Hanieh Alipour**

Entitled: **Model-Driven Machine Learning for Predictive Cloud Auto-scaling**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. Nikolaos Tsantalis

_____ External Examiner
Dr. Dorina Petriu

_____ Examiner
Dr. Olga Ormandjieva

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Ferhat Khendek

_____ Supervisor
Dr. Yan Liu

Approved _____
Dr. Mustafa K. Mehmet Ali, Graduate Program Director

May 2019 _____
Dr. Amir Asif, Dean Faculty of Engineering and Computer Science

Abstract

Model-Driven Machine Learning for Predictive Cloud Auto-scaling

Hanieh Alipour, Ph.D.

Concordia University, 2019

Cloud provisioning of resources requires continuous monitoring and analysis of the workload on virtual computing resources. However, cloud providers offer the rule-based and schedule-based auto-scaling service. Auto-scaling is a cloud system that reacts to real-time metrics and adjusts service instances based on predefined scaling policies. The challenge of this reactive approach to auto-scaling is to cope with fluctuating load changes. For data management applications, the workload is changing and needs forecasting on historical trends and integrating with auto-scaling service. We aim to discover changes and patterns on multi metrics of resource usages of CPU, memory, and networking. To address this problem, the learning-and-inference based prediction has been adopted to predict the needs prior to provision action.

First, we develop a novel machine learning-based auto-scaling process that covers the technique of learning multiple metrics for cloud auto-scaling decision. This technique is used for continuous model training and workload forecasting. Furthermore, the result of workload forecasting triggers the auto-scaling process automatically. Also, we build the serverless functions of this machine learning-based process,

including monitoring, machine learning, model selection, scheduling as microservices and orchestrating these independent services by platform, language orthogonal APIs. We demonstrate this architectural implementation on AWS and Microsoft Azure, and show the prediction results from machine learning on-the-fly. Results show significant cost reductions by our proposed solution compared to a general threshold-based auto-scaling.

Still, there is a need to integrate the machine learning prediction with the auto-scaling system. So, the deployment effort of devising additional machine learning components is increased. So, we present a model-driven framework that defines first-class entities to represent machine learning algorithm types, inputs, outputs, parameters, and evaluation scores. We set up rules for validating machine learning entities. The connection between the machine learning and auto-scaling system is presented by two levels of abstraction models, namely cloud platform independent model and cloud platform specific model. We automate the model-to-model transformation and model-to-deployment transformation. We integrate model-driven with a DevOps approach to make models deployable and executable on a target cloud platform. We demonstrate our method with scaling configuration and deployment of two open source benchmark applications - Dell DVD store and Netflix (NDBench) on three cloud platforms, AWS, Azure, and Rackspace. The evaluation shows our inference-based auto-scaling with model-driven reduces approximately 27% of deployment effort compared to the ordinary auto-scaling.

Acknowledgments

First, and foremost, I am grateful to my Ph.D. supervisor Dr. Yan Liu for her guidance, support, patience, and encouragement. She is the dedicated, professional and caring advisor. Thank you.

I thankfully acknowledge my Ph.D. committee members, Dr. Ferhat Khendek, Dr. Olga Ormandjieva and Dr. Juergen Rilling for their time, effort and constructive comments. I would also like to extend my appreciation to the external examiner Dr. Dorina Petriu for accepting to serve in my Ph.D. thesis committee.

I thank my ex-colleagues in SAP company for their help, cooperation, and encouragements. Special thanks to Pierre-luc Orsini for his companionship, support, ideas, and fruitful discussions.

My very special thanks to Zahra Asadi and Pantea Koochemeshkian, my best friends. You were always there with me through the toughest moments. Your friendly advice, your soothing words, and your big heart helped me face all the obstacles and continue with my work. I am truly lucky to count you amongst my friends.

Also, I am forever indebted to my family for their encouragement, continuous support, love and encourage. My sincere thanks go to my mother Aseah, my dad Mohammad and my brother Hessam. Without you, this thesis would not have been possible. There are no words that can express my gratitude and love for you.

Last but not least, I owe a special thanks to my love and my best friend, Pierre Yves Lambert. I love you for being so understanding and for putting up with me through the toughest moments of my life. Thanks for being extremely supportive throughout this entire process. In addition, I thank Dr. Yves Lambert and Therese Baribeau for their kindness and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing offers many benefits including elastic resource allocation that enables automated and fast deployment of services. A key value of resource elasticity is allowing the provisioning and de-provisioning of computing resources on demand, via auto-scaling [24]. We have three different auto-scaling approaches on cloud environment [49]: The first approach is reactive, and it reacts to changes in the system state. It utilizes a threshold-based mechanism. The defined scaling policies adjust the number of instances, within the minimum and maximum number of instances. Second, the proactive or scheduled-based approach allows to scale the application resources based on the known load that will appear in future. It allows clients to pre-define a schedule where they proactively scale a system at certain points of time. For example, the service or application is heavily used on boxing day or black Friday and less used on other days. Third, the predictive approach predicts usage of the application in the future and thus changes done accordingly. This approach is suitable for environments with unplanned load spikes and fluctuated workload. While auto-scaling has shown considerable benefits of cloud computing, it still has remaining issues to solve.

1.1 Problem statement

Reactive and Proactive rather than Predictive approach: The current auto-scaling systems are reactive, and proactive rather than predictive [33], [50]. The reactive auto-scaling checks heuristic rules such as schedule-based, event-triggered or threshold value-based to determine whether it is necessary to perform scaling

actions. The heuristic rules involve a set of metrics of varies types and kinds, including system level, service level and application level metrics. This means that the decisions of auto-scaling actions are multi-modal. Heuristic rules lack of the adaptivity to combined effects and correlations among entities and metrics involved in the auto-scaling process.

We need to forecast future demand by taking into account the workload history. Having made a determination and prediction, the scaling engine proceeds to make a decision concerning an alternative plan because of this unexpected situation. It forecasts future demand for sustained performance despite workload fluctuations. So, we need to study the behavior of demand from workload history. There is no process that includes different cloud-based services such as a monitoring service, several machine learning models, model selection and prediction in order to analyze and determine the behavior of the systems under various workloads and capacity contractions. Meanwhile, we need to design an architecture where the machine learning service is composed of a set of cloud services rather than being implemented as a monolithic entity. The challenge is to benefit complex cloud services by enabling each independent service to be designed, implemented, scaled, upgraded independently.

Auto-scaling policy is mostly based on a single metric: The auto-scaling policy is mostly based on a single metric, including CPU or Memory usage. The group of metrics together define the behavior of the system, so scaling action based on a single metric is not sufficient. A monitoring service helps an existing auto-scaling by setting alarms to capture the workload changes and sends an alarm when there is an increase in workload to the auto-scaling service. In addition, some cloud providers offer the ability to define multi policies. This means a policy is defined for each metric if you need to consider the effect of several metrics. However, each policy will be defined separately and cannot demonstrate the relationship between metrics. Hence learning-and-inference based approach brings a novel approach to the auto-scaling system to act based on the prediction results learn from multiple metrics as the workload features.

In order to have accurate forecasting for auto-scaling, there is a need to train our models based on the multi metrics. To calculate a mix metrics effect, we need to check the dependency and weight between metrics before training our workload. There are also some important limitations that need to be considered for training the

multi metrics workload with the help of a machine learning approach. For instance, when we have multi metrics, each metric may have a different effect on the behavior of the system. Multi-metrics learning is challenging because we need to find the relationship between metrics. So, we need to calculate the mix metric effect because it is difficult to exploit potential dependencies among the target variables and show the relationship between the different target variables.

How to integrate machine learning with auto-scaling process: Machine Learning is the subject that studies methods for automatically deriving models from data, and it has been successfully applied in many areas of software engineering and cloud computing. A machine learning process includes training and inference phases. Training refers to the process of fitting a machine learning model by optimal model parameters from the sampling data. Inference refers to the process of running a trained model to make predictions. For both training and inference, there are common entities: (1) inputs, (2) outputs, (3) parameters, and (4) assessment metrics. A machine learning model requires a set of inputs and outputs. The inputs are divided into two parts: training dataset and testing dataset. Outputs are a set of data that is being predicted by the trained model. For example, in cloud services inputs can be low-level CPU, memory or network usage or higher-level kinds of data tied to the services or applications, such as requests served per second. For data of distinct categories that are non-numerical, data are first encoded into a numerical form by feature engineering techniques. Hence, in our work, we consider input and output are numeric data and time series type. Parameters include hyper-parameters and model parameters. Hyper-parameters are configuration variables that typically searched by greedy algorithms. To check the accuracy training and inference results, assessment metrics are necessary. The assessment metrics are also used for the model selection that is a process to select a suitable model from a set of candidate models. The value of assessment metrics is also numerical applied.

The machine learning techniques need to be combined with the auto-scaling process in order to scale, based on the prediction results. Integrating machine learning components with the auto-scaling system requires substantial numbers of manual tasks. In a nutshell, entities are in different scales and characters. The integration of machine learning and auto-scaling should plug-and-play different machine models in a uniform way rather than model-by-model. To support a wide range of machine

learning models, we need to generalize the common entities and represent them at an abstract level.

How to automate the process across cloud platforms: The life-cycle of the auto-scaling system includes five common parts: the auto-scaling group, the monitor, the scaling policy, the scaling engine, and the launch configuration [46]. Machine learning service also includes several components: monitoring, data and model storage, machine learning algorithms, validation, and model selection. Integrating machine learning components with the auto-scaling system requires substantial numbers of manual tasks. There is a need to automate the machine learning integration process and reduce deployment effort. In addition, current auto-scaling approaches that are offered by cloud providers typically require an expert to manually configure the added resources for Cloud-based data management applications such as Dell DVD store database [22], [57] and Cassandra database cluster [5] because auto-scaling services only create and add the new instance to the auto-scaling group without re-configuring the cluster. Manually configuring resources may increase the system downtime. For example, for Cassandra dataset cluster [6], if we add a new instance we need to reconfigure the cluster and change some configuration files include Cassandra.yaml.

Also, models should be transformed into deployment entities to help reduce deployment effort. Otherwise, models remained at the design phase and isolated from the rest of the life-cycle of the machine learning process. There is a need to automate the machine learning integration process and reduce deployment effort. The goal is to hide the complexities of using different technologies from developers who are responsible for managing the cloud environment. In other words, the deployment effort is based on the effort required to tackle the considerable complexities arising from the use of different cloud technologies and services. Moreover, minimizing effort is extremely important in practice since deploying and configuring cloud services can be expensive. There is a need to reduce the gap between problem domains and service implementation with technologies that support the efficient transformation of the abstract model to service implementations.

1.2 Research Questions

To tackle these problems, a Model Driven Engineering (MDE) approach has been adopted to build a Cloud platform independent and Cloud specific platform models. In this approach, the models drive the process of machine learning and auto-scaling system. These models are specified at different levels of abstraction and automated tools are used for model-to-model, model-to-deployment and deployment-to-run-time transformations between the levels. Our approach aims to provide answers for four research questions.

- RQ1. How to design an architecture solution to orchestrate the machine learning process?
- RQ2. Does representing machine learning models help an auto-scaling system?
- RQ3. How to integrate the machine learning with the auto-scaling system?
- RQ4. How to evaluate the effectiveness of the proposed approach?

1.3 Contributions

- We introduce a new solution in order to train a model based on multi-metrics. We use a weighting approach based on Shannon information entropy, which expresses the relative intensities of metrics importance to signify the average intrinsic information transmitted to the decision maker. Then, we apply a microservice approach to orchestrate the machine learning process for auto-scaling. Microservices are fully decoupled by means of well-defined and explicitly published interfaces. A microservices approach provides the opportunity and flexibility to use different technologies and different programming languages. In addition, services are communicating with each other using language-agnostic APIs. With the help of our proposed microservice process, we create and train machine learning models, and publish them as web services. One of the advantages of making the machine learning functions a microservice is the training and inference execution can be managed independently. Machine learning algorithms and strategies can be changed without impacting the other services. We calculate the run-time cost for our proposed solution and compare it with the

cost of the general process to demonstrate the improvement. For implemented applications, our solution is saving approximately 2$ for one hour (**RQ 1**).

- We propose a meta-model to represent the common entities of a machine learning process. Hence, a specific machine learning model becomes an instance of the machine learning meta-model. We propose a high-level abstraction for modeling the features of the machine learning such as machine learning algorithm types, inputs, outputs, parameters, and evaluation scores. Furthermore, we propose a new data type metric to meet the machine learning requirements. Then, we integrate a machine learning meta-model with the auto-scaling meta model and introduce the predictive auto-scaling model. A Model Driven Engineering (MDE) approach has been adopted to build a cloud platform independent and Cloud specific platform models. These models are specified at different levels of abstraction. We design the CPIM and the CPSM which they are focused on the predictive auto-scaling system (**RQ 2**).

- We develop a method to integrate the model-driven solution with a DevOps approach. DevOps is an emerging paradigm to actively foster the collaboration between system developers and operations in order to enable efficient end-to-end automation. DevOps is typically combined with cloud computing, which allows rapid, on-demand provisioning of underlying resources such as virtual servers, storage, or database instances using APIs in a self-service manner. The goal is to bring together the strengths of DevOps and model-driven approach in order to minimize the effort. So, models are transformed into scripts of DevOps tools. Hence, the deployment, configuration, and trigger of cloud services are carried out by DevOps tools. Thus, the result of this integration facilitates the deployment actions and make the proposed models deployable and executable on the target cloud platform (**RQ 3**).

- We designed a test scenario for estimating the effort of the predictive auto-scaling model on two Cloud platforms. Based on the evaluation result, we observe the reduced effort using our model-driven method. The evaluation shows our inference-based auto-scaling with model-driven reduces approximately 27% of deployment effort compared to the ordinary auto-scaling (**RQ 4**).

In a nutshell, we categorize our research work in two groups: Scientific and Development contributions.

Scientific Contributions:

- Propose a new solution in order to train a model based on multi-metrics.
- Propose a meta-model to represent the common entities of a machine learning process.
- Integrate a machine learning meta model with the auto-scaling meta-model and introduce the predictive auto-scaling model.

Development Contributions:

- Apply the microservice approach to orchestrating and managing the machine learning process for auto-scaling.
- Employ the DevOps approach to facilitates the deployment actions and make the predictive auto-scaling models deployable and executable on the target cloud platform.

1.4 Structure of the thesis

The rest of the thesis is organized as follows:

Chapter 2 presents the state of the art related to machine learning, auto-scaling, model-driven development and DevOps that will explain the concepts and ideas relevant to this research.

Chapter 3 present the proposed architecture based on a microservice approach for orchestrating and integrating machine learning service and auto-scaling in Cloud. It also includes a new solution to train a model based on multi-metric at system-level.

Chapter 4 describes the proposed new modeling approach for machine learning service in Cloud computing. demonstrates how we integrate our proposed solution with the auto-scaling process. It also includes CPIM, CPSM, and interaction of DevOps and Model-driven approach.

Chapter 5 describes the experiments and evaluation of the approach.

Chapter 6 concludes the thesis by highlighting our contributions.

Chapter 2

Related Work

In this chapter, we will discuss the works from the literature that are closely related to each of the thesis contributions. We first discuss and analyze the works related to the microservice architecture and the machine learning service. After that, we review the works related to the model-driven approach and the auto-scaling cloud service.

2.1 Microservice Architecture

Microservices are fully decoupled by means of well-defined and explicitly published interfaces. A microservices approach provides the opportunity and flexibility to use different technologies and different programming languages. In addition, services are communicating with each other using language-agnostic APIs. With the help of our proposed microservice process, we create and train machine learning models, and publish them as web services. There are several advantages in merging micro-service architecture with the cloud. Villamizar et al. [67] evaluate micro-services on a cloud and report the performance difference between monolithic services and micro-services. They compare the performance metrics and cloud operation cost between two architectures. The monolithic and microservice architectures were deployed as it is shown in Figure 1.

They identified some of the benefits and challenges that microservice architectures provide to businesses. One of the benefits of using microservices is the ability to publish a large application as a set of small applications (microservices) that can be developed, deployed, scaled, operated and monitored independently. The agility, cost

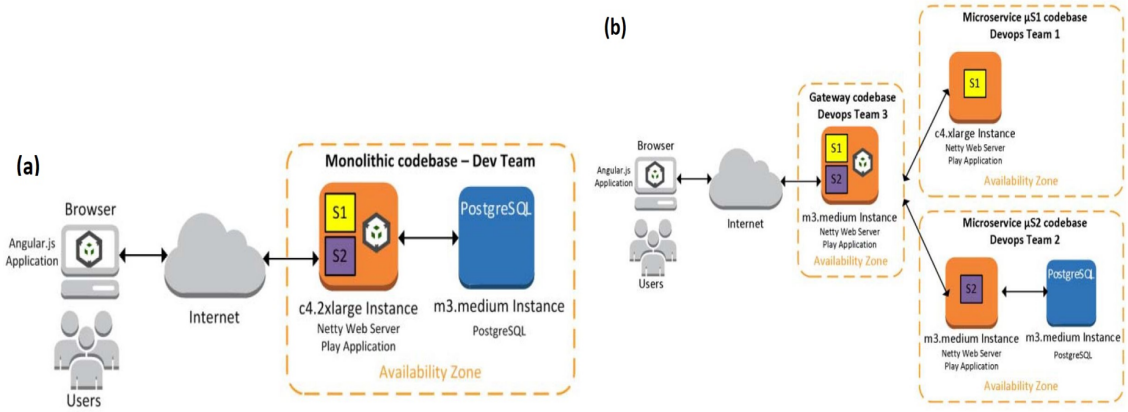


Figure 1: Deployment of (a) the monolithic architecture and (b) the microservice architecture on AWS [67]

reduction and granular scalability, brings some challenges of distributed systems. Their paper [68] presents detailed performance and cost comparison of traditional clouds with microservices and the AWS Lambda serverless architecture. An enterprise application was benchmarked, and results show that serverless infrastructures can reduce cost without impacting performance. In our work, we deployed our proposed architecture on two different cloud environments: Azure and AWS and demonstrated how the workload cost is reduced by proposed microservice architecture.

The work in [44] proposes a microservice architecture for dynamic service registration and discovery. This paper uses OpenStack as a case study and their work illustrates the advantages of containerizing cloud infrastructure services and combining with a microservice style architecture. They identified three main challenges to improving operational efficiency: (1) minimize cross-configuration of services, (2) maintain a state of running services, and (3) provide safe access to host resources. Our work leverages lambda serverless functions to form microservices. These microservices are the facades of machine learning modules running on the cloud platforms. Figure 2 shows the container-based microservice architecture on OpenStack.

This [63] paper focusing on micro-service monitoring and proposing an architecture by integrating management functions into the micro-services. In this paper, they proposed an architecture that enables scalable and resilient self-management of micro-services applications on the cloud.

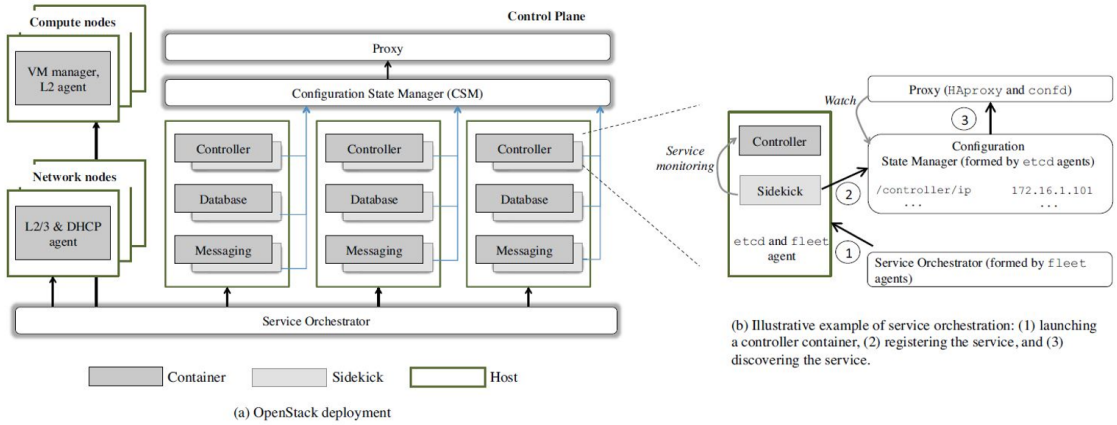


Figure 2: A container-based microservice architecture of OpenStack. (a) OpenStack deployment, (b) Service registration and discovery in microservice architecture [44]

Table 1: Category of Auto-scaling Deployment Automation Method in Service Level and Auto-scaling System Level

Platform	Automation	
	With ML	With Model-Driven
Service Level	[28] [61] [36]	[62] [25] [59] [34] [71] [30]
Autoscaling System Level	[61] [37] [38]	[36] [66] [51]

2.2 Machine Learning Service

The auto-scaling system has received a great interest both within the service and the system level. Studies related to our work fall into two categories for automating the deployment of auto-scaling. The solutions proposed for automation can be classified in "With ML (machine learning)" and "With Model-Driven". Table 1 lists the platform levels and automation with machine learning and model-driven, and references of work.

Machine learning techniques in resource prediction are applied to dynamically build the model of resource consumption under a specific workload. Samuel et al. [21] present methods for predicting resource utilization and provisioning strategies, to improve performance and respect SLAs. They compared three different machine learning algorithms for random-like workload traffic pattern: Support Vector Regression (SVR), Neural Networks (NN) and Linear Regression (LR). They used WEKA for training and testing the three machine learning techniques. However, their focus is only on calculating the performance of algorithms and they just used AWS

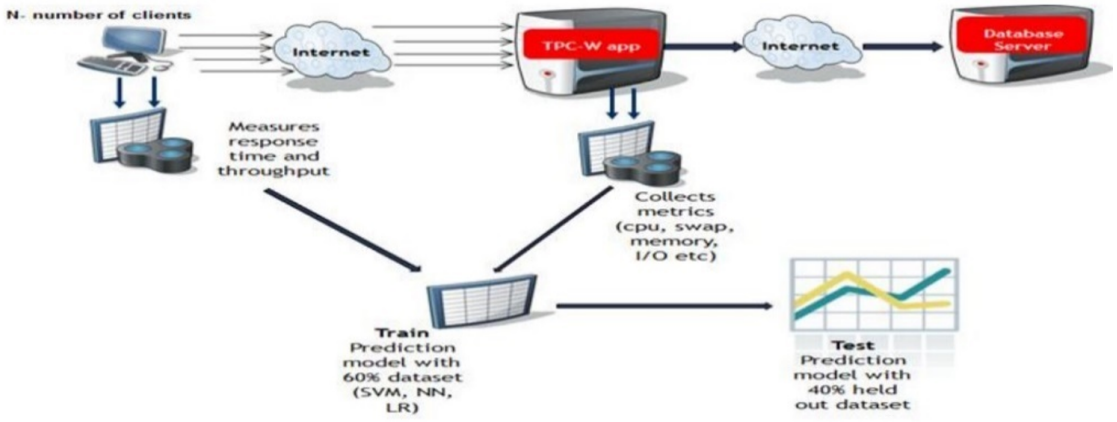


Figure 3: System Architecture for applying machine learning on multi-tier web application on AWS [21]

environment for evaluating their work (Figure 3).

They [42] present a method for learning appropriate application- and workload-specific resource provisioning policies. With the help of access log and unsupervised machine learning algorithm, they identified the parameters of workload patterns for multi-tier Web applications. The approach provides resource allocation policies for each workload.

Roy et al. [55] combined an ARMA model for workload forecasting, with the look-ahead controller in order to optimize the resource allocation problem. They use a second order ARMA for workload prediction, based on the last three observations. The predicted value is then used to estimate the response time. In [54] machine learning techniques, such as Support Vector Machine (SVM) and Neural Networks (NN), were utilized as time-series prediction techniques to model different workload patterns. SVM algorithm which is based on the structural minimization and the ANN algorithm which uses the empirical minimization principle was used. As shown in Figure 4, Monitor, Predictor, and Decision Maker are the main components of a predictive auto-scaling system. To capture the current performance of cloud computing environment, auto-scaling systems monitor one or more performance metric(s). Their result shows ANN has better accuracy in forecasting the unpredictable workload. In addition, Mehran et al. [46] reviewed existing auto-scaling techniques for popular cloud providers. Then, they modeled core features and entities of the auto-scaling operations. Furthermore, the model allows a proactive analysis of workload

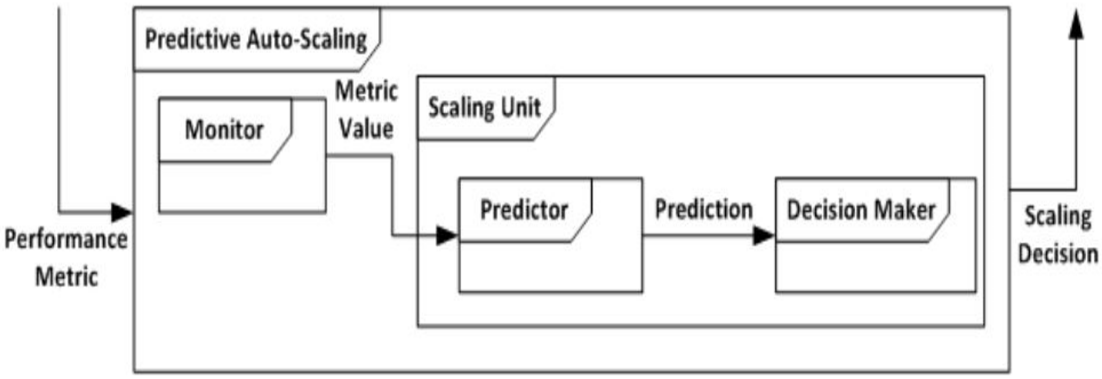


Figure 4: Architectural overview of a predictive auto-scaling system [54]

patterns and estimation of the responsiveness of the auto-scaling operations. They used Google cluster trace data to evaluate their work.

Multi-target learning has rapidly attracted interest in the machine learning literature, Tsoumakas et al. [65] presents a method for multi-target regression that constructs new target variables by the help of random linear target combinations. They also discussed how their work can connect to the multi-label classification algorithms. Furthermore, they tested their approach on 12 multi-metrics datasets. However, in our solution, we have s single continuous target instead of multiple continuous targets based on the set of input variables. This work [20] introduced FIRE (Fitted rule ensembles), an algorithm for learning rule-based ensembles for multi-target regression problems. They tried to improve the accuracy of the algorithm by adding simple linear functions to the ensemble. The goal is having a solution that can learn multi-target models. Still, the weight calculation for metric is missing in this work. In addition, Kocev at al. [47] represented the advantage of multi-target over a single-target modeling approach. In their work, they compared single-target approach (a regression tree) with multi-target approach (a multi-target regression tree) for modeling their data. Variables in dataset are described by multiple scores. To model the multiple scores, they used two approaches: single-target and multi-target regression. With single target regression, they learn a model for each score separately, while with the multi-target regression, we learn one model for all scores. In the meantime, our dataset contains system-level metrics and we need define first the weight of each metrics before apply machine learning algorithm.

There have been a number of works on dynamic resource provisioning on Cloud computing environment. Gujarati et al. [40] introduced a new distributed approach for auto-scaling called Swayam. Their approach is working on resource efficiency and SLA compliance for ML inference services in a distributed setting. They try to provide an appropriate number of service instances by predicting load, creating new instances as needed, and removing unnecessary instances. Their work is based on single metric, while we are working on multiple metrics. Also, we proposed microservice process for machine learning service to decouple the services. Wajahat et al. [70] introduces an application-agnostic machine learning based autoscaler called MLscale. They used neural network for online (black-box) performance modeler and they also presented a regression-based metrics predictor to estimate post-scaling application and system metrics. They are working on single metric, but we consider multi metrics and scaling based on multi-metrics. We present microservice process for workload prediction, so any service can change independently. However, their work is implemented as a simple controller in Python using a few hundred lines of code.

Gandhi et al. [30] proposed ADARES, an adaptive system that dynamically tunes resources of VMs. They used the contextual bandits framework with transfer learning to optimize configurations of VMs in a cluster, and exploits cluster, node and VM-level information to promote efficient resource utilization across VMs. However, their work applied on a single metric and they didnt focusing on modeling auto-scaling system. The key component of their solution [37] is the modeling engine that characterizes the workload and then quantitatively evaluates different scaling options for that workload. Their modeling engine leverages Amdahls Law to model service time scaling in scale-up environments and queuing-theoretic concepts to model performance scaling in and scale-out environments. They employed Kalman filtering to account for inaccuracies in the model-based methodology and to dynamically track changes in the workload and cloud environment. However, they did not present the abstract model of auto-scaling system level.

2.3 Auto-scaling service in Cloud Computing

Many academics and cloud technology vendors have loosely defined the concept of auto-scaling [41]. Gartner defines auto-scaling as follows: Auto-scaling automates

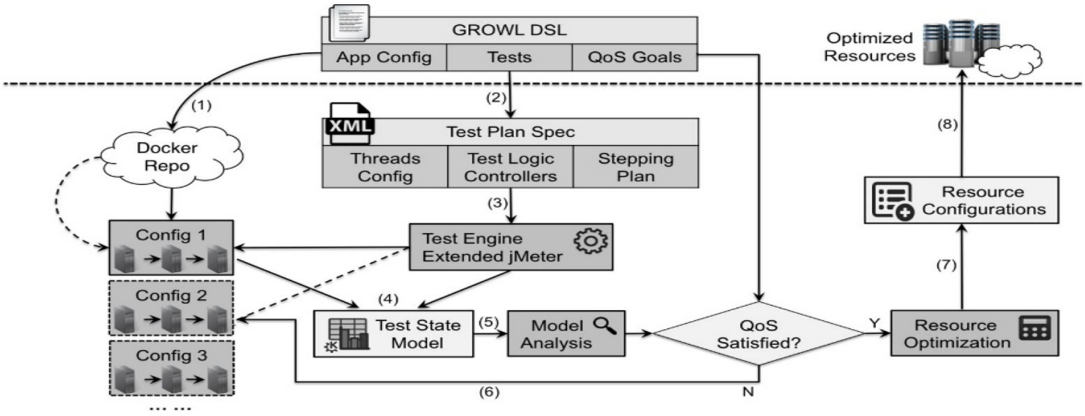


Figure 5: Overview of the Resource Optimization, Allocation and Recommendation System (ROAR) [62]

the expansion or contraction of system capacity that is available for applications and is a commonly desired feature in cloud IaaS and PaaS offerings. When feasible, technology buyers should use it to match provisioned capacity to application demand and save costs. [9] [2]. Meanwhile, In RightScale [18], auto-scaling is defined as a way to automatically scale up or down the number of compute resources that are being allocated to your application based on its needs at any given time.

Sun et al. [62] proposed a ROAR modeling framework to automate, simplify and optimize the testing and derivation of cloud resource allocation for web applications to meet the QoS goal. The key components in the ROAR framework are summarized in Figure 5. Due to the uniqueness of the proposed framework, end-to end test orchestration of resource allocation, load generation, resource utilization metric collection, and QoS metric tracking (delay, throughout) perform automatically. In addition, they support to deploy an application to multiple cloud providers. They mainly focus on optimizing load testing for resource allocation and enhance the existing cloud deployment.

They do not introduce new auto-scaling and their work does not mention how to take care of "scale out" or "scale in" based on unexpected load. They do not think about the load fluctuation issue, which has a huge effect on auto-scaling. There is still a lack of generalization of auto-scaling and synchronization between auto-scaling mechanisms in two different cloud environments. Meanwhile, their work does not consider some quality attributes such as high availability and security.

Meanwhile, Song et al. [61] used the exponentially weighted moving average (EWMA) model to predict the demand for the number of VMs. They proposed an online bin packing approach that uses virtualization technology to allocate cloud resources dynamically based on application demands. Their proposed approach supports green computing by optimizing the number of servers used. Moreover, Bunch et al. [28] designed provisioning systems that predict the future service demand to decide the amount of resources for provision. They designed a pluggable and cost-aware auto-scaling system that forecasts the future demand by analyzing metrics such as the request volume.

2.4 Measurement

This work [31] objects to monitor and analyze load in cloud infrastructure by applying load collection and evaluation techniques. The goal is also to investigate CPU load relationship between host and guest machines under varying workload conditions. Also, Capra et al. [32] worked on cloud computing client-initiated workloads. An investigative presented in the work defines a process of workload trace characterization and synthetic workload generation. They [56] provided an extensive study on the variance of the current most popular Cloud computing provider Amazon EC2. They used established micro-benchmarks to measure performance variance in CPU, Memory, and Network. Li et al. [48] collected metrics adopted in the existing cloud services evaluation work. The collected metrics were arranged following different cloud service features to be evaluated, which essentially constructed an evaluation metrics catalog. This metrics catalog can be used to facilitate the future practice and research in cloud services evaluation. Moreover, considering metrics selection is a prerequisite of benchmark selection in evaluation implementations, this work also supplements the existing research in benchmarking the commercial Cloud services. In addition, they [19] analyzed the research works in the cloud monitoring area. They have considered the main activities on the cloud environment that have convincing benefit from or actual need of monitoring. They have provided background and definitions for key concepts. They derived the main properties that cloud monitoring systems should have, the issues arising from these properties, and the related contributions provided in literature so far.

2.5 Model-driven Software Development (MDD)

The complexity of applications is increasing. At the same time, there are high expectations for the quality of software and applications. Model-driven development (MDD) [69] can be a solution to take care of these challenges. MDD is a software engineering approach that captures domain knowledge in high-level. The key goal is to concentrate on model-orientation more than code-orientation in software production.

Gandhi et al. [36] presented Dependable Compute Cloud (DC2) as a new cloud service. In their model-driven auto-scaling approach, they tried to automatically scale applications in a cost-effective way. They combined a Kalman filtering technique and queuing theoretical model in DC2 to choose the right scaling action. However, they did not consider multiple clouds and the vendor lock-in issue in their approach. In addition, quality attributes, such as high availability, were missing in their work.

MODAClouds [25] follows a model-driven approach to design and execute the application on multiple clouds to support interoperability and prevent vendor lock-in. They provided automatic deployment of applications on multiple clouds with guaranteed QoS. MODAClouds mainly aimed to support migration applications from cloud to cloud as needed. However, their work does not focus on auto-scaling mechanisms in cloud computing. MODAClouds consider three levels of abstraction: CCIM, the cloud enabled Computation Independent Model to describe an application and its data, CPIM, the cloud-Provider Independent Model to describe cloud concerns related to the application in a cloud agnostic way, and CSPM, the Provider Specific Model to describe the cloud concerns needed to deploy and provision the application on a specific Cloud (Figure 6).

In order to have strong and flexible software solutions for cloud software applications, Sharma et al [59] studied the MDA approach to developing software systems. They tried to highlight the benefit of incorporating the MDA approach in the development of cloud SaaS in contemplation of minimizing time, costs and efforts in application development. Furthermore, Eldein et al. [26] studied how to use Model-Driven architecture development and discussed open issues and explained future research problems. In fact, this paper aimed to survey and analyze the research and challenges that have been emerging in Cloud computing and Model-driven.

Ferry et al. [34] [35] proposed a model-based framework called Cloud Modeling Framework (CloudMF). They employed MDE to face the complexity of developing

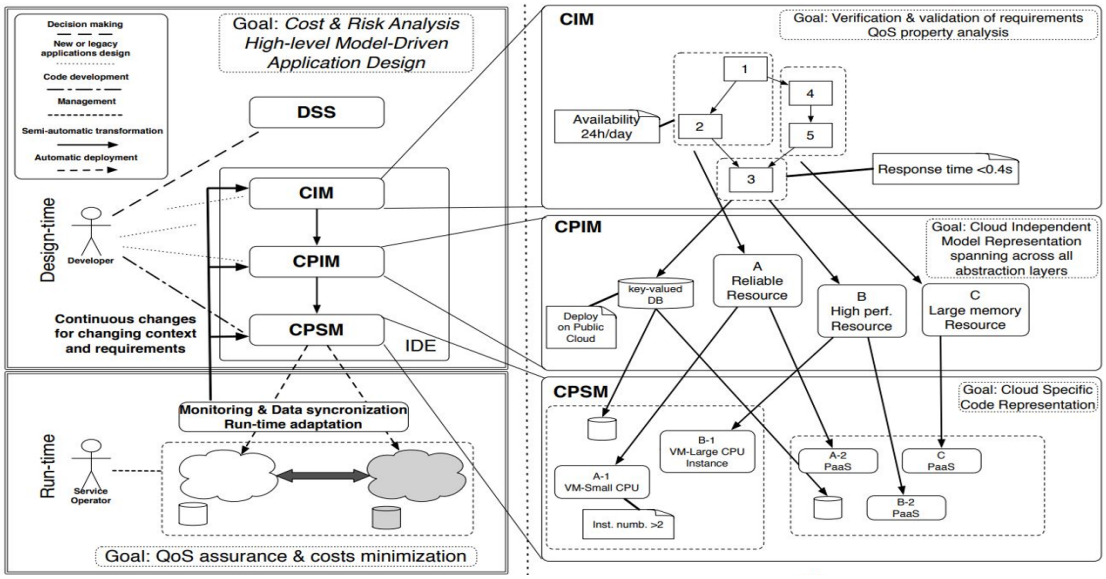


Figure 6: MODAClouds approach [25]

complex systems such as multi-cloud systems. They also introduced CloudML, which relies on model-driven techniques. It is domain-specific modeling language and it facilitates the specification at design-time of provisioning and deployment. On the other hand, Caglar et al. [29] presented a new model-driven engineering solution and described the design of the domain-specific modeling languages. They tried to investigate model-based simulation and automated deployment in the Cloud. Moreover, they tried to reduce the complexity of price calculations and uncertainties by simulating performance and cost.

2.6 DevOps

The DevOps approach has been investigated for its combination with model-driven development to improve quality of service for a complex system. Guerriero et al. [39] introduced SPACE4Cloud as a DevOps environment that links the processes of design-time performance evaluation and the runtime self-adaptation of a cloud application to reduce the cost of cloud applications.

Wettinger et al. [71] presented a concept that integrates the model-driven cloud management and configuration management using Chef. The Chef is an agent-based framework and requires master-agent communication. In our work, we chose Ansible

to demonstrate our work, which is simplified in terms of communication via standard SSH commands.

Bruneo et al. [27] introduced the CloudWave that employs DevOps to create an execution analytics cloud infrastructure to obtain high QoS levels. Their goal was to improve both the development of SaaS solutions and the management of their operation and execution.

2.7 Evaluation methods

We have some research related to how to calculate the effort of deployment. Jiang et al. [43] studied the maintenance of infrastructure-as-code. They reviewed the co-evolution of Puppet and Chef configuration files for 256 OpenStack projects. Their study shows bugs in those configuration files related to the number of changes and size of the files. They believe it is necessary to establish a link between infrastructure-as-code and the software quality. In addition, Elbaum et al. [52] focused on measuring effort of software development process. They compared the complexity of builds and measured the code churn. Furthermore, code churn is defined as a set of changes, such as added, modified or deleted files, from one version to another. They demonstrated the effectiveness of code complexity churn. Sharma et al. [60] developed the tool Puppeteer to detect code smells in Puppet. They analyzed common smells in software engineering and Puppet-related smells and measured them on a number of GitHub repositories. They examined 4621 repositories from GitHub. This work focused on maintainability of configuration code quality, whilst we focus on the portability of configuration code quality.

2.8 Summary

System-level metrics are server information monitored at the physical server or virtual machine (instance) layer, such as the utilization of CPU, memory, and network resources, memory. These data can be obtained through a monitoring platform of the cloud provider. Based on the reviewed researches, we have multiple metrics to monitor and each metric may have a different effect. Moreover, an auto-scaling solution from previous works tries to overcome the limitations of reactive mechanisms

by employing prediction methods. However, the prediction solutions are based on single metric and deployment efforts are still high. In some works, integrating the machine learning service with auto-scaling is missing. None of these methods address an approach for automating the deployment of predictive auto-scaling.

To the best of our knowledge, none of the existing approaches consider commonality and diversity for machine learning service and auto-scaling system in abstract levels as well as scaling based on multi metrics.

Chapter 3

Integrating Machine Learning with Auto-scaling

In this chapter, we explain how to use a microservice approach to orchestrate and integrate machine learning components with the auto-scaling system to have a predictive auto-scaling system.

3.1 Multiple Metrics Analysis

For monitoring in a cloud environment, there are two types of metrics: Application-level and System-level (resource) metrics. In our work, we utilize the system-level metric. The system-level metric includes resources usage of virtual instances such as CPU, memory, disks, and network interfaces. We consider resource utilization because it represents the percentage of time that the resource is busy, or the percentage of the resources capacity that is in use.

We collect samples of multi-metrics data and the following metrics were selected for prediction:

- NetworkIn: This metric identifies the volume of incoming network traffic to an application.
- NetworkOut: This metric tracks the volume of outgoing network traffic to an application.

- CPU utilization: The percentage of allocated computer units that are currently in use.
- Memory utilization: The percentage of allocated memory units that are currently in use.

We monitor CPU utilization because CPU usage is a critical computational resource that plays an important role in resource management [31]. In addition, we monitor memory usage as well. The previous research [32] has analyzed cloud computing workload characteristics and synthetic workload generation. This research has proved that there is a positive relationship between memory and CPU consumption, and they are highly correlated. Also, networkIn and networkOut metrics are important for cloud-based services such as virtual instance that rely on consistently strong network connections.

When we have multiple metrics, each metric may have different effects on the behavior of the system. In order to calculate the effects, we need to compute the weight of each metric and find the multi-metrics mixture effect. Multiple Attribute (metric) Decision Making (MADM) refers to make decisions in the presence of multiple metrics. In line with the MADM concept, our focus is on building a machine learning model which involves multiple criteria and determine the appropriate weight for each metric. Shannons entropy method is one of various methods for finding weights discussed in literature [45], [72]. In terms of determining metric weight, this is one of the most widely adopted approaches as it expresses the relative intensities of metric importance. Here, we present how we solve the problem of multi-metrics via analysis of metrics weights by means of entropy. In this research, we monitored one sample per minute for each metric and thus collected 20160 samples of multi-metrics for training over the course of two weeks. Figure 7 provides the workload of multi-metrics.

Moreover, our proposed solution calculates the multiple metrics mixture effect based on the weight average of metrics. We suppose there are m samples of values (V) in T time period with me number of metrics. Where m is 20160 samples for two weeks and the four metrics are CPU, Memory, NetworkIn and NetworkOut. So, the size of me is 4. The following steps demonstrate our solution for calculating the weight of each metric.

Step 1: Normalize the records of each metrics:

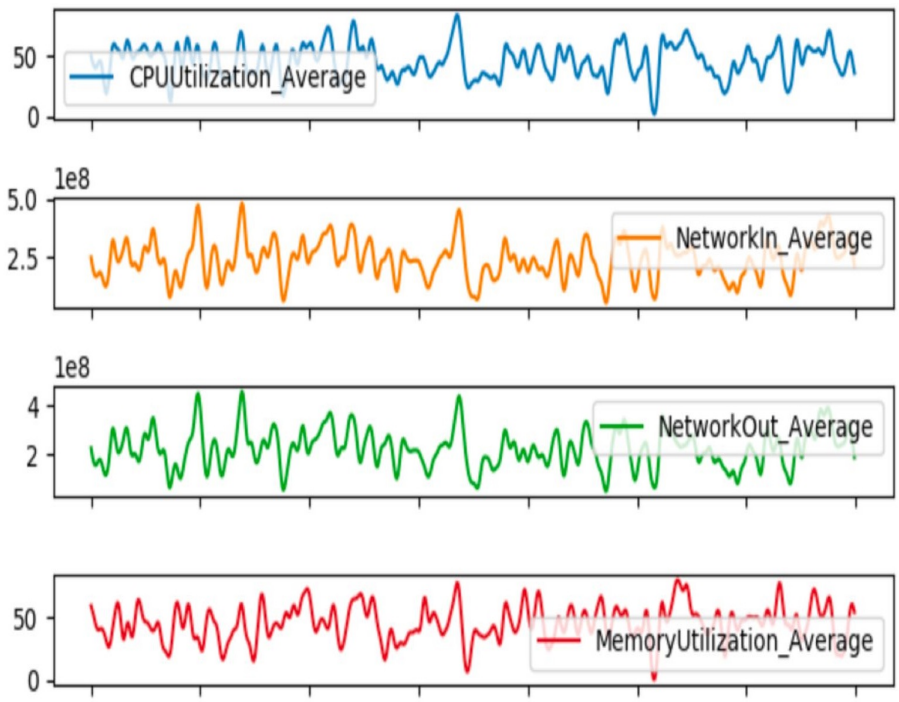


Figure 7: Sample of multi metrics workload generated on cloud environment

$$F_{norm}(V_i^{me}) = V_i^{me} / \sum_{i=1}^{m} V_i^{me} \tag{1}$$

$$i \in [1, m]$$

$$me \in [CPU, Memory, NetworkIn, NetworkOut]$$

Step 2: Calculate Entropy for each metric as :

$$E_{me} = (-K) \sum_{i=1}^{m} (F_{norm}(V_i^{me}) \ln(F_{norm}(V_i^{me}))) \tag{2}$$

K is the entropy constant and is equal to

$$K = 1/ln(m)$$

Step 3: Compute the degree (weight) of importance of metrics as:

$$W_{me} = 1 - E_{me} / \sum_{j=1}^{4} (1 - E_{me}) \tag{3}$$

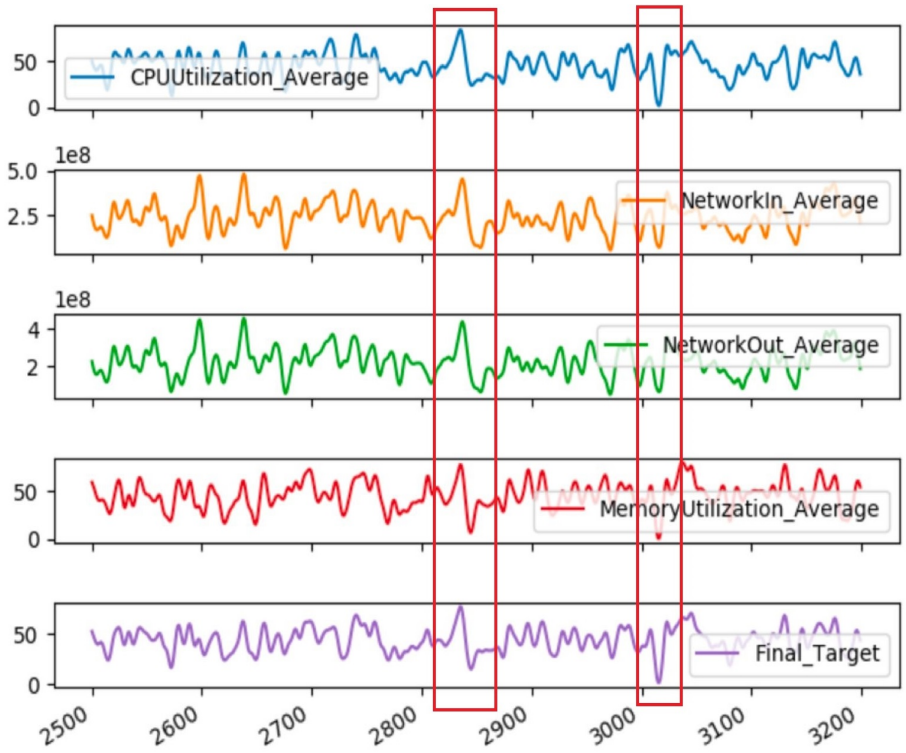


Figure 8: Final result of the Weighted Metrics with Aggregation (WMA) F_{ti}

When we have the weight of each metric, we can use the weight average metric method to calculate the multiple metrics mixture effect which is the Weighted Metrics with Aggregation (WMA) F_{ti}. Therefore, a metric with a higher weight contributes more to the weighted mean than metrics with a lower weight. So, for each row of the dataset, we calculate the average weight of all metrics. After we calculate the multiple metrics mixture effect, we use machine learning algorithms to train and predict the data. Figure 8 demonstrates the final result of aggregated metrics. The calculated final target captures the workload pattern on multiple metrics.

$$F_{t_i} = \sum_{i=1}^{m} (W_{me} * F_{norm}(V_i^{me}) / \sum_{j=1}^{4} (W_{me}) \tag{4}$$

3.2 Machine Learning Models

In our work, we apply six machine learning models: Long Short-Term Memory Model, Bidirectional LSTM, Vector Auto Regression, Support Vector Regression, Gradient

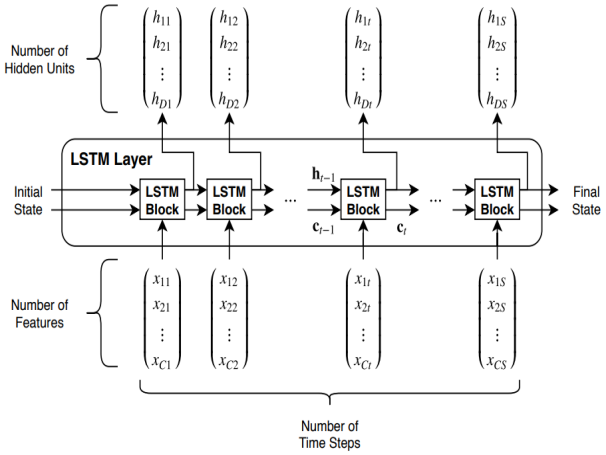


Figure 9: Illustrations for basic LSTM [13]

Boosting Regression, and Linear Regression. These six models are described below with a focus on their online training and prediction.

Long Short-Term Memory Model (LSTM): Long Short-Term Memory (LSTM) network is a variation of Recurrent Neural Networks (RNN), which, at its most basic level, extends memory. LSTMs enable RNNs to remember their inputs over a long period of time. LSTMs contain information in a memory where it can read, write, and delete information. LSTM has three gates: input, forget, and output gate. These gates determine whether new information is inputted (input gate), deleted(forget gate), or whether there is an impact on the output at the current time step (output gate). Different hyper-parameters affect the model capacity differently. Learning Rate is the most important hyper-parameter. The model capacity is maximized if the learning rate is set to the correct value, which may not necessarily be the largest or smallest value. The core idea behind LSTM lies in that at each time step, a few gates are used to control the passing of information along with the sequences that can capture long-range dependencies more accurately (Figure 9).

Bidirectional LSTM Model: Bidirectional LSTM (Bi-LSTM) is a combination of Long Short-Term Memory (LSTM) and Bi-directional Recurrent Networks. However, both LSTM and RNN can only receive information from the previous context so that further improvements are made using the Bidirectional Recurrent Neural Network (Bi-RNN). As its name suggests, Bi-RNN can receive information in two directions, from the front and back. The combination of Bi-RNN combined with LSTM produces Bi-LSTM. So, the advantages of LSTM, with its storage in cell memory, and

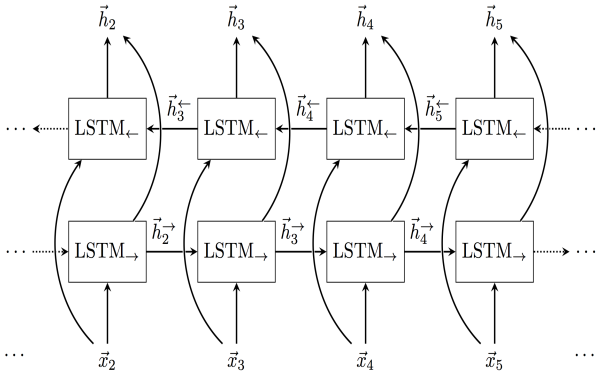


Figure 10: Illustrations for basic Bidirectional LSTM [4]

Bi-RNN, with access to information from the context before and after, combine to make Bi-LSTM excel. Bi-LSTM has the advantage of LSTM with feedback for the next layer. However, Bi-LSTM can also handle data with dependence on the long range (Figure 10).

Vector Auto Regression Model: Vector Auto Regression, known as VAR, is an extension of the univariate autoregressive model to multivariate time series data. The main idea of this model is that the value of a variable at a time point depends linearly on the value of different variables at previous instances of time.

Support Vector Regression model: The model that is produced by the Linear SVR depends on a subset of the training data. It discards any training data that is close to the models prediction. SVR works on similar principles as Support Vector Machine (SVM) classification. It can be argued that SVR is the adapted form of SVM when the dependent variable is numerical rather than categorical. SVM regression is considered a nonparametric technique because it relies on kernel functions. In addition, *Lr* represents the linear kernel for SVR. Also, there are two important parameters for SVR; parameter C, which is a penalty factor, and the parameter *epsilon*, whose value defines a margin of tolerance where no penalty is given to errors.

Gradient Boosting Regression model: The GBR computes a sequence of simple trees, where each successive tree is built for the prediction residuals of the preceding tree. The number of trees and the learning rate are key parameters for the GBR. The *n_estimators* is the limit number of trees in the forest and the *n_estimators* simply corresponds to the number of trees that will be fit in series to correct the prediction errors. Furthermore, the *learning_rate* shrinks the contribution of each tree by the *learning_rate* value. The *learning_rate* corresponds to how quickly the

error is corrected from each tree to the next.

Linear Regression Model: Linear Regression (LR) models the relationship between one or more input variable x and a dependent output variable y by using a linear equation [33]. The generic form of the LR model is

$$y_t = \beta_1 + \beta_2 x_t \tag{5}$$

Where y is the target variable, here it is the prediction workload. x is the explained variable, here it is the time. t indexes the sample interval. The coefficients β_1 , β_2 are determined by solving a linear regression equation based on the previous workloads y_{t-1}, y_{t-2}, y_{t-3}. According to the Cramer Rule, we can obtain the solution of linear simultaneous equations of β_1 , β_2, as shown below:

$$\beta_1 = \frac{\sum x_t^2 \sum y_t - \sum x_t \sum x_t y_t}{n \sum x_t^2 - (\sum x_t)^2} \tag{6}$$

$$\beta_2 = \frac{n \sum x_t y_t - \sum x_t \sum y_t}{n \sum x_t^2 - (\sum x_t)^2} \tag{7}$$

3.2.1 Model Selection Method

The Model Selection phase is a process to select a suitable model from candidate models and the information related to the performance of each model. The accuracy of machine learning models must be evaluated and the best performing model is selected for the prediction. For each model, we calculate three different performance metrics in order to measure their accuracy. We then compare the results and select the best model for online prediction. Meanwhile, the Boosting is a machine learning ensemble meta-algorithm for the group of machine learning algorithms which transform weak models to strong ones. A weak model is characterized as the one which is least accurate. A strong model is then a model that has given the best result. The Mean Absolute Error (MAE) and the Standard Root Mean Square Error (RMSE) measure accuracy and they calculate the difference between the predicted value and the actual value. A model with perfectly correct predictions would have an RMSE and MAE of 0. In addition, the Coefficient of Determination (R^2) is a measure of the proportion of variance of a predicted outcome with a value of 0 to 1. We calculate the model

accuracy noted as Score:

$$Score = MAE + RMSE + (1 - R^2) \tag{8}$$

Input: dataset F_{t_i}, $algorithm\_list : algorithm_1..algorithm_n$
Output: trained_model , score
hyperparameters_list: { $parameter_1 : [values]$,
$parameter_2 : [values]$,...,$parameter_n : [values]$ }
all_models = []
Divide F_{t_i} into two disjointed subsets *trainingSet* , *validationSet*
for *each* $algorithm_i$ *in range (algorithm_list)* **do**
    base_model ← $algorithm_i$, default_hyperparameters, initialized value of parameters
    training(*trainingSet*, gridsearchCV (base_model, hyperparameters_list))
    $model_i$ ← $algorithm_i$, optimal setting of hyperparameters_list, value of parameters
    validating(*validationSet* , $model_i$)
    $score_i \leftarrow MAE + RMSE + (1 - R^2)$
    all_model ← ⟨ $model_i$, $score_i$ ⟩
end
for *each* $model_i$ *in range (all_models)* **do**
    if $score_i \geq best\_score$ **then**
        best_score ← $score_i$
        best_model ← $model_i$
    end
end
trained_model ← best_model
score ← best_score
Return: trained_model , score

Algorithm 1: Unified Algorithm for Machine Learning Training

3.3 Architecture Design for Forecasting based Auto-scaling Process

In this section, we will explain how we can support the predictive approach for the auto-scaling process by the help of machine learning forecasting. e wil then describe how we can integrate workload forecasting with the auto-scaling process.

3.3.1 Original Auto-scaling process

Auto-scaling is a service to automatically scale up or down the number of resources based on demand at any time. The life-cycle of auto-scaling service includes five main parts (Figure 11): The *Auto Scaling Group* is responsible to group and manage

Table 2: Notations for Algorithm 1

Variables	Description
F_{t_i}	Weighted Metrics with Aggregatio as Input
algorithm_list	The list of algorithms used for training
hyperparameters_list	List of Tuning Parameters
trainingSet	The data is used for training
validationSet	The data is used for testing
all_models	List of models for each parameters
scores_list	List of scores for models
best_score	The best score result
best_model	The best model with best score

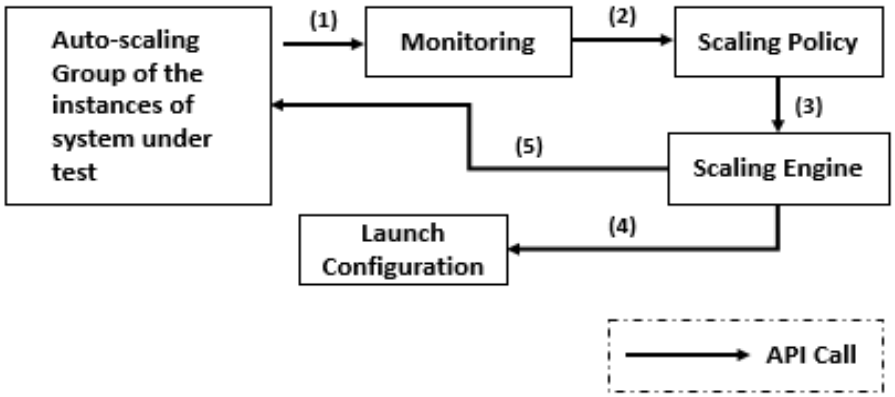


Figure 11: Original Auto-scaling Process

automatically instances. First(1) the *Monitor* checks the load on the *Auto-Scaling Group* based on defined metrics (CPU or memory utilization). (2) When the load for defined metric exceeds the configured threshold, the *Scaling Policy* is invoked by *Monitor* to check the defined policies for current situation. We have two types of policies: Threshold-based policy and Schedule-based policy. An example of the threshold-based policy would be if CPU usage reaches up to 80%, the *Scaling Engine* then needs to add more resources. While, for the schedule-based policy, the *Scaling Engine* adds more resources during Black Friday. By checking the policies, (3) the *Scaling Policy* communicates with the *Scaling Engine* so that it will take an action. (4) The *Scaling Engine* is required to check the *Launch configuration*, if there is a need for more resources. The *Launch configuration* provides all the necessary information that is required to instantiate instances. At the end, (5) the *Scaling Engine* adds or removes resources.

3.3.2 Forecasting based Auto-scaling Process

In order to support predictive approach for auto-scaling service, we need to forecast future demand by taking into account the workload history. Figure 12 illustrates our proposed machine learning-based forecasting for auto-scaling process. Each function and responsibility wraps as a module. So, each module encapsulates a set of related functions and several modules need to work together in order to complete the forecasting process. The proposed solution implements machine learning-based forecasting as a suite of small modules. Each module is described as follows:

Monitoring module: It is responsible for monitoring and collecting the system level metrics that represent the resource demand (1).

Controller module: It coordinates and manages other modules. By changing time intervals, the *Controller module* periodically invokes the *Monitoring module* to collect the workload metrics (2.1). The *Controller module* also calls the *Machine learning algorithms module* to train models (2.3) and calls the *Model selection module* to select a suitable model for prediction part (2.7). Also, it launches the *Prediction module*(2.9).

Data storage module: This module stores the monitoring data (2.2). The data collected from the *Monitoring module* becomes the training samples to learn a workload pattern for prediction and forecasting.

Machine learning algorithms module: It is responsible for using machine learning to train models (2.3). Each algorithm is trained in parallel and independently.

Validation module: This module evaluates the results of each algorithm (2.5). It is important for calculating model accuracy in machine learning. After training the models, the *Validation module* evaluates them to determine accuracy. Each model's trained parameters are saved to the *Model storage module* (2.6).

Model selection module: Validation scores are retrieved for each model from the *Model storage module* (2.8) and the *Model selection module* then selects the appropriate model for prediction (2.7).

Model storage module: It is responsible for keeping the information and data related to models. General data standard is used to deal with the wide variety of formats for datasets without having to be locked into a particular format. One of the popular data formats in cloud computing is a JSON which is what we use.to store the model information. For each model, we have one JSON file that contains the name

of the algorithm, the model ID, the value of parameters, the performance metrics results, and the score.

Prediction module:It is called by the *Controller module* to perform a real-time workload prediction upon predefined intervals (2.9). The *Prediction module* retrieves the selected model from the *Model storage module* (2.10). Finally, the prediction result is returned to the cloud auto-scaling service for resource prevision decision making.

The next step is to integrate the results of the prediction with the auto-scaling process. Most of the Cloud providers offer an auto-scaling service for a single metric, however, our proposed solution is based on multiple metrics. Thus, the solution cannot use the general auto-scaling service. We employ Ansible for provisioning and de-provisioning of resources. Ansible is one of the simplest ways to automate infrastructure and configuration management. Also, Ansible provides a lot of inbuilt modules for multiple Clouds, each of which we mapped. Ansible modules map to the *Auto Scaling Group*, the *Scaling Policy*, *Launch configuration*, and the *Scaling Engine*. Therefore, the prediction result is sent to the Ansible playbook which is *Scaling Policy* (2.11) in order to check the defined policies related to the results. After checking the policies, the *Scaling Policy* communicates with another Ansible playbook that is called the *Scaling Engine* to take an action (3). At the end, the *Scaling Engine* adds or removes resources (5).

3.3.3 Microservice Design

Each Cloud platform offers different types of services. We need to orchestrate and coordinate Cloud services in order to integrate a machine learning approach with an auto-scaling process. Each Cloud service needs to map onto one of the modules that are presented in Section 5.2. In the Cloud platform, each Cloud service represents an independent service that can be used to construct microservices architecture. The Cloud Service loose coupling promotes the independent design and evolution of a microservice implementation while still guaranteeing interoperability with other microservices. If each Cloud service function as an independent microservice, it means a process is serving as single independent function and deliver the responsibility. We have different techniques and implementation services on cloud platforms. So, microservices should be programming language agnostic, and each microservice should

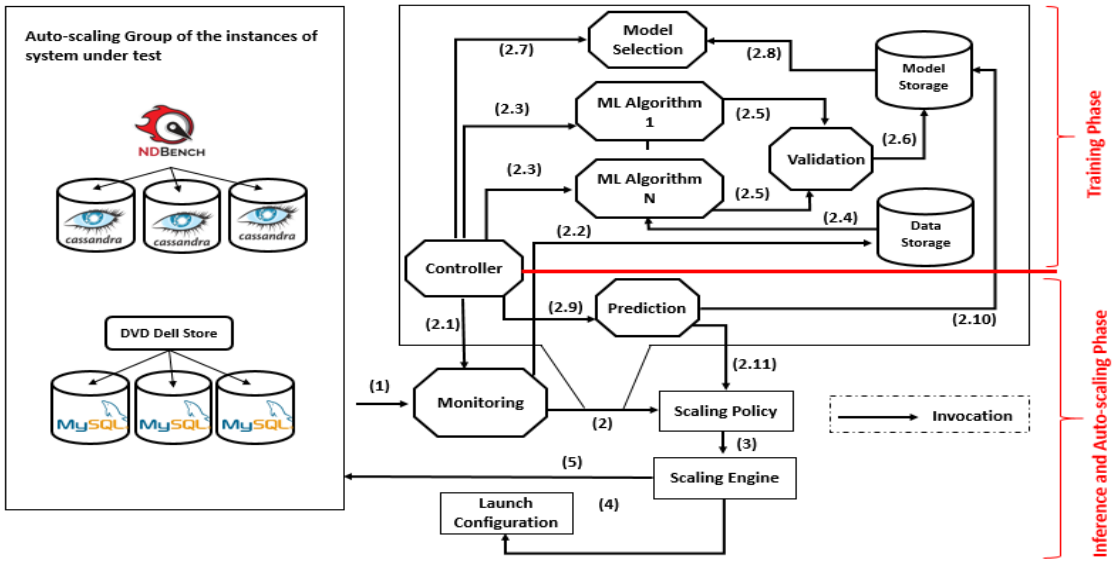


Figure 12: An Inference-based Forecasting for Auto-scaling Process

be able to function with the help of different languages including Python, Java, or R. In order to take advantage of using different cloud services, we should be able to scale each service from small to dedicated large size. At the core of this process, each module is encapsulated as a microservice. The microservices depicted as polygons in Figure 12 are independently deployed and running. The composition of the Cloud services are through their endpoints. Each microserivces life cycle is driven by requests for the service that originate from the need of model training and prediction. In our solution, all the requests are launched through a controller and then transferred to other microservices. The services are the **Monitoring service**, the **Controller service**, the **Data storage service and Model storage service**, the **Machine learning algorithms service**, the **Validation service**, and the **Model selection service**.

3.3.4 API Design

As the microservices represent different Cloud services that receive inputs, there are APIs to retrieve and present information requested from other microservices. The API is a way to access functions in each microservice. The API is required because we need to achieve a high level of separation and independence in our processes. Most of the Cloud providers offer REST API for their services. The interfaces between the

Table 3: Example of REST APIs

Operation	HTTP method
Get monitored metrics (2.1)	GET {base URL}/API/?metrics & metric-name = CPUUtilization & start-time=2018-04-12T14:18:00 & end-time=2018-04-12T16:18:00 & period=3600
Create new machine learning model (2.3)	POST {base URL}/API/?create-model & model-Name =GBR-Model & n_estimators = 1000 & learning_rate = 0.1 & Training-DataSource-Id=2018-04-12
Get selected model for prediction (2.7)	GET {base URL}/API/?selected_model & model-Names= LR-Model, SVR-Model, GBR-Model
Predict based on real-time metric (2.9)	POST {base URL}/API/?predict& 98, 67.34 , 33.10, 34.02 & modelid=LR-Model-v1

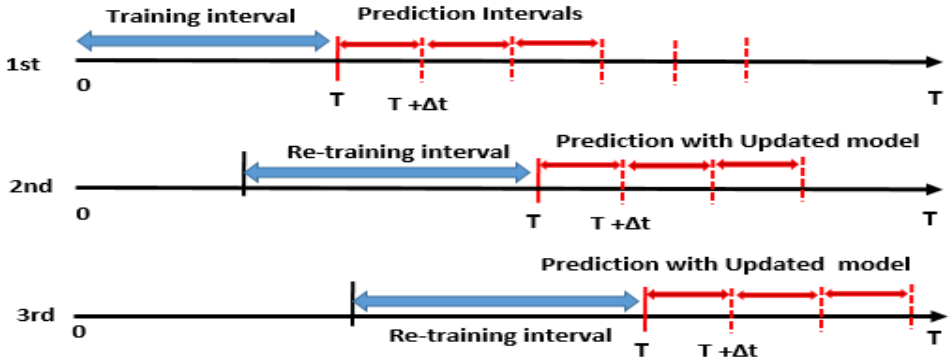


Figure 13: Continuous Training and Inference Phases

controller, monitoring, machine learning, model selection, and prediction are REST API. We selected REST because it is standard-based, lightweight, and can support multiple data representations. REST is a client-server architectural style for loosely coupled distributed systems. Table 3 summarizes the example of resources that we defined for our REST APIs.

3.3.5 Schedule-based Controller Service

The forecasting process includes four stages: Monitoring, Training, Prediction, and Re-training. First stage is monitoring the workload history at a time interval $[0, T]$. In the second stage , the workload history and the accumulated data that it holds in storage is moved to the training phase, and a new coming workload is stored for the next training phase.

We can have different sizes of time windows for training. For instance, two weeks, two days, or one hour. The advantage of the continuous training is that the model

has a higher probability of catching most recent workload patterns by training on the recent workload history. During the third stage, there is an iterative process that is able to predict real-time data and the data arrived for prediction in $T + \Delta t$ period. The fourth stage, entails re-training; for the subsequent training of the model, the window time is shifted and a model is trained and updated by new workload history. Our method is able to update the prediction model smoothly. Figure 13 demonstrates the continuous training and prediction intervals.

Therefore, there is a need to schedule multiple events and invoke APIs over different time periods for forecasting process. In this work, the *Controller* is a schedule-based service and communicates with the multiple services over different time periods. The *Controller* includes a time trigger to call APIs. For instance, when the training time is triggered, the *Controller service* calls the *Machine learning algorithms* to start model training stage for constructing, testing, and validating the models. In this work, we consider a 2 week time frame for training and the prediction stage is called every 10 minute repeatedly.

3.4 Evaluation of Integrating Machine Learning with Auto-scaling

The question is how to reduce the cost of the auto-scaling process through the use of forecasting of future load. The goal of this evaluation is to measure the run-time cost of machine learning-based forecasting for the auto-scaling process and demonstrate how the run-time cost is reduced thanks to the work we have conducted. We determine four metrics for evaluation:

- Multiple Applications: We implement multiple back-end applications to have different workload patterns for forecasting and calculate the performances.
- Multiple Cloud Platforms: The microservices of proposed forecasting process are mapped to the Cloud services and deploy the solution on AWS and Microsoft Azure Cloud platforms.
- Efficiency: The model training time and the prediction time are measured for NDBench and Dell DVD store applications on AWS and Microsoft Azure Cloud platforms.

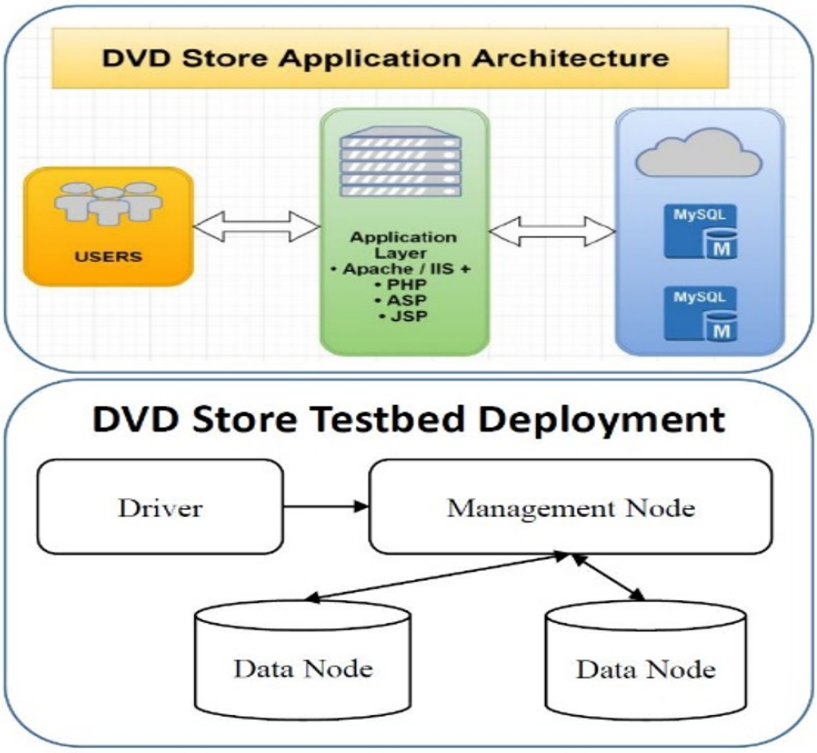


Figure 14: DVD store architecture

- Run-time Cost: We calculate the run-time cost and compare it with the cost of normal auto-scaling.

3.4.1 Demonstration Application

We implement two different back-end applications to have different workload patterns for forecasting and then calculate the performances.

Dell DVD store

DVD store is an intensive used benchmark and has been used in prior performance engineering research [53], [58], [23]. Dell DVD store (DS2) application [8] is an open source e-commerce test application that simulates an electronic commerce system to benchmark new hardware system installations. This application includes a back-end database component, a web application layer and, a driver program. Figure 14 demonstrates the general deployment architecture of the DVD store. This benchmark simulates online transaction processing of a DVD store. The goal is to design a

Table 4: AWS configuration of each node for Dell DVD store application

Instance	DVDStore	VCPU	Memory
Data Node 1	T2.medium	2	4GB
Data Node 2	T2.medium	2	4GB
Management Node	T2.medium	2	4GB
Driver	T2.Small	1	2GB

database component to utilize many advanced database features while keeping the database easy to install and understand. This application may be used to test a database or as a stress tool for any purpose. This application may be installed on Windows or Linux and it can use many different database programs. DVD store application is an application stack that can run on a single or multiple virtual machine. Table 4 listed the AWS configuration for Dell DVD store instances.

Netflix Data Benchmark

We use the Netflix Data Benchmark (NDBench) [14] to emulate the resource demand of a microservice. NDBench is designed to examine the microservices performance impact on the back-end data systems. NDBench is pluggable for a wide range of cloud providers. It works with Archaius for configuration, with Spectator for metrics, and with Eureka for service discovery. NDBench generates two types of loads; namely, random traffic and sliding window traffic. In our work, we adopt the sliding window traffic that generates realistic loads locally for the catching layers with disk input and output operations. We also write and read operations for each node in a networked cluster.

We have a separate instance for NDBench and connect to an Apache Cassandra cluster of data storage as a backend system (Figure 15). Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many servers, providing high availability with no single point of failure. Cassandra is a type of NoSQL database. Table 5 listed the Cassandra and NDBench instances on AWS and Azure Clouds.

3.4.2 Deployment Components on AWS and Azure

The following entities are deployed on the AWS (Figure 16):

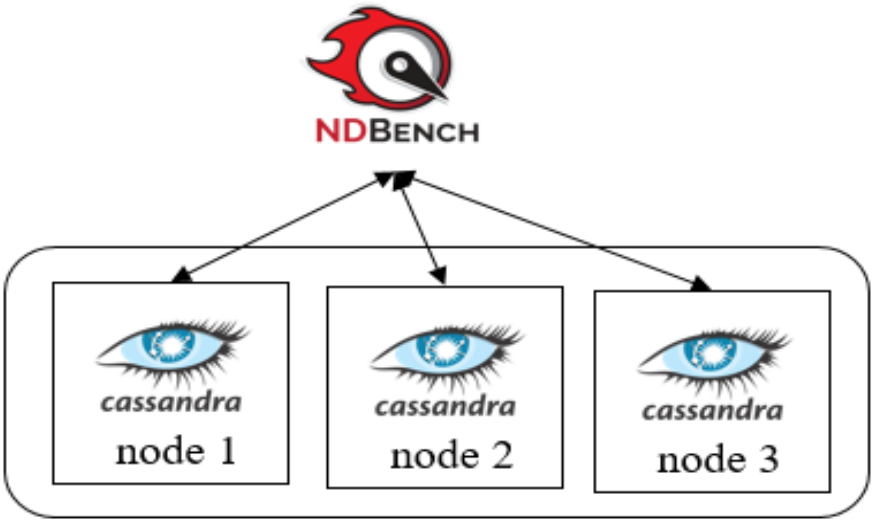


Figure 15: The NDBench Cluster with a Cassandra Cluster

Table 5: AWS and Azure configuration of NDBench node

In AWS Cloud Platform				In Azure Cloud Platform		
Name	Type	VCPU	Memory	Type	VCPU	Memory
Cassandra Node 1	M3.medium	2	4GB	DS1V2	1	4GB
Cassandra Node 2	M3.medium	2	4GB	DS1V2	1	4GB
Cassandra Node 3	M3.medium	2	4GB	DS1V2	1	4GB
NDBench Driver	T2.Small	1	2GB	DS1V2	1	4GB

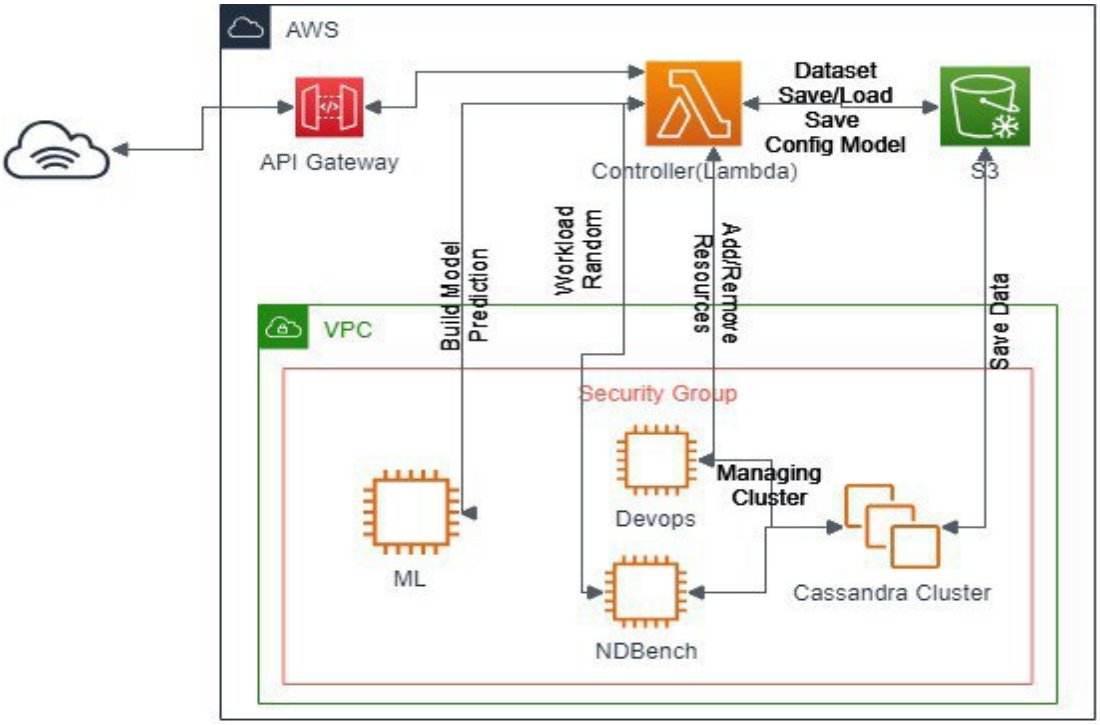


Figure 16: Deployment Architecture for AWS

- Lambda Function: The Lambda function employs a serverless architecture and the code runs without managing any servers or a backend service. The Lambda function calls CloudWatch to collect the workload metrics. Also, the Lambda function calls the machine learning API, to train the models, and the predictor, for the prediction.

- Machine Learning EC2 Instance: It hosts the training and prediction services. It is a fully-managed cloud instance for a predictive analytic solution. The workload history is pulled from S3 and utilized by a Machine Learning EC2 instance to train the models.

- Amazon S3 Storage: It stores the metrics collected by CloudWatch.

- DynamoDB Storage: It stores the training results including the performance scores, values of trained parameters, and configuration of the tuning parameters. These values are stored in a NoSQL storage with key/value pairs.

The following entities are deployed on Azure (Figure 17):

- Azure Function App: It periodically invokes the Azure Monitoring to collect the workload metrics. Also, the Azure Function App calls the machine learning algorithms to train models. Furthermore, it is the entry to launch the prediction service using a trained model.

- Azure Data Science Virtual Machine: As customized VM image on the Microsoft Azure cloud explicitly built for data science. Furthermore, we implement a REST API to access the inference model as a service for the real-time prediction phase. This service is called by the Azure Function App to perform training and prediction.

- Azure Blob Storage: It contains Blob storage, File storage, and Queue storage. For deployment, we use the Azure Blob Storage. The Azure Blob Storage saves the model training results and the collected metrics as inputs for training.

- Resource Group: It groups Azure instances (resources) for scaling and managing the instances based on the minimum and maximum number of running instances allowed at any time.

3.4.3 Tuning the Parameters of Machine Learning Algorithms

Optimizing the parameters of machine learning algorithms is an important task. In order to find the suitable values for each parameter, we measure performance metrics for each set of parameters and compare the results of the calculated scores. In concise terms, the parameters are a factor to be considered when selecting a best score for each machine learning algorithm as different parameters can present different performance results depending on the value of the parameters. In order to improve the performance of the SVR model, we need to adjust and select the best values for the SVR parameters. There are two important parameters for SVR: C takes 1 by default. Also, the parameter epsilon is a float and default value is 0.1. Table 6, 7 depict different values that we uses for tuning SVR algorithm. Results from the column *Score* of Table 6, 7 show the result of different scores when the value of C and *epsilon* are changed. Furthermore, the best value for C is 100 and for *epsilon* is $1e-5$.

For GBR algorithm, the default value for the *n_estimators* and the *learning_rate* respectively are 100 and 1. The parameter tuning is an important task in finding the

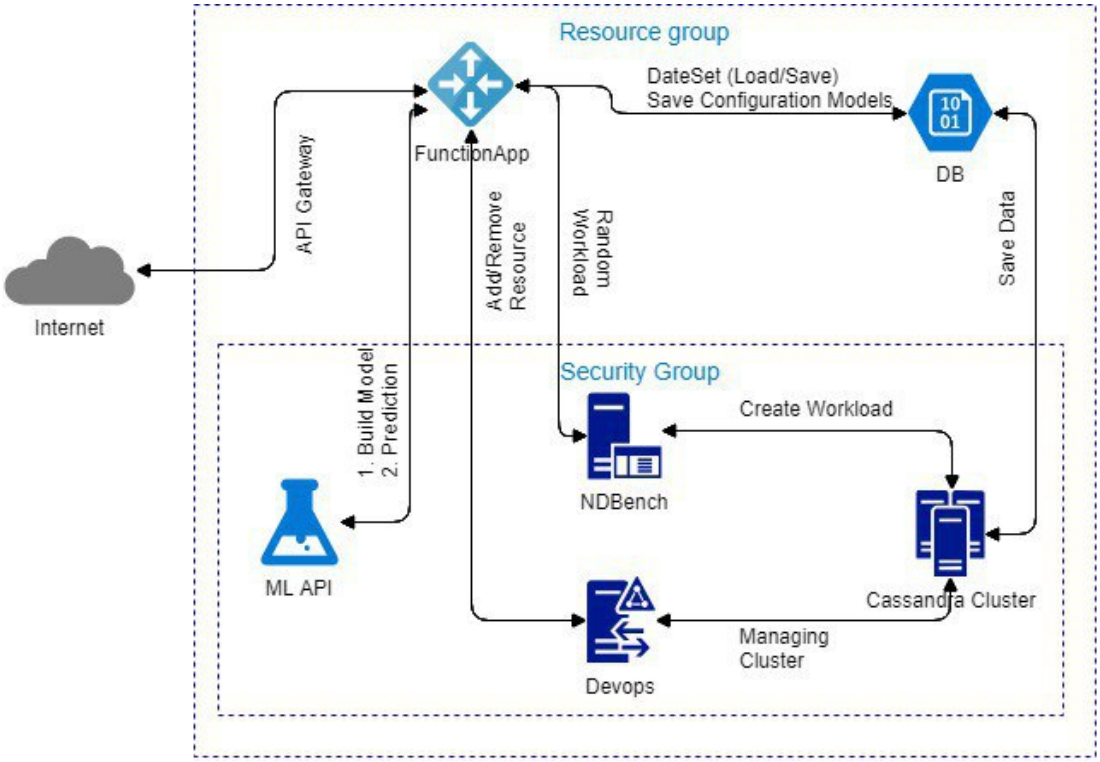


Figure 17: Deployment Architecture for Azure

model with the highest performance. So, we run different values for each parameter and find the appropriate performance results that are presented in Table 10, 11. Lowering the value of the learning rate and increasing the number of trees produces better performance. So, a larger number for the *n_estimators* results in better performance. We dont have specific parameters for linear regression algorithm, so we only calculate the performance metrics. The best performance results are presented in Table 8 and 9.

Table 6: Tuning SVR parameters on Azure

C	Epsilon	MAE	RMSE	R^2	Score
1	0.1	0.04486	0.048696	0.9999912	0.09357
20	0.02	0.04680	0.051773	0.9999905	0.09858
40	0.01	0.04483	0.050094	0.9999914	0.09493
60	0.001	0.04516	0.050144	0.9999911	0.09531
80	0.0001	0.04471	0.049709	0.9999913	0.09442
100	1.00E-05	0.04386	0.04861	0.9999914	0.09249

Table 7: Tuning SVR parameters on AWS

C	Epsilon	MAE	RMSE	R^2	Score
1	0.1	0.04514	0.049259	0.9999918	0.09464
20	0.02	0.04617	0.050541	0.9999911	0.09649
40	0.01	0.04513	0.050127	0.9999906	0.09421
60	0.001	0.04634	0.050692	0.9999915	0.09683
80	0.0001	0.04489	0.048999	0.9999904	0.09384
100	1.00E-05	0.04401	0.04742	0.9999901	0.09253

Table 8: Performance metrics results LR on Azure

MAE	RMSE	R^2	Score
0.000015435	0.000027958	0.9999956	0.000024492

Table 9: Performance metrics results LR on AWS

MAE	RMSE	R^2	Score
0.000010671	0.000021374	0.9999929	0.000034713

Table 10: Tuning GBR parameters on Azure

n_estimator	learning_rate	MAE	RMSE	R^2	Score
10	1	0.364528	0.572674	0.998830081	0.938372
10	0.1	0.359521	0.565862	0.998894623	0.926488
10	0.5	0.250691	0.419449	0.999397346	0.97014
100	1	0.302858	0.576622	0.998874099	0.970743
100	0.1	0.101868	0.211845	0.999845792	0.926488
100	0.5	0.175009	0.287338	0.999706122	0.97014
500	1	0.310088	0.562292	0.998910182	0.970743
500	0.1	0.082371	0.151487	0.99991855	0.933939
500	0.5	0.165836	0.30911	0.999672378	0.904344
1000	1	0.323547	0.579635	0.998838125	0.975274
1000	0.1	0.083081	0.157534	0.999914351	0.890701
1000	0.5	0.170609	0.282764	0.999716063	0.953657

Table 11: Tuning GBR parameters on AWS

n_estimator	learning_rate	MAE	RMSE	R^2	Score
10	1	0.356518	0.576821	0.998890081	0.949531
10	0.1	0.357134	0.565862	0.998787921	0.937529
10	0.5	0.249091	0.419449	0.999289263	0.9784329
100	1	0.309356	0.576622	0.998783212	0.963012
100	0.1	0.101868	0.297145	0.999845792	0.919875
100	0.5	0.167456	0.263338	0.999810439	0.98302
500	1	0.301658	0.569291	0.998890387	0.970743
500	0.1	0.088436	0.160382	0.999867132	0.940875
500	0.5	0.198463	0.308543	0.999816326	0.904344
1000	1	0.312474	0.569983	0.998728375	0.968432
1000	0.1	0.082975	0.162947	0.999920924	0.891948
1000	0.5	0.180131	0.275346	0.999728512	0.947048

Table 12: Hardware Specification of Machine Learning instances on AWS and Azure

Cloud Provider	Type	VCPU	Memory
Azure	B2s	2	4 GB
AWS	M3.medium	2	4 GB

3.4.4 Evaluation Results

For both Dell DVD store and NDBench, we produce one sample per minute and collect 20160 samples within the span of two weeks. We collect a total of 20160 samples of multi-metrics for training. Among these samples, 70% (14112 samples) are used for training and 30% (6048 samples) are used for validation. To setup the experiment environment for the machine learning algorithms and the prediction, we use AWS EC2 instance and the Azure Data Science Virtual Machine. In order to implement the inference process, we use the scikit-learn which is a free machine learning library for Python. The capacities of instances are presented in Table 12. At runtime, the models are constantly updated: whenever a new dataset for the next two weeks arrives, new models are created and new calculated scores are incorporated to the model storage and older scores are removed. The forecasting process is then repeated, which may lead to changes in the models, performance metrics, and score results.

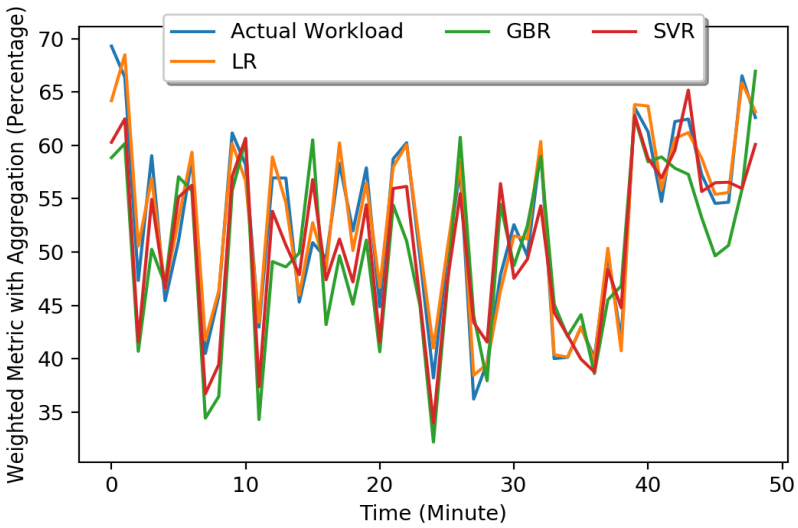


Figure 18: The prediction result of machine learning algorithms for Dell DVD store

Table 13: Training and prediction times for machine learning algorithms for Dell DVD store on AWS only

Technique	Training Time	Prediction Time
LR	1 min 20 sec	55 sec
SVR	17 min 22 sec	3 min 30 sec
GBR	37 min 16 sec	5 min 2 sec

Evaluation Results for Dell DVD store

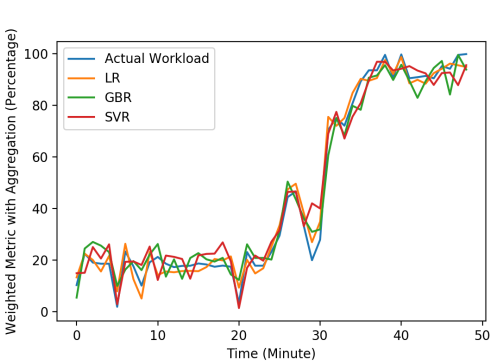
We implement the DVD store benchmark on AWS and generate workload. Figure 18 illustrates the graph of the actual workload and predicted results for LR, SVR, and GBR. In addition, Table 13 displays the time measured for the model training and the prediction on AWS environment for DVD Dell store.

Evaluation Results for NDBench

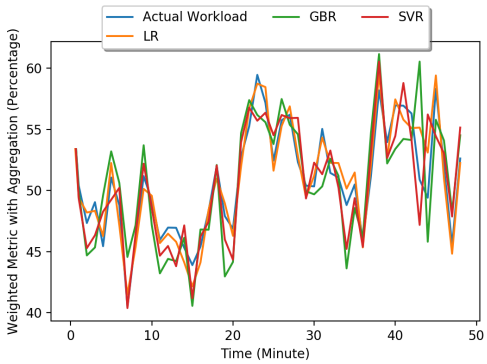
We implement NDBench on AWS and Azure Cloud environments. Table 14 represents the time that we measure for the model training and the prediction on Azure and AWS environments for NDBench. Figure 19 displays the graph of the actual and predicted final target for LR, SVR and GBR on Azure and AWS cloud platforms. The LR model achieves better prediction results in comparison to the other models. As seen in Figure 19, the red line relating to the LR model prediction result, is in higher

Table 14: Training and prediction times for machine learning algorithms on AWS and Azure for NDBench

	In AWS Cloud Platform		In Azure Cloud Platform	
Technique	Training Time (min)	Prediction Time (min)	Training Time (min)	Prediction Time (min)
LR	2 min 6 sec	45 sec	1 min 15 sec	1 min
SVR	16 min 15 sec	4 min 42 sec	14 min 53 sec	03 min 18 sec
GBR	38 min 41 sec	6 min 29 sec	46 min 11 sec	5 min 13 sec



(a) The prediction result of machine learning algorithms on Azure



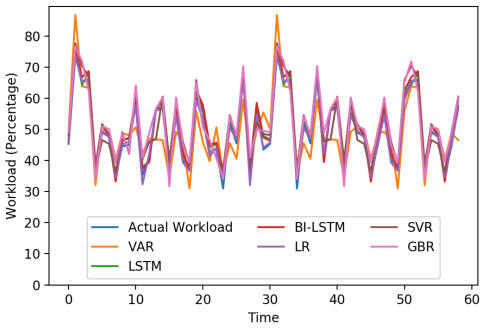
(b) The prediction result of machine learning algorithms on AWS

Figure 19: The prediction result of machine learning algorithms on AWS and Azure for NDBench

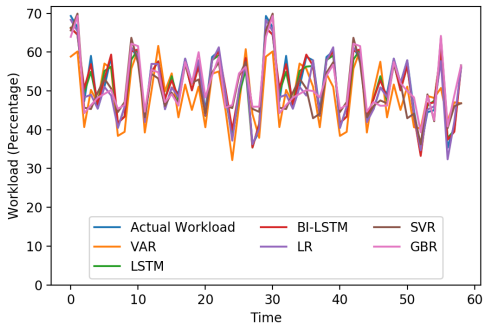
conformity with the blue line which is related to Weighted Metrics with Aggregation (F_{t_i}). The SVR can be considered as the second-best model for prediction. Except for slightly large variations from F_{t_i} in a few number of points, the SVR tracks F_{t_i} variations suitably. Figure 20 illustrates the graph of the actual workload and predicted results for VAR, LSTM, BI-LSTM, LR, SVR, and GBR.

3.4.5 Resource Provision and De-provision

While using an auto-scaling service on AWS and Azure, one of the issues is the new added instance does not configure and join the cluster properly. Thus, adding a new instance alone is not enough for applications. There are some configuration steps required after adding a new instance in order to manage the increased workload. So, there are still some manual tasks performed by application providers to handle unexpected workload.



(a) The prediction results for NDBench



(b) The prediction results for DVD store

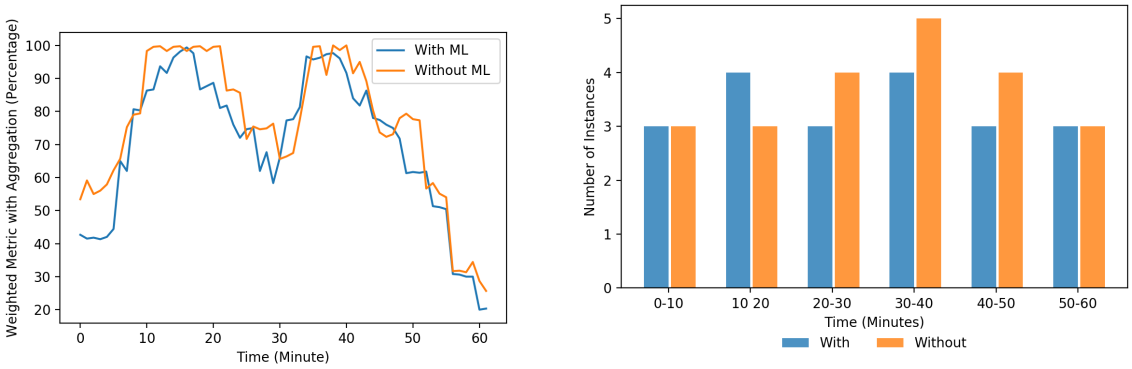
Figure 20: The prediction result of machine learning algorithms (VAR, LSTM, BI-LSTM, LR, SVR, GBR) for NDBench and DVD store

Table 15: Score Results for NDBench on AWS

	MAE	RMSE	R^2
VAR	0.864528	0.472674	0.9588300
LSTM	0.290691	0.179449	0.9999973
BI-LSTM	0.302354	0.179635	0.9999209
LR	0.106718	0.021376	0.9999929
SVR	0.340163	0.117421	0.9999601
GBR	0.529759	0.262947	0.9988381

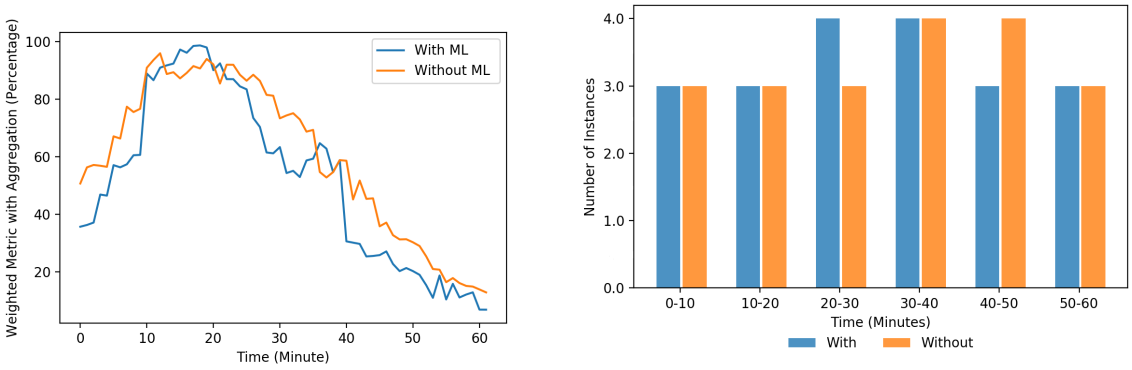
Table 16: Score Results for DVD Store on AWS

	MAE	RMSE	R^2
VAR	0.749091	0.419449	0.9982892
LSTM	0.209356	0.176622	0.9997832
BI-LSTM	0.201868	0.197145	0.9998457
LR	0.167456	0.096333	0.9999104
SVR	0.501658	0.269291	0.9998903
GBR	0.884361	0.460382	0.9988671



(a) Changes in workload over time on Azure for NDBench

(b) Azure instances for a Microservice change over time for NDBench

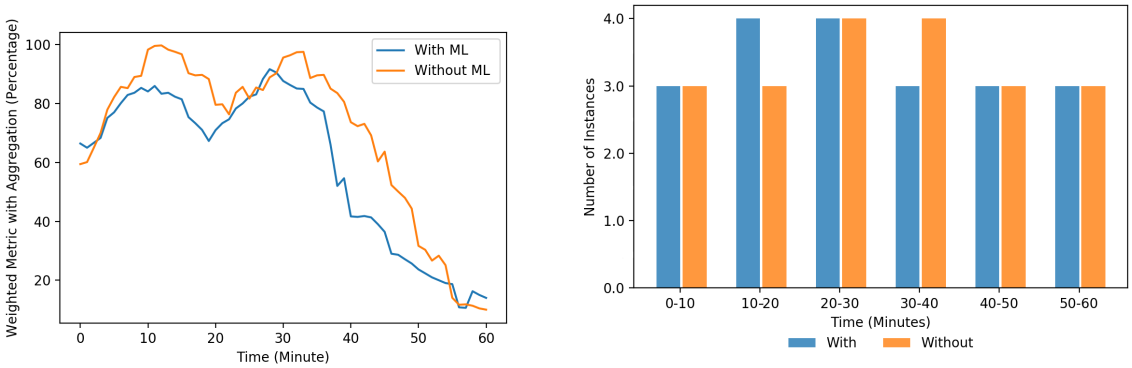


(c) Changes in workload over time on AWS for NDBench

(d) AWS instances for a Microservice change over time for NDBench

Figure 21: Resource Provision and De-provision on AWS and Azure NDBench

We employ Ansible to automatically configure the new instance and join it to the cluster. When the prediction service decides (Figures 48 and 43),the Ansible is invoked in order to add or remove resources based on predicted workload. In addition, Ansible must re-balance a cluster because a new instance assumes management of this portion of the data. We consider two scenarios for evaluating our approach. In the first scenario, we deploy normal threshold-based auto-scaling for CPU usage and set window times, which is the amount of time monitored before the metric and threshold values are compared, for 10 minutes. The threshold is set at 80% for adding more resources and at 30% for removing idle resources. We then compare this result with the result of our solution of applying machine learning prediction for multi-metrics and its integration with Ansible for resource provision and de-provision. In addition, we assume the time frame for the prediction is 10 minutes.



(a) Changes in workload over time on AWS for Dell DVD store

(b) AWS instances for a Microservice change over time for Dell DVD store

Figure 22: Resource Provision and De-provision on AWS for Dell DVD store

In addition, Figure 19 demonstrates the comparison between two aforementioned scenarios for NDBench on AWS and Azure. Figure 21b reflects the number of Azure instances that are added and removed based on the workload that we present in Figure 21a. As we can see in the Figure 21b, the final target usage exceeds the threshold in the second 10 minutes for our method (blue line). The workload goes under the threshold at Time 26 which after resources are added. However, for the orange line, which is the normal auto-scaling solution, the resources are added and the workload goes down under threshold at Time 31. In our method, Azure VM spin up approximately around 5 minutes and Ansible needs around 3 minutes to reconfigure the cluster. Whereas, in normal auto-scaling, we have 5 minutes for a spin up VM and about 5 minutes or more for manual configurations. This experiment under the workload generated by NDBench on Azure cloud environment shows our method is more proactive than the auto-scaling group by default for a single metric. Figures 22a demonstrates the comparison between two mentioned scenarios for the Dell DVD store and also Figures 22b illustrates the number of AWS instances that are added and removed based on the workload.

3.4.6 Cost Estimation

The cost is the price charged by cloud providers for the service utilization in a specific time period. The cost estimation includes the workload cost and inference cost. Table 18 demonstrates the types of cost. We assume the workload has an associated

Table 17: The Run-time Cost Comparison for Applications on Multi-Clouds for Duration of 1 hour

Application	Cloud platform	Instance Usage / With ML (billing time unite)	Cost /With ML($)	Instance Usage /WO ML (billing time unite)	Cost //WO ML ($)	Saved
NDBench	AWS(2.25$/h)	11.06	24.88$	11.93	26.84$	1.96$/h
NDBench	Azure(0.65$/h)	11.43	7.42$	13.09	9.03$	1.61$/h
DVD Dell	AWS(2.25$/h)	11.27	25.35$	12.62	28.39$	3.04$/h

Table 18: The Type of Costs for Cloud Services

Cost Type	Service Used
Workload Cost	Virtual Instance
Inference Cost	Monitoring Model Storage Training Prediction

cost, which depends on the virtual instance in which the workload run. The *Cost (workload)* presents the cost of workload. The inference procedure includes monitoring, training, model selection, and prediction. So, the inference cost refers to the cost of cloud services that are part of the inference procedure. In our work, we calculate a metric as the workload cost to compare each method of provisioning resources. Assume T is the total time of evaluation, and I_i is the number of the virtual instance at the time duration t_i. The p_i is the charge rate per billing time unite, and the τ is the billing time. Thus, the workload cost for each application on multiple clouds is calculated as equation 9.

$$Cost(workload) = \sum_{i=1}^{T}(p_i * I_i * (t_i/\tau)) \tag{9}$$

We assume the p_i is constant because the price of instances that are used for deploying the applications on AWS and Azure does not change. So we have:

$$Cost(workload) = p_i * \sum_{i=1}^{T}(I_i * (t_i/\tau)) \tag{10}$$

We assume T is one hour (60 minutes) and τ is billing time for one hour. In addition, we consider the time period (t_i) as 10 minutes. For example, referring back to Figure 9b (for NDBench on AWS), The p_i is 0.65$ per hour for virtual instance. We calculate the instance usage for one hour is 13.09 of using auto-scaling directly without

using machine learning. So the cost is 9.03$ while the cost of our machine learning method is 7.42$ for one hour. Thus, our solution is saving 1.61$ for one hour, and we suppose our solution save 38.64$ for 24 hours and 14111.65$ for 8765 hours. It means our method helps the system to use less cost in total compared to the default auto-scaling method. Table 17 demonstrates a comparison between the costs for NDBencha and Dell DVD store on AWS and Azure. The cost of instances are different on AWS and Azure, so we have a different result on each Cloud environment. The instance that we use for AWS is more expensive than the Azure instance. However, the comparison shows we save a significant amount of money for both Cloud platforms. The inference costs for AWS and Azure are 2.68$ and 0.21$, respectively. The inference cost looks high because we deploy our forecasting process on the separate instance and the cost of that instance also needs to be added. However, in our work, the size of the cluster is small and includes 3 instances. Although, for a large size cluster, inference cost does not increase but the computing cost changes. For instance, for a cluster with 10 large EC2 instances on AWS, the inference cost remains unchanged, but the saved cost increases significantly.

3.5 Summary

In this chapter, we answered the first research question:

RQ 1. How to design an architecture solution to orchestrate the machine learning process?

Motivation: In order to support a predictive approach for auto-scaling service, we need to forecast future demand by taking into account the workload history. Each cloud platform offers different types of services. We need to orchestrate and coordinate cloud services to integrate machine learning approach with auto-scaling process.

Approach: We introduced a new solution to train models based on multi-metrics. In order to calculate the effects of multi-metrics, we used Shannon Entropy to compute the weight of each metric. Also, we presented the architecture based on microservice approach for a forecasting-based auto-scaling process that adaptively monitors the workload based on multi-metrics and schedules multiple machine learning models to learn the workload pattern online and predict the workload classification at runtime. To evaluate, we used Dell DVD Store and Netflix Data Benchmarks and applied the

proposed solution on Amazon Web Services and Azure cloud environment. In addition, we deployed six machine learning regression algorithms on cloud environments. We demonstrated that the real-time prediction is integrated to the auto-scaling configuration of a cloud infrastructure to add or remove computing resources.

Chapter 4

Model-Driven Architecture

In this chapter, we present a model-driven approach for the predictive auto-scaling. We will explain why we need a meta model for machine learning service. Also, we introduce the overview of model-based configuration and deployment process. Then, we describe CPIM and CPSM. Finally, we discuss how to combine the Model-driven approach with DevOps.

4.1 Model Driven Approach

The proposed method captures the predictive auto-scaling system including its components, connections, and resources. The proposed method relies on a model-driven approach which is a branch of software engineering that focuses on models rather than source code and improves the service deployment. The proposed model-driven framework consists of three main phases: Design, Deployment, and Run-time (Figure 23).

The *Design phase* includes the model-to-model transformation. This phase relies on types of models representing two layers of abstraction, the Cloud Platform Independent Model (CPIM) and the Cloud Platform Specific Model (CPSM). The CPIM specifies the provisioning and deployment in a cloud provider-agnostic way. The CPSM refines the CPIM with cloud provider-specific information. Model transformation automatically generates parts of deployment scripts. We are using Eclipse Modeling Framework (EMF) for the Design phase. EMF is the core modeling framework and code generation facility for building tools and applications based on models

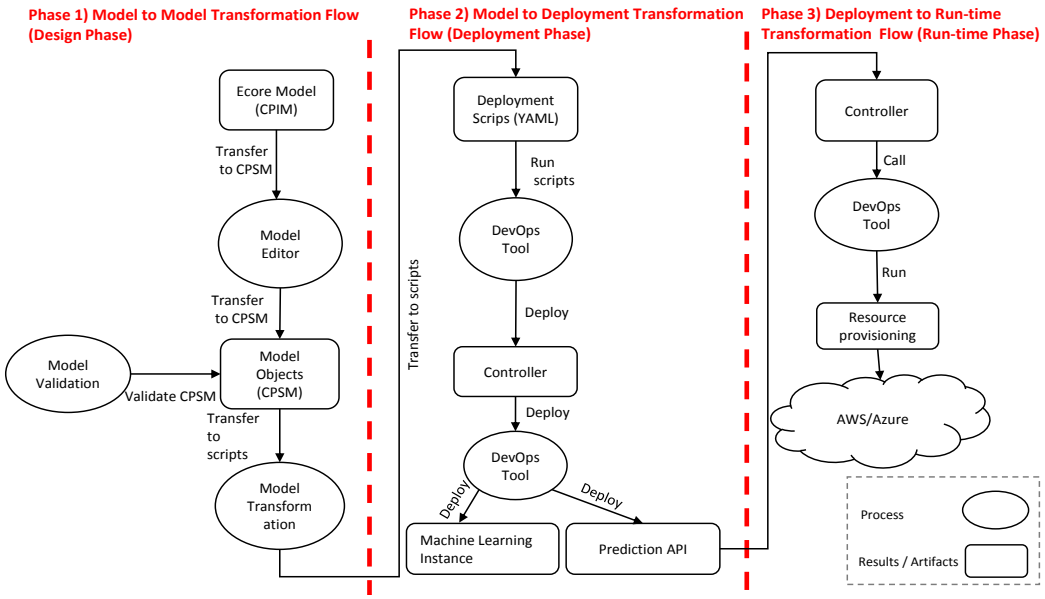


Figure 23: Model Transformation in the Life-cycle of Auto-scaling

defined in the ecore meta model. The *Deployment phase* includes the process of automatic deployment of the predictive auto-scaling system on a target platform. The deployment scripts provide a description of the most important classes and corresponding properties of the target. The *Run-time phase* phase triggers the execution of deployed machine learning components upon the collection of the system level metrics.

4.2 Cloud Platform Independent Model

Cloud Platform Independent Model (CPIM) focuses on the operation of the system while hiding details related to the use of a particular platform. CPIM maintains platform independence to be suitable for use with different platforms. The CPIM in an ecore model follows the model-driven principles to create an abstract representation of the knowledge and activities in the context of a predictive auto-scaling system.

4.2.1 Machine Learning Abstraction

A machine learning process includes training and inference phases. Training refers to the process of fitting a machine learning model by optimal model parameters from

the sampling data. Inference refers to the process of running a trained model to make predictions. In Chapter 3, we review the basics of six different machine learning algorithms.

Machine Learning Model Elements Commonality

A machine learning process includes training and inference phases. Training refers to the process of fitting a machine learning model by optimal model parameters from the sampling data. Inference refers to the process of running a trained model to make predictions. For both training and inference, there are common entities: (1) inputs, (2) outputs, (3) parameters, and (4) assessment metrics. A machine learning model requires a set of inputs and outputs. The inputs are divided into two parts: the training dataset and testing dataset. Outputs are a set of data that is being predicted by the trained model. For example, in cloud services, inputs can be low-level CPU, memory or network usage, or higher-level kinds of data tied to the services or applications, such as requests served per second. For data of distinct categories that are non-numerical, data is first encoded into a numerical form by feature engineering techniques. Hence, in our work, we consider input and output as numeric data and time series type. Parameters include hyper-parameters and model parameters. Hyper-parameters are configuration variables that are typically searched by greedy algorithms. To check the accuracy training and inference results, assessment metrics are necessary. The assessment metrics are also used for the model selection that is the process of selecting a suitable model from a set of candidate models. The value of assessment metrics is also numerically applied. In conclusion, entities are in different scales and characters. The integration of machine learning and auto-scaling should plug-and-play different machine models in a uniform way rather than model-by-model.

Machine Learning Model Environments Commonality

Machine learning model environment includes frameworks, programming languages, and libraries. There are three popular languages: Python, R, and MATLAB. Python is a programming language that consists of a large standard library. This library is structured to focus on general programming and contains modules for OS specific

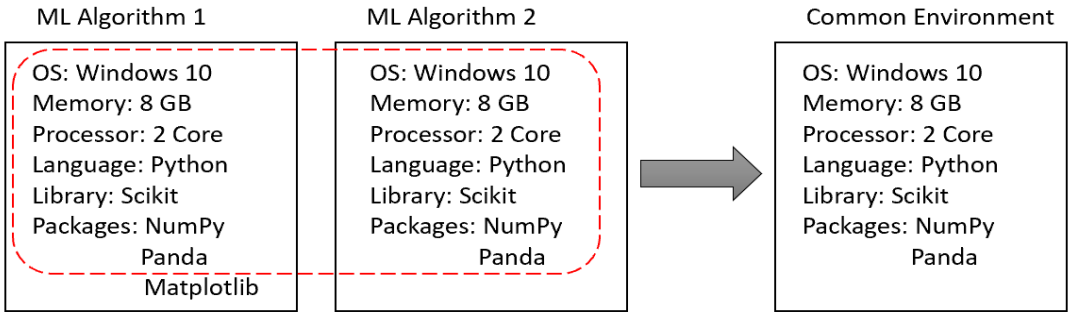


Figure 24: An Example of the Machine Learning Environment Commonality

threading, networking, and databases. R is the most comprehensive statistical analysis package available and incorporates all of the standard statistical tests, models, and analyses, as well as provides a comprehensive language for managing big data. Finally, MATLAB is a commercial numerical computing environment and a programming language. MATLAB similarly has a standard library, but its uses include matrix algebra and a large network for data processing and plotting.

Libraries are sets of functions that are written in a given language. A robust set of libraries can make it easier for developers to perform complex tasks without rewriting many lines of code. Scikit-learn is one of the most popular machine learning libraries. It supports many supervised and unsupervised learning algorithms. Tensorflow is another machine learning library and contains several deep learning and neural network algorithms. Figure 24 provides an example of two machine learning algorithms with different configurations. The commonality of configurations is used to specify the same setting between machine learning models of certain group of algorithms. The use of the common environment for different machine learning algorithms leads to a reduction in production time and improved deployment process.

4.2.2 Machine Learning Meta Model

The auto-scaling system needs to decide when to perform the scaling actions and it commonly performs actions reactively when workload change has already occurred. The actions are based on the predefined rules of the analysis phase which are static. There is a need to predictively provide resources ahead of workload changes. So, we are focusing on enhancing the analysis phase with a machine learning approach

Table 19: Example of Machine Learning Algorithms for Workload Forecasting

Algorithm Type	Version	Input	Tuning Parameters	Output	Score
Support Vector Regression	V_{ts}	I_{t_i}	-C (Penalty Factor) -*Epsilone* (Margin of Tolerance)	-Model ID -Parameters Value list	-MAE -$RMSE$ R^2
Gradient Boosting Regression	V_{ts}	I_{t_i}	-$N\_estimators$ (Limit number of trees) -$Learning\_rate$ (How quickly the error is corrected)	-Model ID -Parameters Value list	-MAE -$RMSE$ -R^2
Linear Regression	V_{ts}	I_{t_i}	– –	-Model ID -Parameters Value list	-MAE -$RMSE$ -R^2
LSTM [1]	V_{ts}	I_{t_i}	- $HiddenLayerSize$ - $HiddenUniteSize$ - $Learning\_rate$	-Model ID -Parameters Value list	-MAE -$RMSE$ -R^2
Bi-LSTM [1]	V_{ts}	I_{t_i}	- $HiddenLayerSize$ - $HiddenUniteSize$ - $Learning\_rate$	-Model ID -Parameters Value list	-MAE -$RMSE$ -R^2
Vector Auto Regression	V_{ts}	I_{t_i}	- $ErrorVector$	-Model ID -Parameters Value list	-MAE -$RMSE$ -R^2

instead of heuristic rules. Also, we plan to automate the machine learning process into the auto-scaling system. For the rest of this section, we try to answer the following questions:

- How to develop meta model for machine learning in general?
- Do we need new data types?

The machine learning service includes two main phases: training and inference. In both phases, we have a set of common entities for machine learning algorithms. In Table 19, we list different machine learning algorithms and demonstrate key common entities of the machine learning algorithms. There are common elements across algorithms which are version, input, output, tuning parameters, and score.

.

To support a wide range of machine learning models, we need to generalize the common entities and represent them at an abstract level. In our work, we propose a

[1]Default parameters are explained here, other optional parameters are explained in Section 4.3.1

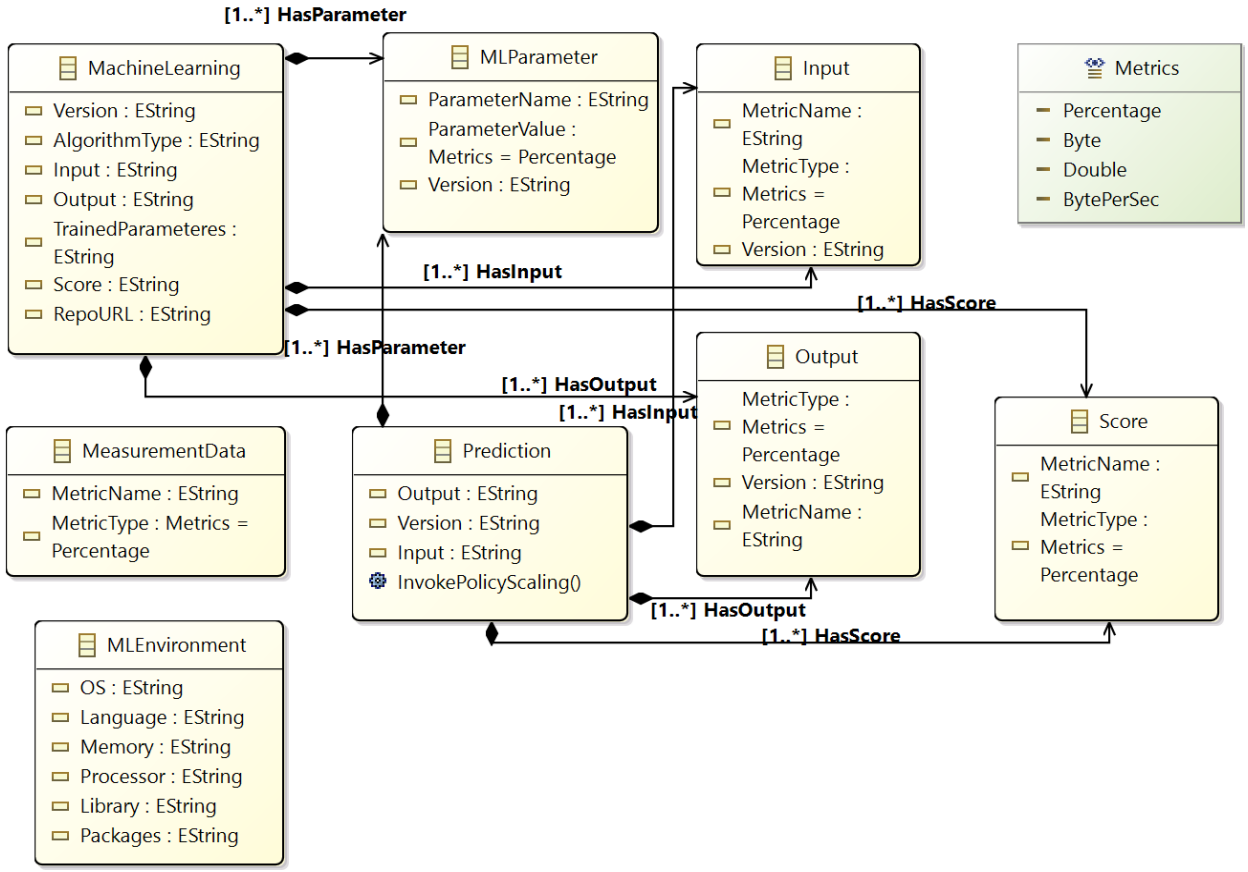


Figure 25: Meta-model of Machine Learning Model

meta model to represent the common entities of a machine learning process. Hence, a specific machine learning model becomes an instance of the machine learning meta model. In Figure 25, we try to decompose machine learning into reusable and independent small elements, which we refer to as machine learning meta model elements. We propose higher-level abstractions to model the machine learning itself. Therefore, the meta model will be applied to different machine learning algorithms.

- **MachineLearning:** This component is responsible for using machine learning algorithms to train models. Each algorithm is trained in parallel and independently. Then, it evaluates the results of each algorithm to determine accuracy. At the end, it selects the appropriate model for prediction.

- **MLparameter:** Tuning parameters differ from the model parameters in that they are not learned by the model automatically through training. Instead,

these parameters, such as learning rate, are related to how fast the machine learning model converges for a solution. Therefore, tuning parameters should be explicitly modeled. Each time we change a tuning parameter, we have a new instance of the machine learning model. To distinguish each setting of turning parameters tried, we introduce the version as an essential attribute for modeling in addition to the intrinsic elements of machine learning as inputs, outputs, and scores.

- **Input:** Machine learning entities require a set of inputs and outputs. Inputs are a set of data that are used by machine learning algorithms to train the model. The inputs are divided into two parts: the training dataset and testing dataset. We consider the input as time series data.

- **Output:** Outputs are a set of data that are being predicted by the trained model. For example, in cloud services, outputs can be low-level usage provided by infrastructure, or higher-level types of data tied to the service or application, such as requests served per second. Input and output values are numeric data and time series type. Data generally includes distinct categories, which are non-numerical and thus need to be converted to a numerical format by data processing techniques.

- **Score:** To check the accuracy of the training and inference, assessment metrics (Score) are necessary. The assessment metrics are used for the model selection phase which is a process to select a suitable model from candidate models and from given information related to the performance of each model. The accuracy of machine learning models needs to be evaluated and the best performing model is selected for the prediction. The value of assessment metrics is numeric and ordering is not applied.

- **Prediction:** It is called to perform a real-time workload prediction. The prediction result is returned to the cloud auto-scaling service for resource provision decision making.

- **MeasurementData:** This component represents the metrics that we need to collect for the Monitor component.

- **MLEnvironment:** It represents the commonality of configurations required between machine learning models of certain group of algorithms.

- **Metrics:** The numeric values of inputs, outputs, tuning parameters, and scores are of different types and scales. The values of inputs are time series, and ordering is essential; while the value of performance scores are singular real numbers. The outputs are trained values of model parameters that are also real numbers. The output also contains inference results as the probability that are real numbers between 0 to 1. To customize these variations of data types, one possible solution is using the default data types with constraints attached. However, this way lacks of explicitly in the first order of modeling, as the characters of the data type depending on the constraints defined behind. Instead, we introduce a data type, called *Metrics* for representing metrics of which each attribute is of a specific type. The attribute *Percentage* is of double type in the range of 0 to 1. The *Byte* attribute has the type of positive long values. The attribute *Double* is used for values of parameters and scores. Finally, the attribute *BytePerSec* is used to capture metrics of rates, such as *NetworkIn* and *NetworkOut* system level network communication metrics.

4.2.3 Auto-scaling Process Abstraction

The life-cycle of an auto-scaling service includes five main parts (Figure 26): The Auto Scaling Group is responsible for grouping and managing instances automatically. The Monitor (1) checks the load on the Auto-Scaling Group based on defined metrics (CPU or memory utilization). When the load for the defined metric exceeds the configured threshold, the Scaling Policy (2) is invoked by the Monitor to check the defined policies for the current situation. We have two types of policies: a threshold-based policy and a schedule-based policy. For the threshold-based policy, if CPU usage reaches up to 80%, the Scaling Engine needs to add more resources. For the schedule-based policy, the Scaling Engine adds more resources during the Black Friday. By checking the policies, the Scaling Policy (3) communicates with Scaling Engine to take an action. The Scaling Engine (4) is required to check the Launch configuration if there is a need for more resources.

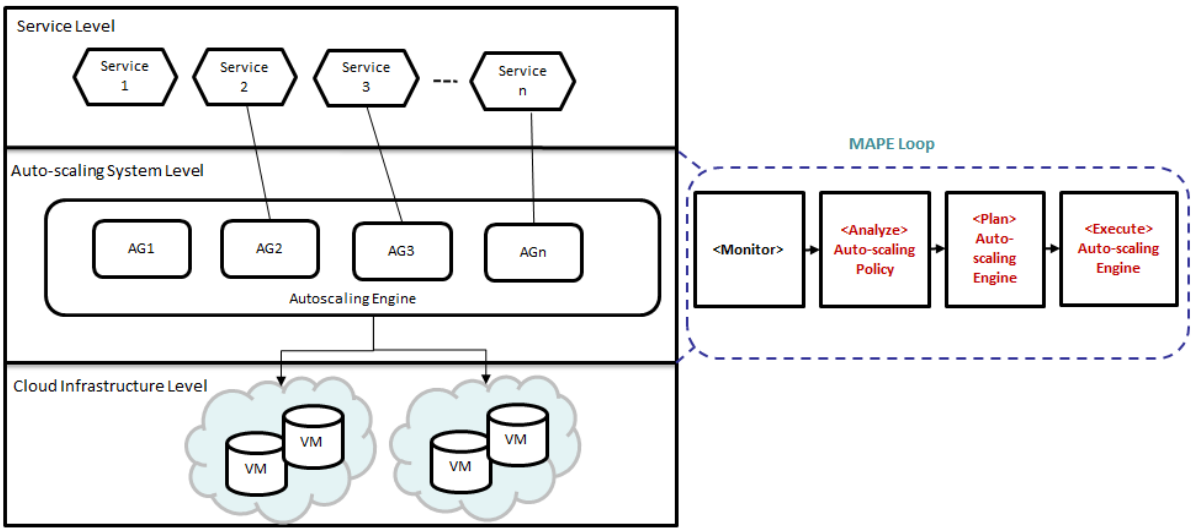


Figure 26: Predictive Auto-scaling System based on Monitor-Analyze-Plan-Execute (MAPE) loop

4.2.4 Auto-scaling System Meta Model

In this section, we present the meta model of an auto-scaling system. In this model, we identify features of auto-scaling as monitoring, scaling out, scaling in, and the auto-scaling group. We focus on the techniques, derived functions and associated connections of each feature. We build a meta model to capture these artifacts shown in Figure 27.

The resulting model provides a CPIM that represents the main auto-scaling components. The CPIM components and their responsibilities are as follows:

- **AutoScalingGroup**: This represents a group of virtual machine instances that the application deployed on. *AutoScalingGroup* has a launch configuration and a list of virtual machine instances. It is related to the *Monitor* component for sending the states of instances. *AutoScalingGroup* communicates with the *LoadBalancer* component to allow *LoadBalancer* automatically spreads incoming traffic across the instances in *AutoScalingGroup*.

- **Monitor**: collects states (or metrics) of virtual machine instances within an *AutoScalingGroup*.

- **PolicyScaling**:The current auto-scaling systems are following two approaches:

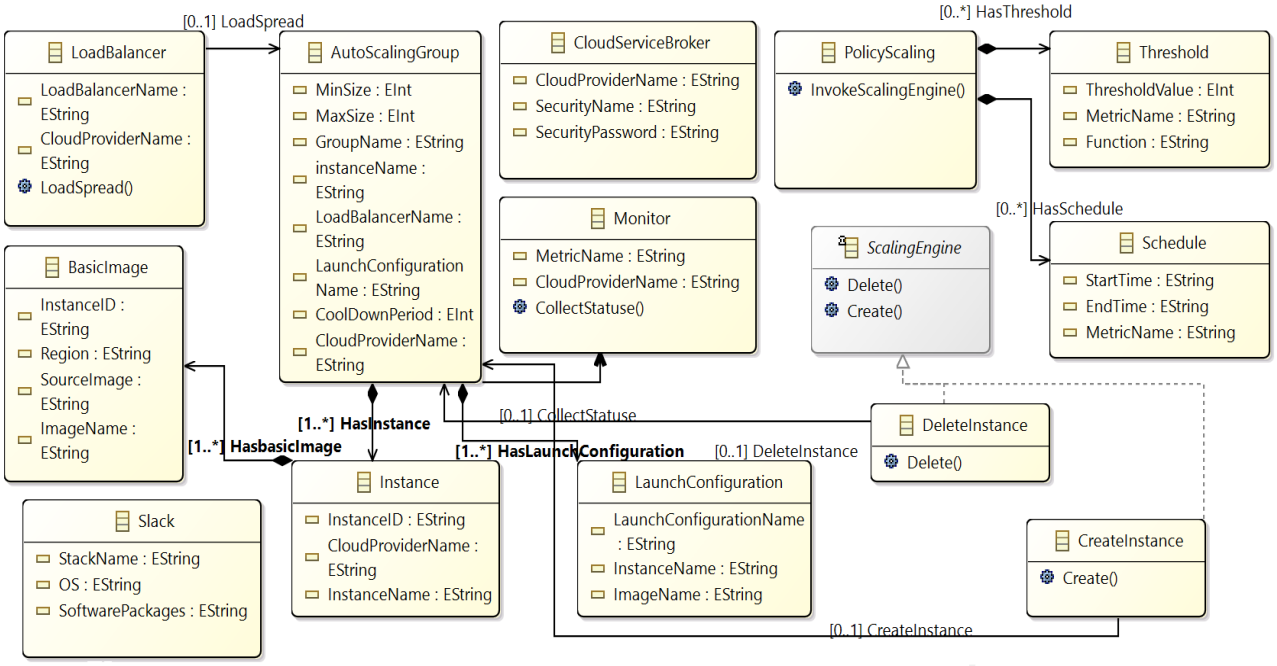


Figure 27: Cloud Platform Independent Model for Auto-scaling System

threshold-based and schedule-based. *PolicyScaling* inputs to *ScalingEngine* on actions such as scaling in or scaling out.

- **ScalingEngine**: This acts on the decision from *PolicyScaling* to create or delete virtual machine instances.
- **CloudServiceBroker**: This specifies the cloud platform name, security-username and security-password. This information is used to create security credentials to access the target cloud platform.
- **LoadBalancer**: This is responsible for controlling the load spread equally across virtual machine instances in an *AutoScalingGroup*.
- **LaunchConfiguration**: an essential entity to instantiate a new instance in *AutoScalingGroup*. It contains primary configuration information.
- **Instance**: This represents the virtual resource of cloud computing. It can be storage, computing or network resources.
- **Stack**: This contains a list of software package information. Each software is installed on an image that is eventually launched on a newly provisioned node.

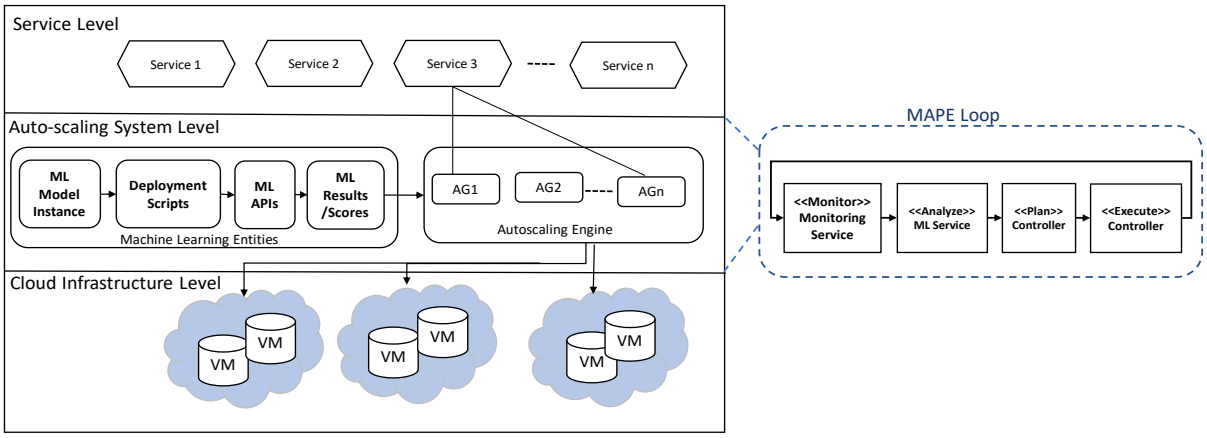


Figure 28: Predictive Auto-scaling System based on Monitor-Analyze-Plan-Execute (MAPE) loop

- **BasicImage**: This contains a pre-configured operating system and installed software packages, which is used to quickly create new instances of a computing resource.

4.2.5 The Predictive Auto-scaling System

To have the predictive auto-scaling system, we need to integrate machine learning entities with the auto-scaling system. Figure 28 presents the machine learning entities as: ML Model Instance, Deployment Scripts, ML APIs, and ML Results and Scores. During the Design Phase, ML model instance is created and transferred to a set of deployable scripts. Then, the Deployment Scripts are deployed with the help of Ansible (DevOps tool) and generate the ML APIs (Deployment Phase). The controller uses the ML APIs to run the models and receive the prediction results and scores during Run-time Phase. More details about implementation and deployment process can be found in [11].

4.2.6 The Predictive Auto-scaling System Cloud Independent Model

In our work, we build a new meta model shown in Figure 29. The CPIM represents the main predictive auto-scaling components. The CPIM model hides cloud platforms details and remains at the service level abstraction.

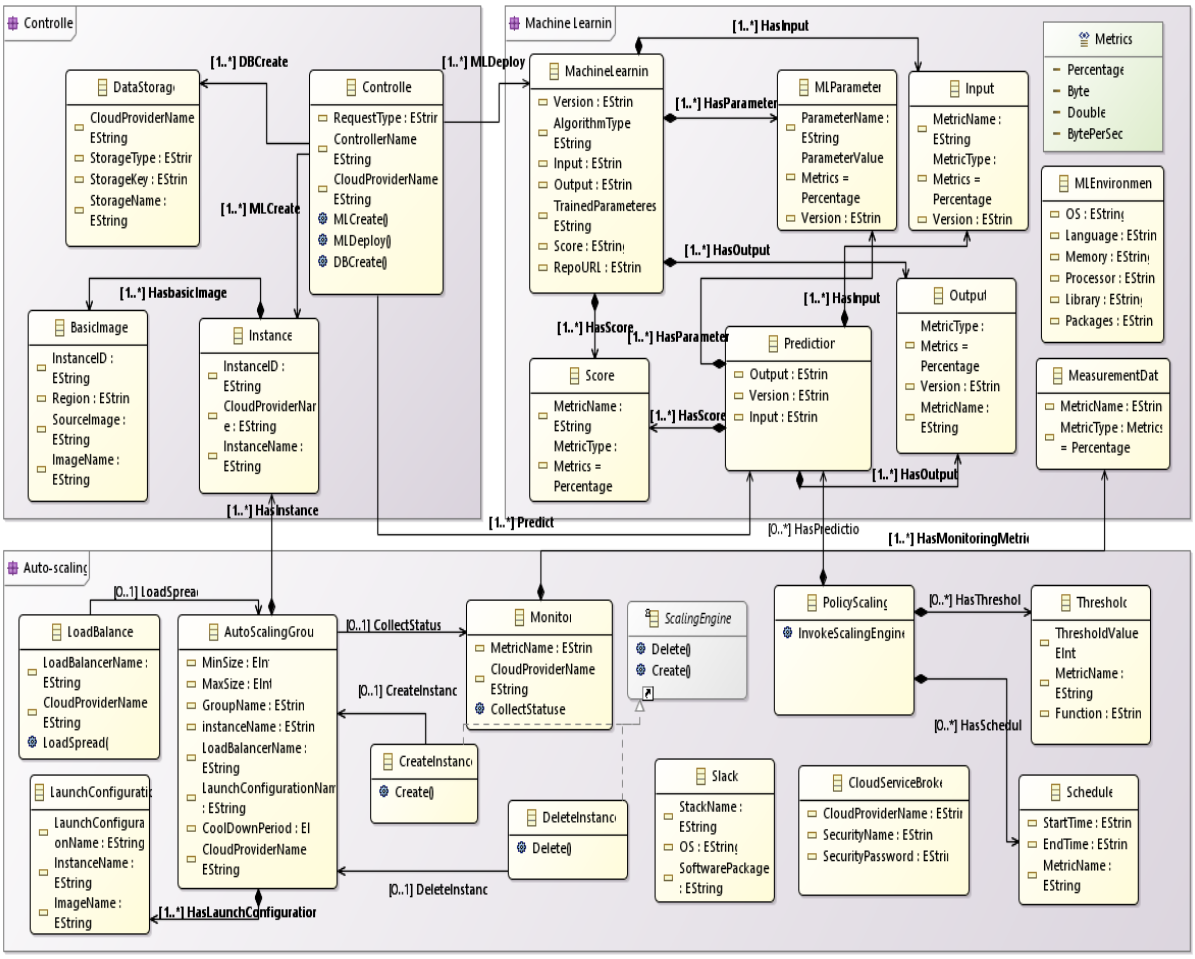


Figure 29: Cloud Platform Independent Model of Integrated Machine Learning with Auto-scaling Service

In order to support a predictive approach for auto-scaling process with the help of machine learning forecasting, we integrate workload forecasting with the auto-scaling process. We need to coordinate and orchestrate two models (CPIMs for auto-scaling and machine learning), so we add two more components:

- **Data Storage** is responsible for keeping the information and data related to models such as the name of algorithm, the model id, the value of parameters, the performance metrics results and the score. In addition, it stores the monitoring data. The data collected from the Monitoring instance becomes the training samples to learn a workload pattern for prediction and forecasting.

- **Controller** coordinates and manages the *Monitor*, the *MachineLearning*, and the *Prediction*. The *Controller* is responsible for invoking the *Monitor* to collect the workload metrics. The *Controller* calls the *MachineLearning* to train models and launches the *Predictor*. Also, The *Controller* also invokes the scaling engine after having received the prediction results.

4.3 Cloud Platform Specific Model

The idea of the proposed framework is to reduce as much as possible the deployment process in the Cloud by using an abstraction layer isolating the predictive auto-scaling system from the underlying Cloud platform and hiding details. In fact, shielding users from having to manually write scripts using low-level APIs hides the deployment complexity and reduces manual efforts and the time required to configure Cloud resources.

The CPIM is transferred to the model objects which includes class objects and associations that is specified for a target Cloud platform. A CPSM model is created as a model instance of the CPIM using the EMF generator function. The CPSM objects require details to be customized for a specific Cloud platform. The next step is to add model objects with their attribute values configured. The model object is selected by using the EMF editor (Figure 30). Upon the creation of a model object, the rule defined at the class level is checked and validated. The *Epsilon Validation Language* validates the CPSM in order to ensure the model generated is correct and complete.

A CPSM model is created as a model instance of CPIM using the EMF generator function. Such an instance model requires details to be customized for a cloud specific platform (as shown in Figure 31).

The question of how to integrate the machine learning with the auto-scaling system is interesting. The life-cycle of auto-scaling system includes five common parts: the auto-scaling group, the monitor, the scaling policy, the scaling engine and the launch configuration. The machine learning service also includes several components: monitoring, data and model storage, machine learning algorithms, validation, and model selection. Several cloud services need to work together to deliver the predictive auto-scaling system. Each cloud provider offers different types of APIs and programming languages. There is a need to automate and orchestrate the integration process.

Instead of having separate deployment processes for machine learning and auto-scaling, we have one framework for the predictive auto-scaling. The model-driven development principle is used to describe the predictive auto-scaling system at the service level. The mapping to the cloud platform-specific configuration is automatically generated and deployable.

4.3.1 CPSM for Machine Learning Model

For a machine learning instance, we need to choose programming language and available programming libraries. One prominent example is the Python platform for a machine learning environment.

The *MLEnvironment* component allows us to specify the required configuration for building an environment for running machine learning models. Figure 39 demonstrates how the *MLEnvironment* instance is created from CPIM and specifies the setting and configuration.

Figure 32 depicts an example of specifications for the machine learning instance. The instance is for SVR algorithm and CPU metric. In addition, we use MAE, $RMSE$, and R^2 as performance metrics (Score) to evaluate the accuracy of the machine learning model. In *MachineLearning* instance, there is an attribute called *RepoURL*. This attributed addresses the location of machine learning codes. We are using a central repository called GitHub, so in this stage we need to add the address of the location of the codes.

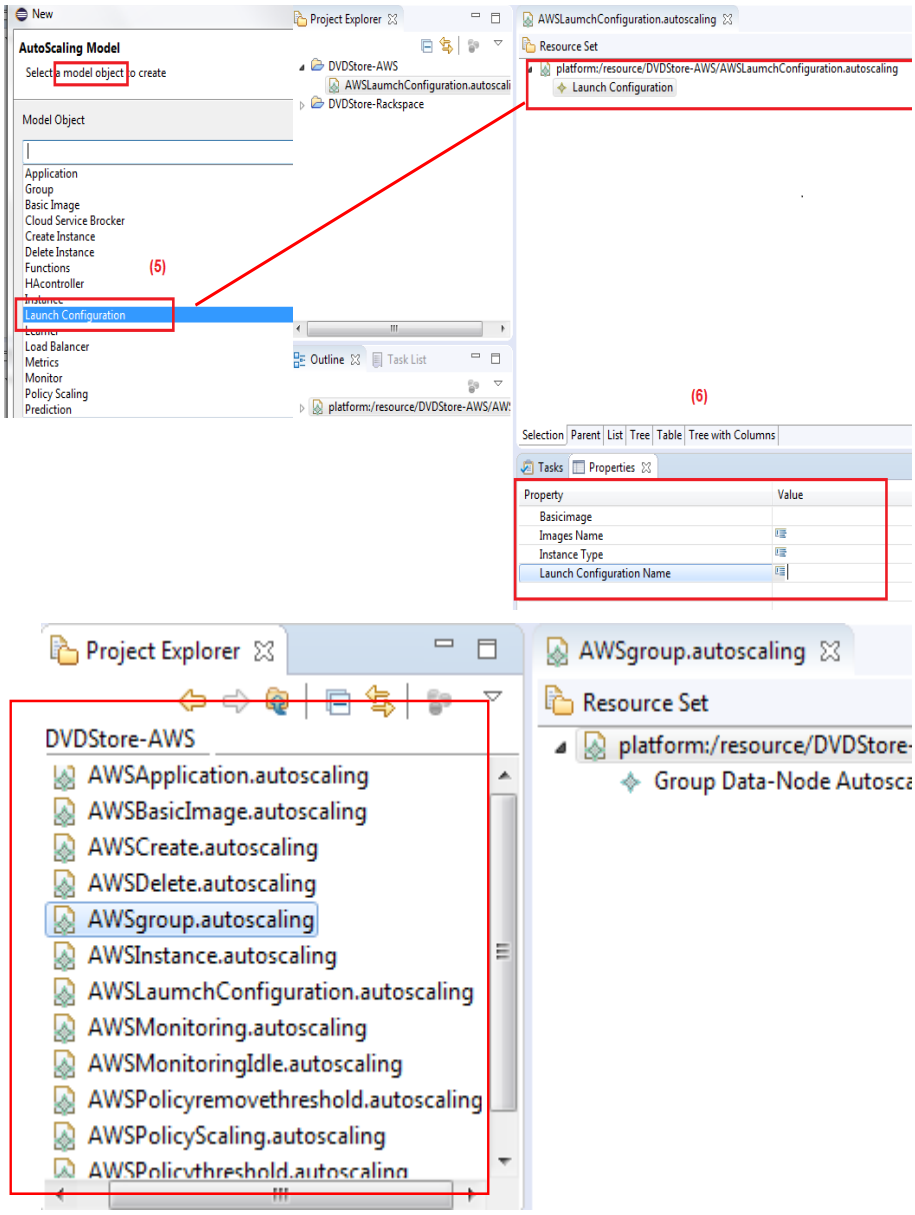


Figure 30: Adding model objects to CPSM

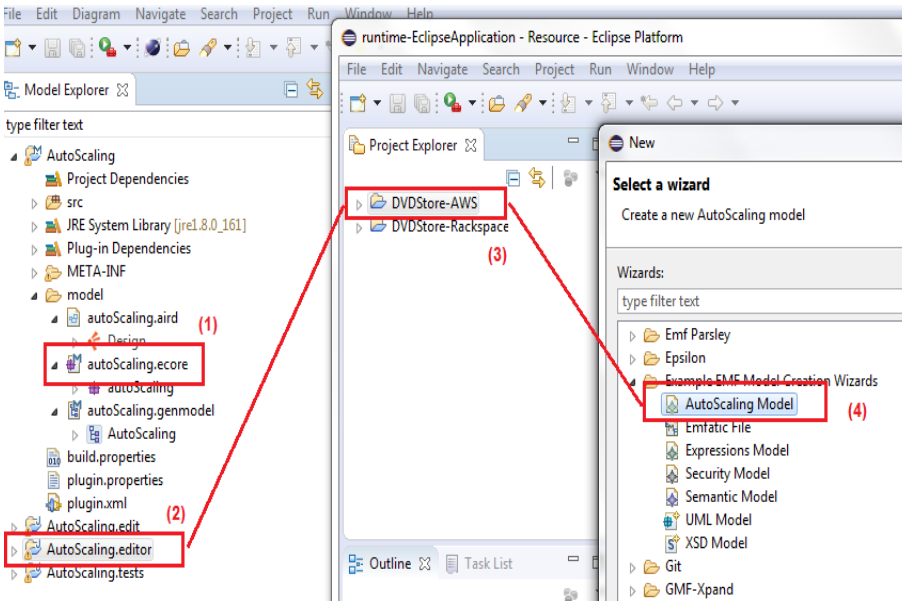


Figure 31: An example of creating CPSM from CPIM

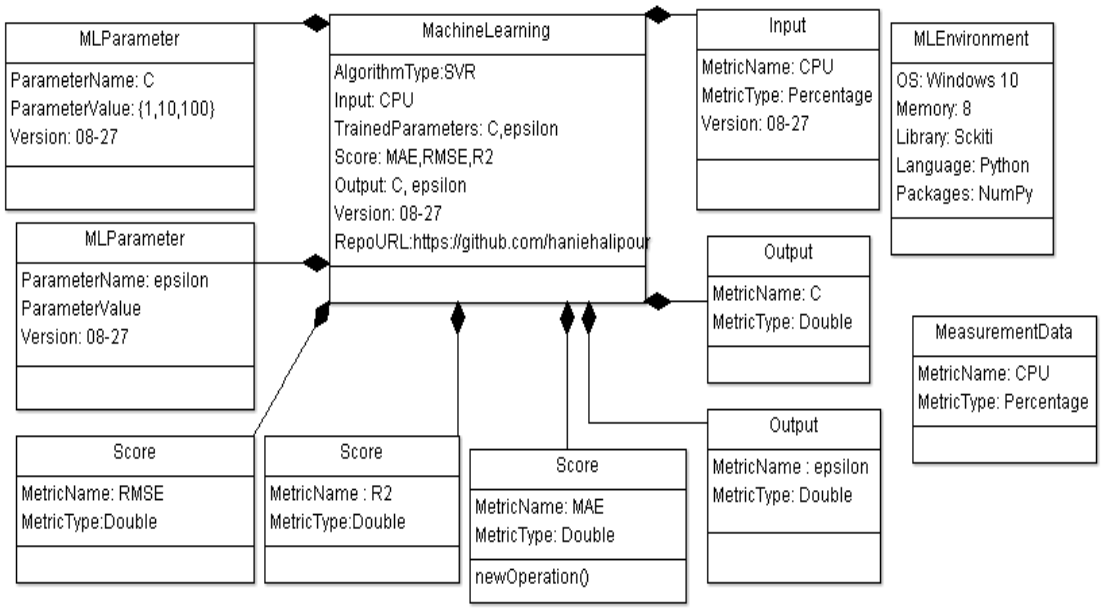


Figure 32: Example of Cloud Platform Specific Model for Machine Learning Instance (SVR Algorithm)

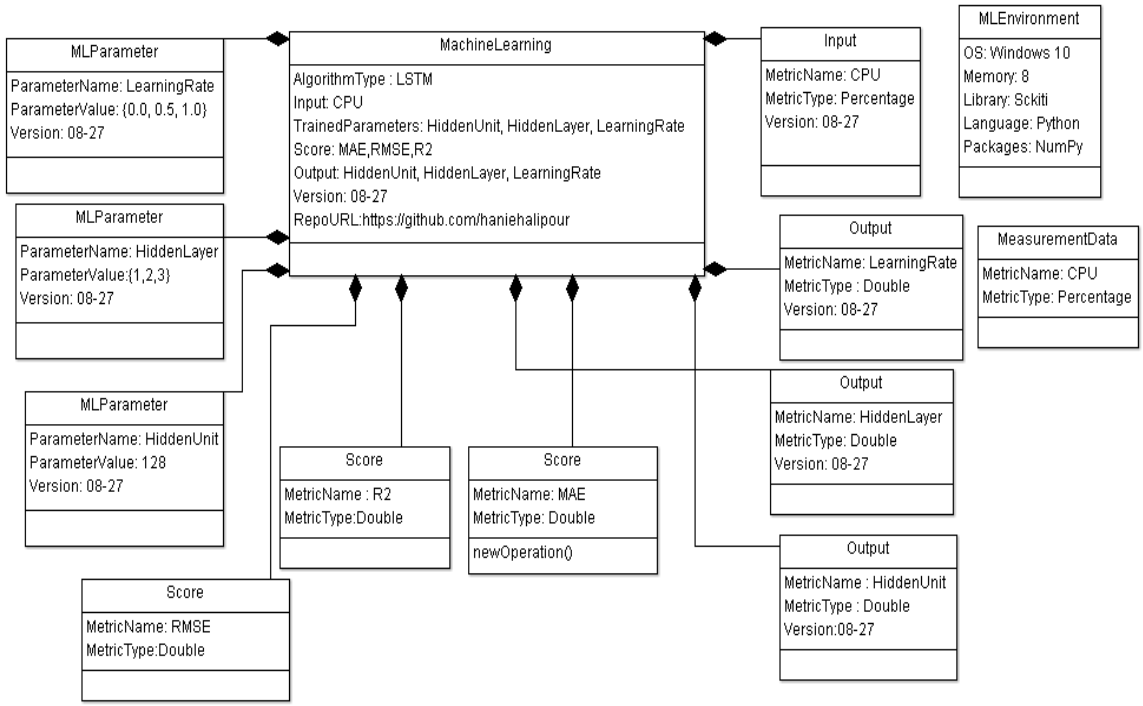


Figure 33: Example of Cloud Platform Specific Model for Machine Learning Instance (LSTM Algorithm)

Moreover, there is another example of CPSM for LSTM algorithm (Figure 33). We specified three different parameters for the LSTM algorithm:

- **LearningRate:** controls how quickly or slowly a neural network model learns a problem.
- **HiddenLayer:** represents the number of layers in deep learning neural networks.
- **HiddenUnit:** refers to the dimensionality of the 'hidden state' of the LSTM.

LSTM and BI-LSTM have more parameters such as Epochs and Batch Size. The proposed CPIM of machine learning allows us to define several parameters based on our need. In the CPIM of machine learning, we have a *MLParameter* class and it has a relation of one to many with *MachineLearning* class. So, based on requirements, we can decide which parameters fit and how many parameters are needed.

Table 20: Notations for Constraints and Rules

Variables	Description
m	MachineLearning class
I_m	instance of MachineLearning class
$\circ$	composition relation
vm	Instance
VMs	Group of instances
alg_n	Machine learning algorithm
$Algs$	Group of algorithms
$Comp_{alg_n} Instance_{vm}$	Algorithm alg_n is deployed on vm (instance)

4.3.2 Constraints and Rules

Table 20 presents a description of the notations used in constraints and rules. The constraints that need to be satisfied in order to have a machine learning service for the auto-scaling system are presented below:

- At least one machine learning (virtual machine) instance is required:

$$\sum_{vm \in VMs} number\_of_{vm} > 0$$

- Machine Learning algorithms must be deployed on the machine learning instance:

$$\forall alg_x, alg_y | alg_x, alg_y \in Algs : (\forall vm | vm \in VMs :$$
$$Comp_{alg_x} Instance_{vm} \equiv Comp_{alg_y} Instance_{vm})$$

- We have previously discussed variation of tuning parameters which produce a new instance of machine learning. Therefore, tracking the instance linkages and ensuring consistency is mandatory.

$$I_m \circ I_{input} \Rightarrow I_m(Version) \equiv I_{input}(Version)$$
$$I_m \circ I_{output} \Rightarrow I_m(Version) \equiv I_{output}(Version)$$
$$I_m \circ I_{score} \Rightarrow I_m(Version) \equiv I_{score}(Version)$$
$$I_m \circ I_{par} \Rightarrow I_m(Version) \equiv I_{par}(Version)$$

4.4 Model Validation

Epsilon Validation Language (EVL) delivers model validation capabilities by defining constraints and conditions on models. The constraints are specified within a context which represents the set of model instances that will be evaluated against the constraint. The constraint may have a guard that determines the condition to be met. If this is not satisfied, an appropriate error message will be displayed. EVL supports two types of constraints: errors and warnings. Error constraints are critical and cause the execution to terminate. EVL also provides a feature for fixing those validation errors programmatically. Therefore, EVL is used to validate the CPSM. We define the set of constraints that need to be satisfied in CPSM.

The rule is about defining the value of attributes. There are two types of attributes. For the first type, the rules need to check the input to ensure that they are not empty. In order to have accurate deployment, the input of those attributes are significantly essential, and they should define adequately. For the second type, we set the warning and suggest the default value. For example, for a machine learning object, we need to specify the name of algorithms and tuning parameters. Otherwise, this object cannot be mapped to a specific deployable service. The code snippet in Figure 34 demonstrates example of constraints that defined for CPSM.

The *MachineLearning* class is a composition class of composing classes such as *Input*, *Output*, *MLParameter*, and *Score*. We have discussed the fact that any variation of tuning parameters produces a new instance of machine learning. Therefore, tracking the instance linkages and ensuring consistency becomes a mandatory requirement. Such a requirement is defined as rules for checking instance consistency by the attribute of *Version*. There is a rule that each instance which has composition instance have to have the same value for attribute of *Version*. Figure 35 demonstrates the rule in modeling language format.

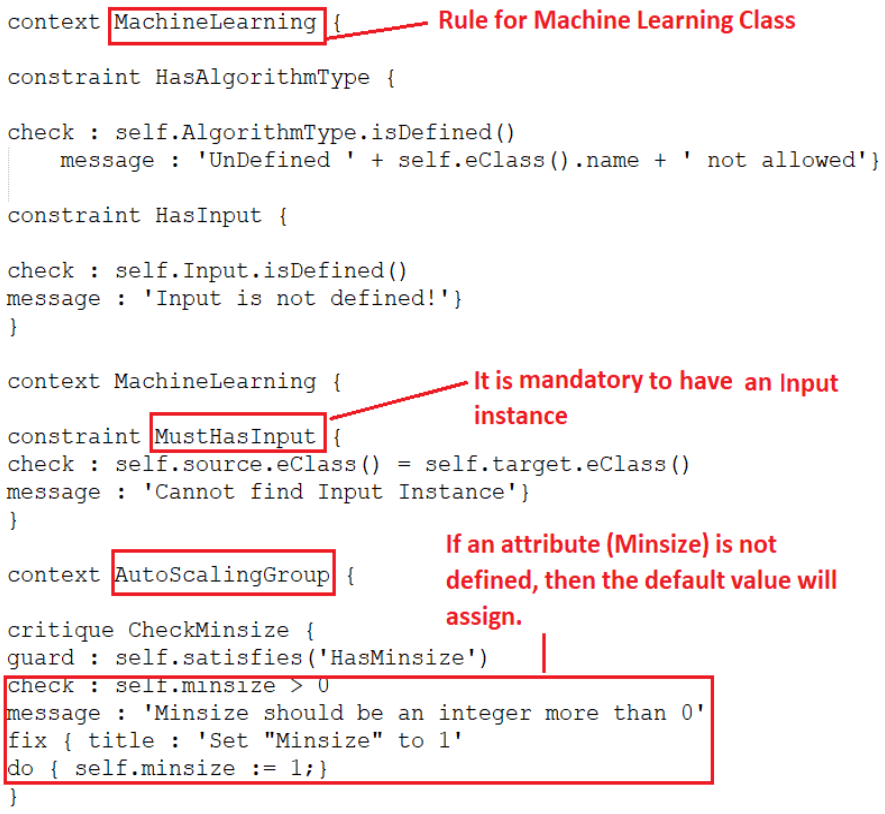


Figure 34: Example of EVL rules to validate CPSM

```
context MachineLearning {

constraint HasVersion {
check : self.version.isDefined()
message : 'Version is not defined!'}
constraint set_version_to_global {
check : version = self.self.version}
}
context Input {

constraint HasVersion {
check : self.version.isDefined()
message : 'Version is not defined!'}
constraint is_input_set {
check : input_is_set = self.version.isDefined()}
critique set_version_to_global {
check : self.self.version == version
message : 'Version is not equal'}
}
```

Versions need to be consistent

Figure 35: The machine learning meta-model rule in modeling language format

4.5 Model Transformation

A CPSM is the further input of the *Deployment phase*. By the help of *Epsilon (EGL)*, the CPSM is transferred to a set of deployable scripts. The model to deployment entails transforming the models of a predictive auto-scaling service to scripts. We automate this transformation using Epsilon [71] on top of EMF. Epsilon includes Epsilon Object Language (EOL) and Epsilon Generation Language (EGL) for parsing EMF models, UML models, and XML files to generate text with EGL templates.

- The *EGL template* contains a static text and a dynamic text retrieved from CPIM and CPSM models defined using EMF.
- The *EGL rule* defines the transformation templates and targets.
- The *EGX program* is the driver to execute EGL rules within Eclipse to launch deployment tools to run on the scripts and/or configuration files created using EGL.

An example of this is how, for building machine learning instance on AWS, we need to transfer CPSM to a JSON file for using Packer as a DevOps tool. The Packer is a deployment tool for building pre-configured images for multiple platforms from a single source of configuration. Packer uses a JSON configuration file to describe the related information of software packages in an image. Packer installs and configures all the software at the time.

In this example, the CPSM for AWS is called *AWSPSM*. First, we need *EGL template* that is generating the JSON file. It describes the basic image of a data node (Figure 36 demonstrates part of our *EGL template*). The tag pair [% %] is used to limit a dynamic section. Any text not enclosed in such a tag pair is contained in a static section. Second, we defined the *EGL rule* which is presents our rule in EGL for generating the builder JSON file for Packer (Figure 37). The *PackerImage rule* of the EGX program will transform every *AWSPSM* elements into a target file (in this case is our JSON file), using the specified template (AWSPSM.egl).

Finally, we need to execute *EGL rule* within Eclipse to run deployment tools (Packer) to run on the JSON file. An EGX program (shown in Figure 38) coordinates EGL rules and launches Packer to run on the builder.json file created from our CSPM instance on AWS *AWSPSM*.

Figure 36: The EGL template for generation code from EMF models

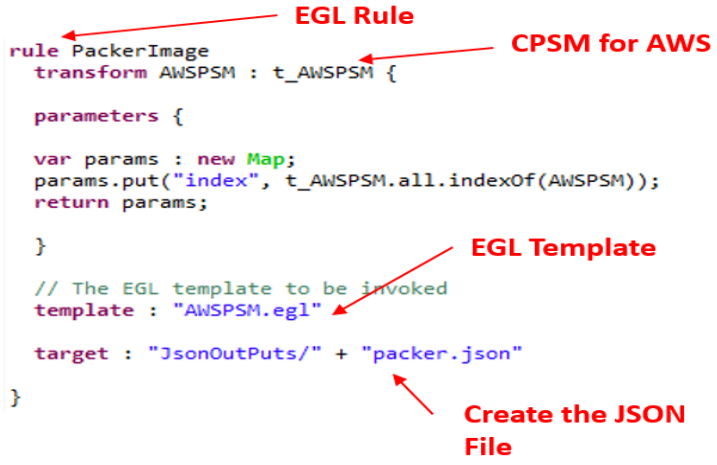


Figure 37: The EGL rule for generating JSON file

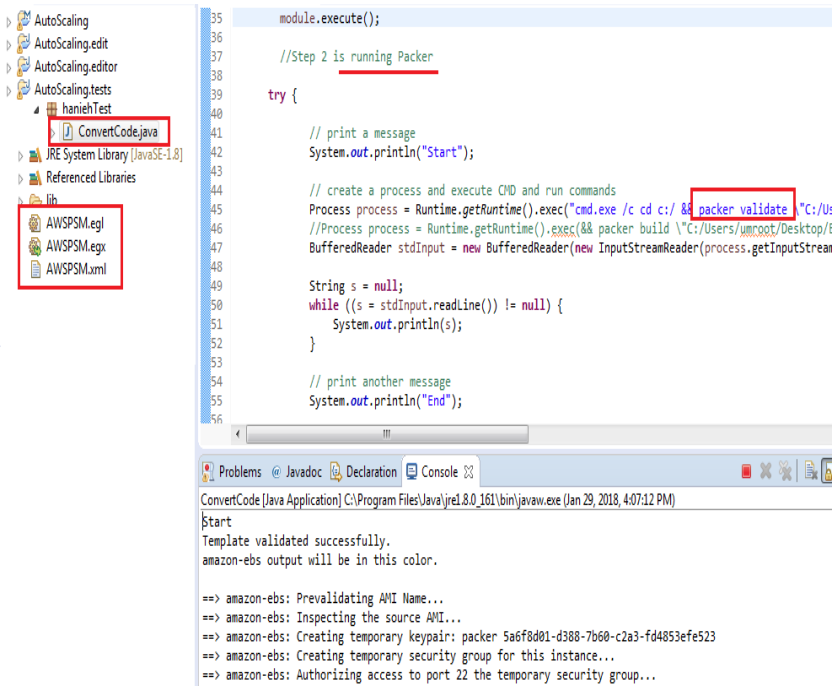


Figure 38: The EGX driver for launch Packer

4.6 Deployment Phase

The deployment process of predictive auto-scaling on a target cloud provider may require the construction of different vendor-specific configuration artifacts. Also, integrating machine learning services with an auto-scaling system requires a substantial amount of manual tasks. We are proposing a model-driven approach to abstract and automate a predictive auto-scaling system through model-driven techniques and DevOps. In order to deploy the deployment scripts on the target cloud platform, we employ DevOps tools: Packer and Ansible. Cloud-based DevOps processes facilitate the continuous delivery of infrastructure and services. Infrastructure as Code is the cornerstone of DevOps for automating the infrastructure provisioning based on practices from software development.

We used Packer [15] as a DevOps tool in order to create pre-configured images. Packer is an open source tool for creating identical machine images for multiple platforms from a single source of configuration. Packer installs and configures all the software for a machine at the time the image is built. Moreover, Packer uses a JSON template to create an image.

A JSON template has the following three main parts:

Table 21: Ansible Terms

Terms	Definition
Host and Groups	A host is a remote machine that Ansible manages. Hosts can be organized in groups. All hosts have a name (IP or domain name) where they can be reached.
Inventory	A file that describes Hosts and Groups in Ansible Modules Modules are the units of work that Ansible sends to remote machines. Ansible refers to the collection of available modules as a library.
Playbooks	Playbooks are the language by which Ansible configures, administers, or deploys systems.
Plays	A play is a mapping between selected hosts and the tasks that run on those hosts to define the role that those systems will operate. A playbook is a list of plays.

- Variables: where custom variables are defined.
- Builders: where all the required image parameters are mentioned.
- Provisioners: where Ansible playbook for configuring and settings in the image is integrated.

Ansible [1] allows the management of infrastructure and easily adapts to new cloud environments in a migration scenario. It is a configuration management and provisioning tool, similar to Chef, Puppet or Salt. In addition, Ansible supports public and private Cloud technologies and vendors like AWS, Google Compute Engine, Microsoft Azure, OpenStack, Rackspace Cloud service and VMware. In our work, we chose Ansible to demonstrate the process which is simplified in terms of communication via standard SSH commands. Furthermore, Table 21 contains some important terms.

4.6.1 How to create machine learning environment?

To perform training and prediction on workload history, we need to create a machine learning environment. Figure 39 demonstrates how CPSM of the *MLEnvironment* instance transfer to deployment script are used by Packer.

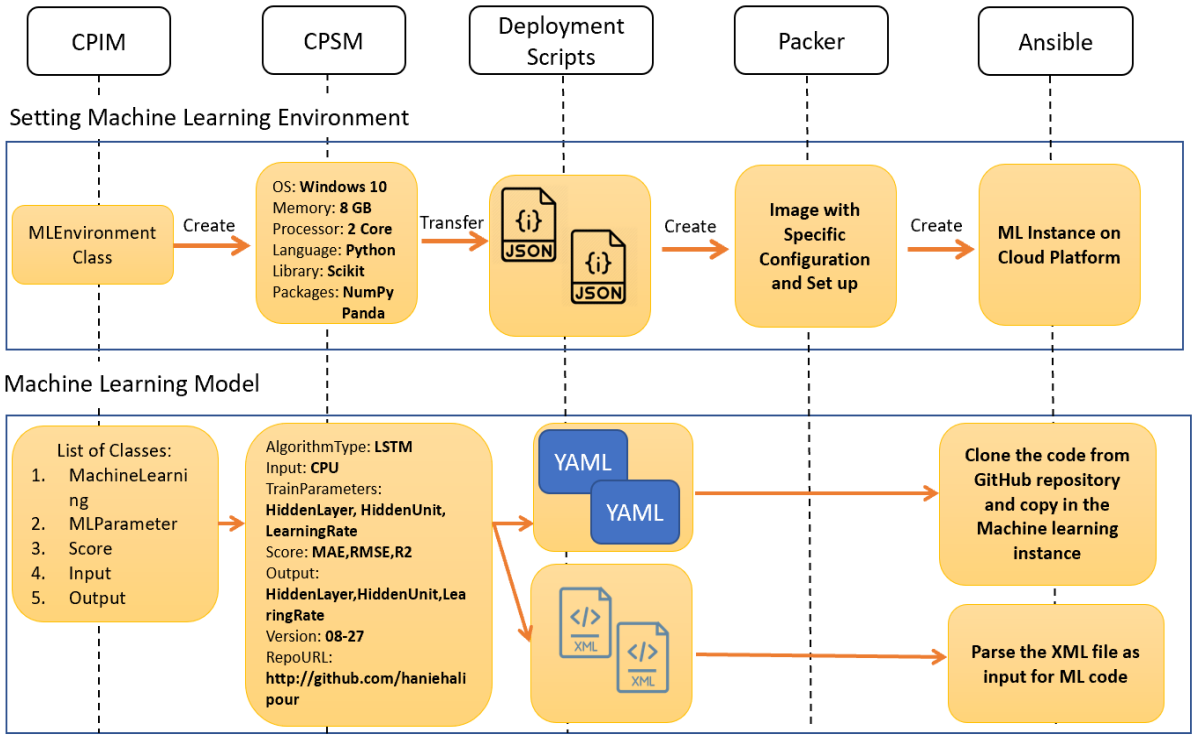


Figure 39: Sequence of Actions for Generating CPSMs and Deployment Scripts for Machine Learning Environment and Models

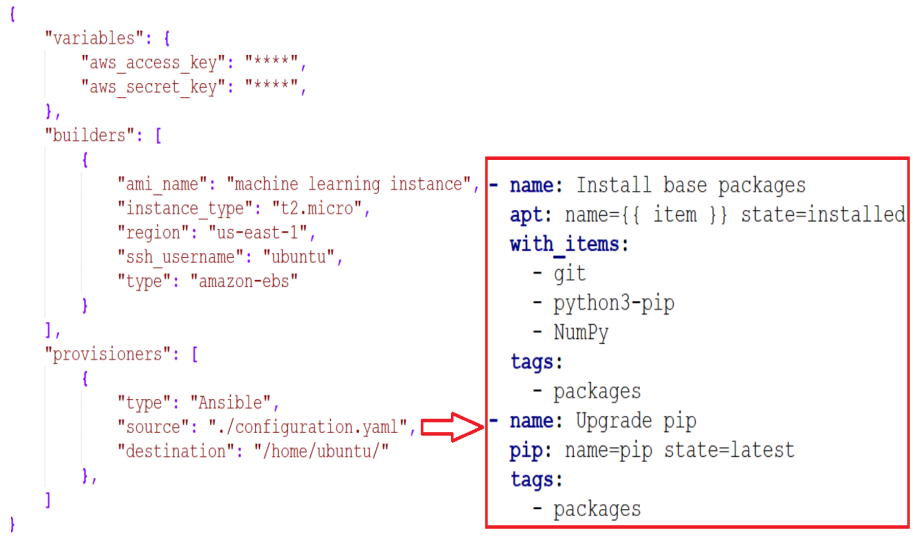


Figure 40: Example of JSON template for creating an machine learning image

One of the classes in our CPIM is the *MLEnvironment* class which represents the setting and configuration for machine learning instance and help us launch completely configured cloud instance from the pre-build image (Figure 39). A machine (instance) image is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines. Machine image formats change for each platform.

Then, in CPSM instance (shows in Figure 39), we specify the environment configuration such as language, package and library. As previously mentioned, the CPSM is in XML format and it must be automatically transferred it to JSON file for Packer. To create an image, we employ Packer. Figure 40 demonstrates A JSON template used by Packer. The created image by Packer is used to instantiate machine learning instance.

4.6.2 How to deploy machine learning codes on the environment?

When the machine learning instance is ready, we need to deploy machine learning algorithms. The related classes in our CPIM are *MachineLearning*, *Input*, *Output*, *Score*, *MLParameter*. Then, in CPSM instances (shows in Figure 39), we specify the input, output, performance metrics and hyperparameters for each algorithm. As Figure 39 illustrates, In *MachineLearning* instance, there is an attribute called *RepoURL*.

```
<autoscalingv2:MachineLearning AlgorithmType="LSTM"
 RepoURL="http://github.com/haniehalipur">
<HasMLParameter ParameterName="HiddenLayer"
ParameterValue="1,2,3" Version="08-27"/>
<HasMLParameter ParameterName="HiddenUnit"
 ParameterValue="128" Version="08-27"/>
<HasMLParameter ParameterName="LearningRate"
 ParameterValue="0,0.5,1" Version="08-27"/>
<HasScore MetricName="R2" MetricType="Double"/>
<HasScore MetricName="RMSE" MetricType="Double"/>
<HasScore MetricName="MAE" MetricType="Double"/>
<HasInput MetricName="CPU" MetricType="Percentage"
 Version="08-27"/>
<HasOutput MetricName="HiddenLayer"
MetricValue="Double" Version="08-27"/>
<HasOutput MetricName="HiddenUnit"
MetricValue="Double" Version="08-27"/>
<HasOutput MetricName="LearningRate"
 MetricValue="Double" Version="08-27"/>
</autoscalingv2:MachineLearning >
```

Figure 41: Example of XML template for a machine learning algorithm

This attributed address the location of machine learning code for algorithms.

This attribute addresses the location of machine learning code for algorithms. We are using Ansible to pull the machine learning codes from the repository. Ansible allows for the management of infrastructure and is easily adaptable to new cloud environments. In this research, code from a central repository called GitHub is being deployed. In addition, CPSM instances for the machine learning model are in an XML format (Figure 41). It contains the essential information such as input, output, score, and version. These XML file is parsed and converted YAML file (Figure 42) for Ansible.

4.6.3 Architecture Deployment on Microsoft Azure

Microsoft Azure [3] is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services on Microsoft cloud platform. Figure 43 illustrates the deployment approach for our proposed architecture on the Microsoft Azure. We use the PowerShell script to deploy the CPSM model. The Ansible script runs the PowerShell script and then the Azure Function App is deployed with the necessary codes. The following entities are deployed by the Ansible script:

```yaml
hosts: Machine learning instance
vars:
    gh_repo: git@github.com:haniehalipour.git
    dest_dir: /home/devOps
    dest_dir_owner: haniehalipour
    dest_dir_perm: 0755
name: installing git
apt:
      name: git
      state: present
become: yes
name: creating source directory
file:
   path: "/home"
   state: directory
become: yes
file:
   path: "home/devOps"
   mode: 0777
   state: directory
become: yes
name: cloning repo
git:
    repo: "haniehalipour.git"
    dest: "home/devOps"
```

Figure 42: Example of YAML template for cloning code from GitHub [11]

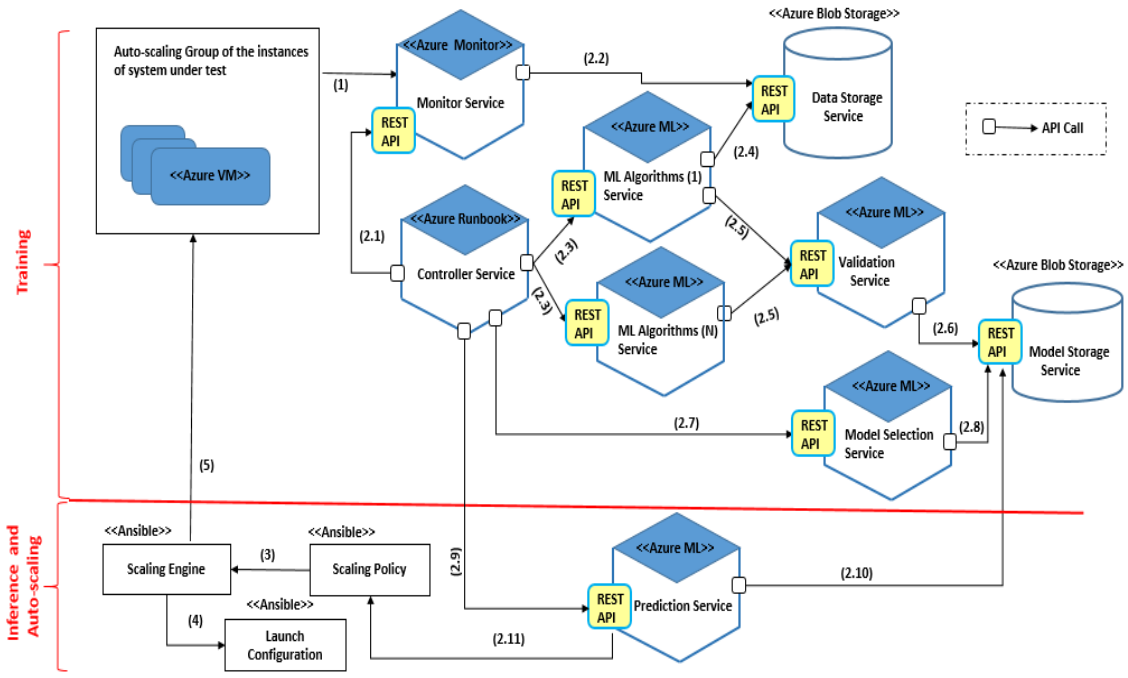


Figure 43: Architecture Deployment on Microsoft Azure

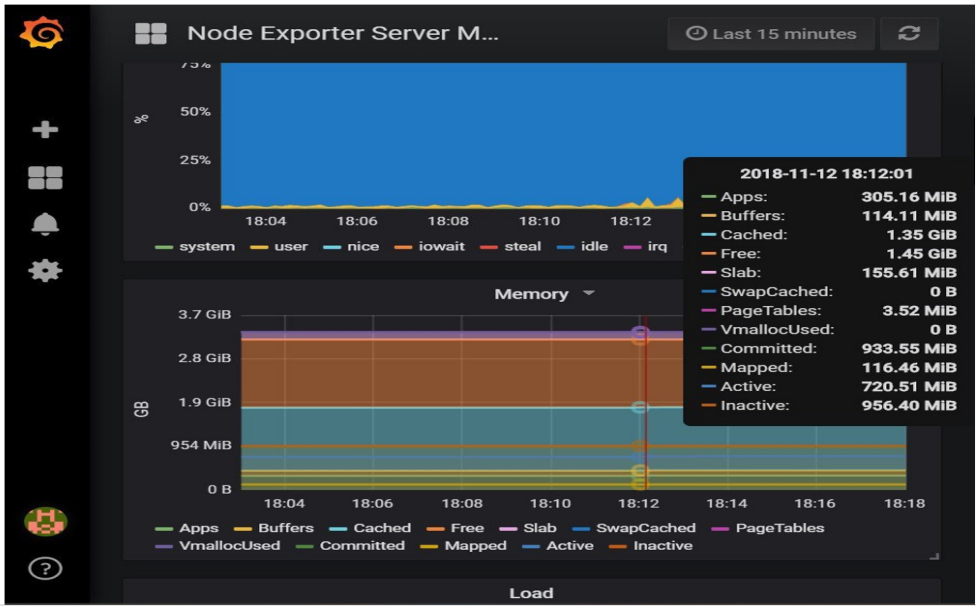


Figure 44: Grafana Plug-in a on Azure Instance

- Monitoring service: The Azure Monitoring is responsible to collect the generated workload on instances of the system under test (1). It monitors base-level metrics including CPU percentage or Memory usage and stores them in the Azure Storage (2.2). We use the Prometheus and the Grafana plugin for Azure Monitor service. Prometheus [16] is a monitoring solution for storing time series data like metrics. Grafana [10] allows to visualize the data stored in Prometheus (and other sources). Figure 44 demonstrates Grafana plugin a on Azure instances. The following queries are the example of the queries for each metric:

  - Memory_query: avg ((node_memory_MemTotal_bytes-node_memory_MemFree_bytes-node_memory_Cached_bytes)/(node_memory_MemTotal bytes) * 100)

  - CPU_query: avg(100-(avg by(instance) (irate (node_cpu_seconds_totalmode=”idle” [5m]))*100))

  - Network_in: avg(node_network_receive_bytes_total)

  - Network_out: avg(node_network_transmit_bytes_total)

- Data Storage service: The Microsoft Azure Storage is a powerful cloud service and includes three different data services: Blob storage, File storage, and Queue storage. In our implementation, we utilize the Azure Blob storage. The data

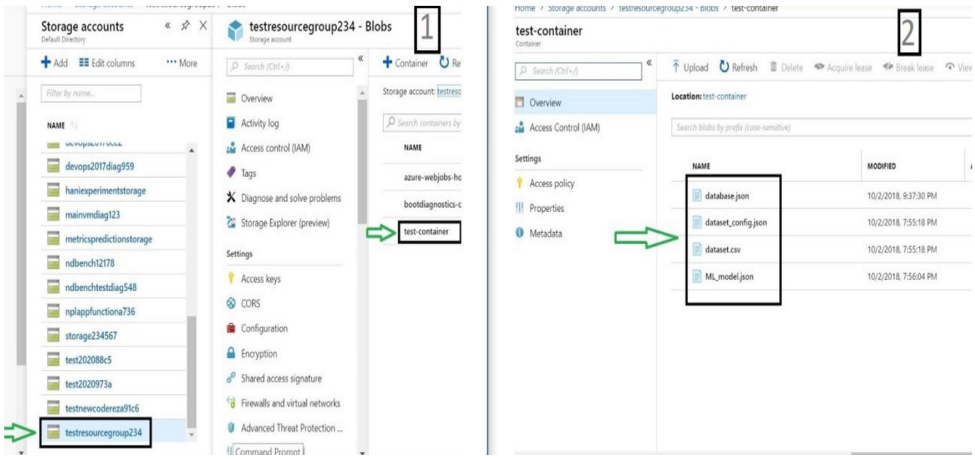


Figure 45: Azure Blob Storage files

collected from the Azure Monitoring becomes the training samples to learn a workload pattern for prediction and forecasting. The Azure Blob Storage saves the model training results and the collected metrics as inputs for training. Figure 45 demonstrates the files of blob storage in Azure. The ML-model.json contains the model information. The following information is an example of the model information and the data is stored in JSON format.

"selected_model": "LR", "model_id": "ski_model_LR.pkl", "params":"penalty": "l2", "dual": false, "tol": 0.0001, "C": 1.0, "fit_intercept": true, "intercept_scaling": 1, "solver": "liblinear", "max_iter": 100, "multi_class": "ovr", "verbose": 0, "warm_start": false, "n_jobs": 1 The database.json contains the information related to add or remove resources, also the dataset.csv is stored the time series workload. In addition, the dataset-config.json file includes the configuration data that is related to dataset. The following example shows the information related to our dataset. "type": "csv", "name": "dataset.csv", url":"dataset.csv" , "has_header_row": "yes", "train_columns": ["col_name": "cpu", "col_number": "2", "col_name": "memory", "col_number": "3", "col_name": "network_in", "col_number": "4", "col_name": "network_out", "col_number": "5"], "multi-class_target_col": "col_name": "final_class", "col_number": "7", "regression_tag": "col_name": "final_target", "col_number": "6", "expt_model_name": "ski_model"

- Controller service: The Azure Function App is an automation tool to manage different resources. The Azure Function App periodically invokes the Azure

Table 22: Hardware Specification of Machine Learning instances on AWS and Azure

Cloud Provider	Type	VCPU	Memory
Azure	B2s	2	4 GB
AWS	M3.medium	2	4 GB

Monitoring to collect the workload metrics (2.1). Also, the Runbook calls the machine learning algorithms to train models (2.3). Furthermore, this service launches the prediction service using a trained model (2.9). Azure Functions can be triggered on a schedule. When a function timer is triggered, the function performs the responsibility.

- Azure Data Science Virtual Machine: The monitored data from the Azure Blob Storage is retrieved and then converted to the proper dataset for our machine learning algorithms (2.4). The Data Science Virtual Machine is a customized VM image on the Microsoft Azure cloud built specifically for doing data science. There are several pre-installed machine learning toolkits on that VM, so we can have different machine learning algorithms for our implementation (2.3). As customized VM image on the Microsoft Azure cloud explicitly built for data science. Furthermore, we implement a REST API to access the inference model as a service for the real-time prediction phase. This service is called by the Azure Function App used to perform training and prediction. In order to implement The inference process, we use the scikit-learn [17] which is a free machine learning library for the Python. The capacities of Azure instances are presented on Table 22. At runtime, the models are constantly updated whenever a new dataset for the next two weeks arrives. New models are created, new calculated scores are incorporated to the model storage, and older scores are removed. The forecasting process is then repeated, which may lead to changes in the models, performance metrics, and score results.

- Resource Group: It groups Azure instances (resources) for scaling and managing based on the minimum and maximum number of running instances allowed at any time.

The Ansible playbooks map the Azure Autoscale service. The Ansible resource group playbook contains the capacity setting which indicates the minimum, maximum, and default values for number of instances. Also, there is an Ansible playbook

```
- name: Create a resource group
azure_rm_resourcegroup:
name: groupName
location: region
tags:
min_size : minsize
max_Size : maxsize
desired_Number : defaultvalues
```

Figure 46: The example of Ansible Auto-scaling group for Azure

```
LambdaFunction:
  Type: AWS::Lambda::Function
  Properties:
    Code:
      S3Bucket: {Ref: S3Bucket}
      S3Key: {Ref: S3Key}
      S3ObjectVersion: {Ref: S3ObjectVersion}
    Handler: main.lambda_handler
    Role: {'Fn::GetAtt': [IAMRole, Arn]}
    Runtime: python3.6
 Outputs:
 apiGatewayInvokeURL:
  Value: "https://execute-api.amazonaws.com/dev"
```

Figure 47: CloudFormation Template for Creating AWS Lambda

for setting the rules, policy, and scale actions. Figure 46 illustrates an example of Ansible auto-scaling group playbook.

4.6.4 Architecture Deployment on Microsoft AWS

For AWS, we use the AWS *CloudFormation* service to deploy machine learning elements. The *CloudFormation* service includes a template that contains all the information extracted from the CPSM model. When the template is submitted by the Ansible script, the *CloudFormation* service launches the necessary resources such as a *Machine Learning EC2 Instance*, a *Lambda Function*, and a *DynamoDB Storage*. The example of the CloudFormation template for creating AWS Lambda (Figure 47).

The following Figure 48 shows our solution that we have adopted for experimenting with Amazon cloud environment. The following entities are deployed by the Ansible script and the *CloudFormation* service on the AWS:

- Monitoring service: The CloudWatch tracks metrics, generates log files, sets

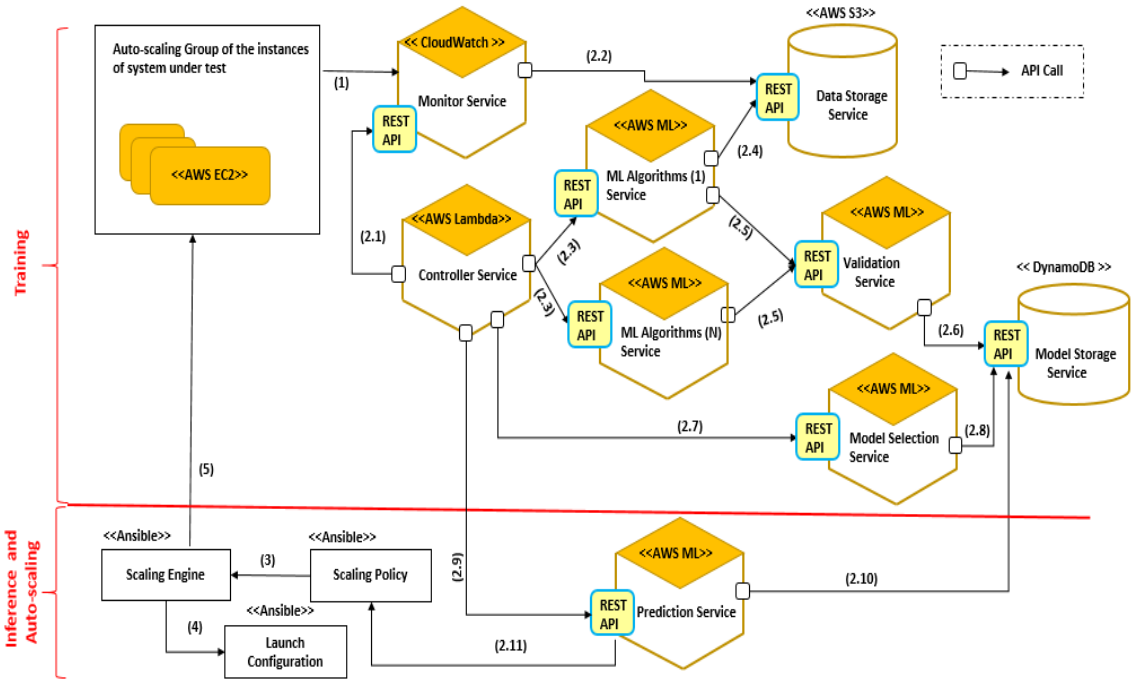


Figure 48: Architecture Deployment on Microsoft AWS

alarms, and automatically sends alarm to AWS resources (1). It is responsible for retrieving logs and storing them into AWS S3 (2.2). Amazon Cloudwatch monitoring service provides hypervisor-specific metrics. To have OS-specific metrics such as memory utilization, we need to add an extra monitoring script. So, we can collect the percentage of system memory as a custom metric. In addition, we need to standardize the range of variables and generally performed during the data processing step. Since the range of values in the raw data varies widely, it is required we normalize the metrics so that they are all in the same scale.Network metrics are in Bytes which need to be transformed into a percentage. We need to measure the network bandwidth between Amazon EC2 instances in order to convert Bytes to a percentage. AWS suggests iPerf3 [12] which is a tool for achieving measurements of maximum achievable IP networks. We used this tool to calculate the maximum network bandwidth and then convert the collected metrics to percentage form.

- Data storage service: The S3 acts as our data warehouse where we can efficiently retrieve datasets when we are testing or training our models. We use the AWS S3 as the data storage.

- Controller service:We use Amazon Lambda function to orchestrate our different parts of the architecture. Lambda employs a serverless architecture which means the code runs without managing any servers or a backend service. The Lambda calls the CloudWatch to collect the workload metrics (2.1). Also, the Lambda function calls machine learning API to train our models (2.3) and the prediction service for prediction functions (2.9). We create a Lambda function and direct AWS Lambda to execute it on a regular schedule. We can specify a fixed rate (for example, execute a Lambda function every hour or 10 minutes).

- Machine learning algorithms service: We implement different machine learning models on EC2 instance (2.3). The workload history is pulled from S3 (2.4), training the models and test them.

- Validation service: By machine learning EC2 instance, we calculate performance metrics to evaluate the result of each algorithm (2.5). Once the validation is completed, the parameter values are stored in AWS DynamoDB as the model storage (2.6).

- Model selection service: The result of validation service is retrieved from DynamoDB (2.8) and used for selecting a suitable model for the prediction (2.7).

- Model storage service: DynamoDB stores our machine learning models, the result of performance metrics, and scores.

- Prediction service: The machine learning service and the prediction service are implemented in the machine learning EC2 instance. The selected model, which is retrieved by the prediction service from the DynamoDB (2.10) is used for the prediction part (2.9). Then, the result is returned to the auto-scaling service for decision making.

There is an Ansible playbook which is responsible for grouping the EC2 instances in order to scale and manage the instances, based on the minimum and maximum number of running EC2 instances(Figure 49). Also, we have an Ansible playbook which provides all the necessary information that is required to instantiate EC2 instances. In addition, a set of policies for scaling in and out are defined on the separate Ansible playbooks.

```
- ec2_asg:
name : groupName
launch_configuration_name : configName
health_check_period : time
min_size : minsize
max_Size : maxsize
desired_capacity: desiredCapacity
region : region
```

Figure 49: The Ansible Auto-scaling group for AWS

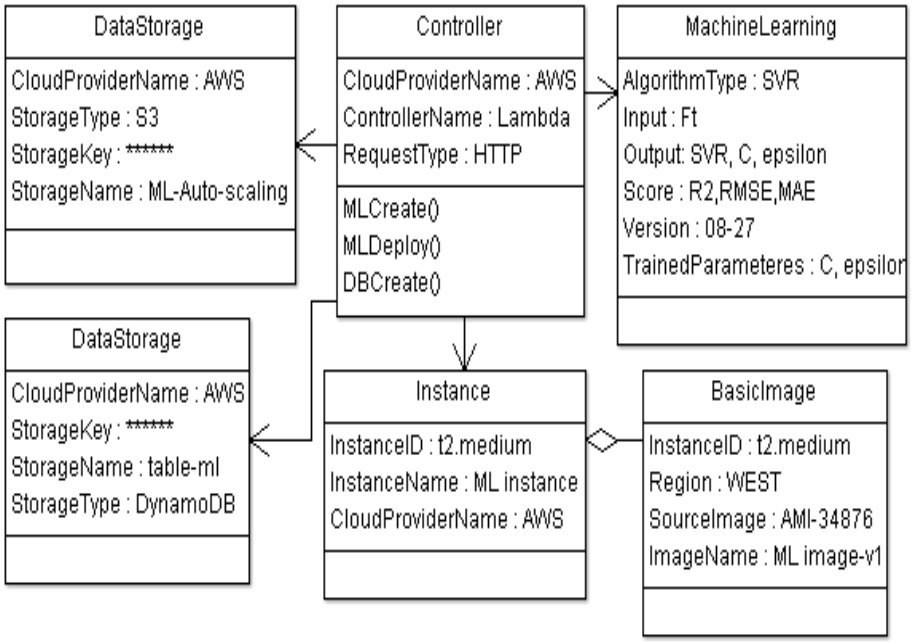


Figure 50: Example of Cloud Platform Specific Model for Controller

4.7 Run-time Phase

The run-time phase triggers the execution of deployed machine learning components upon the collection of the system level metrics. The core component to bridge the machine learning components and components of the cloud auto-scaling system is the *Controller* component. Figure 50 shows a cloud platform specific model generated for the *Controller* instance on the AWS cloud. The *Controller* is a schedule-based service, and it is trigger by a time interval to call APIs of machine learning services. For example, when the training time is triggered, the *Controller* calls the *Machine-Learning* to start model training stage for constructing, testing and validating the models. Each machine learning instance contains one machine learning algorithm. The API is defined on the instance level. The same machine learning algorithm with different tuning parameter values produces multiple instances. Each instance has its unique REST API.

4.8 Summary

In this chapter, we answered the second and third research questions:

RQ 2. Does representing machine learning models help an auto-scaling system?

RQ 3. How to integrate the machine learning with the auto-scaling system?

Motivation: The machine learning techniques need to be combined with the auto-scaling process in order to scale based on the prediction results. Integrating the machine learning components with the auto-scaling system requires that a substantial number of manual tasks be completed. Also, the life-cycle of auto-scaling system includes five common parts: the auto-scaling group, the monitor, the scaling policy, the scaling engine, and the launch configuration. Machine learning service also includes several components: monitoring, data and model storage, machine learning algorithms, validation, and model selection. Several cloud services need to work together to deliver the predictive auto-scaling system. There is a need to automate and orchestrate the integration process. The deployment process of predictive auto-scaling on a target cloud provider may require the construction of different vendor-specific configuration artifacts.

Approach: We propose a high-level abstraction for modeling the entities of the machine learning models such as machine learning algorithm types, inputs, outputs, parameters, and evaluation scores. Then, we integrate it with proposed auto-scaling meta model to have a predictive auto-scaling model. Moreover, instead of having a separate deployment process for machine learning and auto-scaling, we have one framework for the predictive auto-scaling. The model-driven development principle is used to describe the predictive auto-scaling system at different levels and the mapping to cloud platform-specific configuration is automatically generated and deployable. We are proposing a model-driven approach to abstract and automate a predictive auto-scaling system through model-driven techniques and DevOps.

Chapter 5

Evaluation

In this chapter, the results of the experiments conducted to evaluate our proposed approach are presented. The experiments are deployed on three cloud environments: Rackspace, Amazon EC2, and Microsoft Azure. The goal of these experiments is to evaluate the advantage of using a model-driven approach for machine learning service in cloud computing and to illustrate how we combine this approach with the DevOps approach. In addition, the subsequent goal is to measure the run-time cost of machine learning-based forecasting for the auto-scaling process and demonstrate how the run-time cost is reduced with the help of this proposed architecture.

5.1 Evaluation Metric (CMP)

Cloud Migration Point (CMP) approach has been developed in [64] as an important software size measure for legacy-to-cloud migration projects. They showed that CMP is more suitable for cloud migration projects than other existing size metrics in previous literature since it captures special aspects of the cloud migration context. The CMP model takes into consideration cloud-specific dependencies for each migration task. This model satisfies all necessary conditions of a software size measurement and it has been empirically validated as a predictor of effort estimation for cloud migration projects. We use CMP as an evaluation metric to estimate the deployment effort of our proposed model-driven framework.

The cloud deployment process includes five main components, namely installation, configuration, database migration, code modification, and network connection. Since

the data node of the NDBench and DVD store consist of the full stack of packages, the predictive auto-scaling service deployment involves building an image of software packages and launching the image on provisioned instances. There are no application code changes required. In addition, for the machine learning service, we employ the Scikit-learn library. The codes are applicable to both cloud environments. There are no code changes required. Our evaluation involves installation, configuration, and model template modification.

The evaluation is structured into three phases, corresponding to the Cloud Migration Point (CMP) [64] approach. During the first phase, the deployment tasks are analyzed to identify and classify the tasks into three categories, namely the *Storage and Database*, the *Template Changes*, and the *Installation and Configuration*. In the second phase, each task is assigned a complexity level, which is determined by the functionality of the cloud services and the interaction of the cloud services with each other. In the last phase, the CMP value is computed as a weighted sum. Based on [64], each function (task) is weighted based on its type and the level of its complexity, in agreement with standard values as specified in the Counting Practices Manual [7]. Interviews and surveys used to collect data from different projects for the weight of tasks.

The evaluation starts with identifying the type of deployment tasks.

- *Storage and Database*- Tasks involved account for authentication and establishing connections.
- *Template Changes*- Deployment tasks consist of creating template and model objects. Also, tasks include specification of attributes.
- *Installation and Configuration*- Setup software installation scripts and configure the values of environment variables.

The behavior of each task is taken into account to evaluate its complexity level in terms of the number of methods changed, the number of services edited and the number of attributes modified. The value of CMP is defined as a weighted sum of its three categories CMP_i with $i \in \{$*Installation and Configuration*, *Template Changes*,

Table 23: Complexity Evaluation for Each Installation and Configuration Task

Configuration	Installation		
	Run No-Installation	Package Library	Installation
< 2	Low	Low	Average
$2 - 5$	Low	Average	High
$>= 6$	Average	High	High

Storage and Database}.

$$CMP = \sum_{i=0}^{2} CMP_i * w_i \tag{11}$$

Where CMP_i is the value of CMP of type i, and w_i is the weighted value for CMP type i.

CMP_{Ins}

The *Installation and Configuration* category (CMP_{Ins}) reviews tasks such as installation of software, servers, third-party library, and configuration environment variable. All tasks are classified into two types:

- Infrastructure level: Installation of infrastructure level software and servers. For example, setting up an Azure or AWS instance or image, installing OS, and installing the database such as S3 and Blob.
- Application level: Configure application level environment and libraries.

We estimate the complexity level (Low, Average, or High) of each task based on the number of configuration steps and installation type as shown in Table 23. Each task is allocated with a weighted value as shown in Table 24 based on its type and complexity level.

Table 24: Weight Evaluation for Each Installation and Configuration Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{Ins}	Application	1	2	7
	Infrastructure	1	3	9

Table 25: Complexity Evaluation for Each Storage and Database Task

Storage and Database	Complexity Level
Database Changes	High
API Changes	Average

CMP_{db}

First, all cloud storage related tasks are grouped into two types, database changes, and API changes. The complexity of each task is determined based on differences between cloud storages and steps for configuring APIs as shown in Table 25. Then, each task is allocated with a weighted value as shown in Table 26 based on its type and complexity level.

Table 26: Weight Evaluation for Each Storage and Database Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{db}	Database Changes	1	4	7
	API Changes	1	3	6

CMP_{code}

The *Template Changes* category CMP_{code} assesses all tasks related to create or modify a new model object and a template. Three different types are defined to capture aspects of code and template changes:

- Create and Instantiation: Tasks that accommodate the creation of a new model object or template.
- Add or Remove Attributes: Each cloud service requires a set of input (attribute value) and this type cover tasks related to changes in attributes.
- Change or Edit Service: Different cloud services communicate with each other in order to deliver service or functionality.

Based on three types, there are three dimensions to evaluate complexity level as shown in Table 27. Then, each task is allocated with a weighted value as shown in Table 28 based on its type and complexity level.

Table 27: Complexity Evaluation for Each Template Changes Task

Create and Instantiation	Add or Remove Attributes		
	$0-3$	$4-7$	$7-10$
$0-4$	Low	Low	Average
$5-8$	Low	Average	High
$>=9$	Average	High	High

(a) For Change or Edit Service $(0-2)$

Create and Instantiation	Add or Remove Attributes		
	$0-4$	$5-8$	$>=9$
$0-3$	Low	Low	Average
$4-7$	Low	Average	High
$>=8$	Average	High	High

(b) For Change or Edit Service $(2-4)$

Create and Instantiation	Add or Remove Attributes		
	$0-3$	$4-7$	$>=8$
$0-2$	Low	Low	Average
$3-6$	Low	Average	High
$>=7$	Average	High	High

(c) For Change or Edit Service $(>=5)$

Table 28: Weight Evaluation for Each Template Changes Task

CMP	Type	Complexity Level		
		Low	Average	High
CMP_{code}	Change or Edit Service	1	2	7
	Create and Instantiation	1	4	9
	Add or Remove Attributes	1	3	6

5.2 Evaluation Scenarios

We devise scenarios that auto-scales data nodes of a benchmark application on three cloud platforms: AWS, Rackspace and Azure. The goal of these experiments is to evaluate the advantage of using model-driven for predictive auto-scaling service in cloud computing and to show how the deployment effort is reduced. We use the Netflix Data Benchmark (NDBench) and Dell DVD store application to evaluate our scenarios.

Scenario 1) Deployment Effort of Manual Process Cross Platforms: A hybrid cloud environment is, typically, a cloud computing environment that uses a combination of on premises, private cloud and public cloud services by combining between the two platforms. The combining is defined as allowing workloads to move between private and public clouds as computing needs and costs change. The hybrid cloud solutions give businesses greater flexibility and more data deployment options. Sometimes a monolithic middle-ware is used by hybrid cloud computing to integrate different services and resources. For instance, the public and private cloud may offer different auto-scaling techniques that are not compatible with each other even by using a middle-ware. To have the predictive auto-scaling system, we embedded machine leaning service to our general scaling strategy. However, there are still several steps to deploy the predictive auto-scaling system. So, the model-driven approach is acting as a bridge to facilitate the deployment process. For the first scenario, we are focusing on deployment effort differences across cloud platforms on the hybrid environment. In this case, we review the auto-scaling system on AWS and Rackspace. AWS and Rackspace offer different mechanisms for configuration and deployment for an auto-scaling service.

Scenario 2) Deployment Effort of Manual vs. MDD Without ML: In this scenario, we compare two categories for DVD store on AWS and Rackspace cloud platforms. In the first category, we deploy the auto-scaling system manually. Then,

Table 29: Manual procedure of auto-scaling data nodes

AWS	Rackspace
Download AWS Java SDK	Download Rackspace API
Install Java	Install Java
Configure username and password on Eclipse	Install Apache Jclou
Call Auto-scaling API	Provide URL for authorizing
Create EC2 Instance	Call Auto-scaling API
Create AMI	Create Cloud Server
Create Auto-scaling Group	Create Cloud Server Image
Create Launch Configuration	Create Auto-scaling Group
Create Auto-scaling Policy	Create Launch Configuration
-	Create Policy

we compare the result with the second category which contains deployment of the auto-scaling system with our proposed model-driven framework.

Scenario 3) Deployment Effort of MDD vs. MDD+ML: In this scenario, the evaluation of our model-driven framework focuses on the effort in terms of the impact of changes required between calling auto-scaling and predictive auto-scaling system for DVD store on AWS.

Scenario 4) Deployment Effort Manual ML vs. MDD+ML: We evaluate the effort of our model by the scenario of reconfiguring predictive auto-scaling service when the back-end cluster of NDBench is migrated from one cloud platform to another. We measure the effort in terms of the impact of changes on both clouds (AWS and Azure).

5.3 CMP Results

We calculate CMP for the manual deployment procedure for machine learning, manual deployment process of the threshold-based auto-scaling service, and our model-driven method to deploy the predictive auto-scaling service on the different cloud environment. The detail of calculations are presented in Tables 38, 39, 34, 35, 40, 41, 37, 36.

Scenario 1) For the first scenario, we are focusing on deployment effort differences across cloud platforms on a hybrid environment. In this case, we review the auto-scaling system on AWS and Rackspace. AWS and Rackspace offer different

Table 30: For **Scenario 1**, deployment effort differences for the manual procedure for the threshold-based auto-scaling system on AWS, Azure and Rackspace are represented. For **Scenario 2**, we calculate CMP for the manual deployment procedure for the threshold-based auto-scaling system, and our model-driven method to deploy the auto-scaling system without machine learning.

Category	DVD store		NDBench	
	Rackspace	AWS	Azure	AWS
Manual Auto-scaling	400	478	492	478
MDD Auto-scaling	123	123	123	123

mechanisms for configuration and deployment for an auto-scaling service. We present a comparison between manual procedures listed in Table 29.

Scenario-2) We calculate CMP for the manual deployment procedure for the threshold-based auto-scaling system and our model-driven method to deploy the auto-scaling system without machine learning. Figure 51 demonstrates the comparison between the mentioned scenarios for NDBench. There are two categories: Existing Auto-scaling and Auto-scaling with MDD without machine learning.

1. Existing Auto-scaling
    - Manual Auto-scaling service for DVD on AWS
    - Manual Auto-scaling service for DVD on Rackspace
2. Auto-scaling with MDD
    - MDD Auto-scaling service for DVD on AWS
    - MDD Auto-scaling service for DVD on Rackspace

Scenario-3) We calculate CMP for the MDD deployment procedure for auto-scaling system, and model-driven method to deploy the predictive auto-scaling system for DVD store on AWS cloud environment.

Scenario-4) We calculate CMP for the manual deployment procedure for machine learning, manual deployment process of the threshold based auto-scaling system, and our model-driven method to deploy the predictive auto-scaling system. Figure 52 demonstrates the comparison between the two-mentioned scenarios for NDBench. There are two groups: Existing Auto-scaling and Auto-scaling with MDD ML. We observe the reduced effort is approximately 25.3% for AWS and 26.6% for Azure.

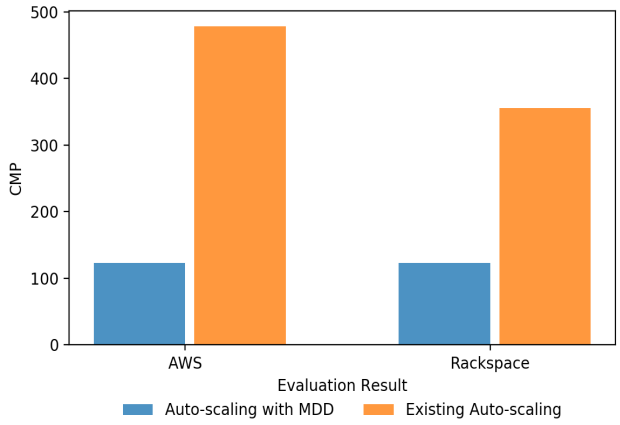


Figure 51: The CMP Result for the Manual Threshold-based Auto-scaling Service, and Proposed Model-driven Method Deployment Procedure Without Machine Learning

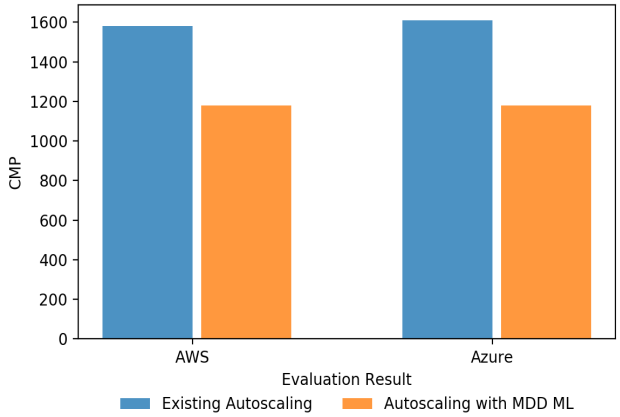


Figure 52: The CMP result for the Machine Learning, the Threshold based Auto-scaling service, and Proposed model-driven Method Deployment Procedure

Table 31: For **Scenario 3**, CMP results for the model-driven deployment procedure for the auto-scaling system, and the predictive auto-scaling system for DVD store on AWS.

Category	AWS	
	DVD Stoe	NDBench
Reactive Auto-scaling	123	123
Predictive Auto-scaling	1181	1181

Table 32: For **Scenario 4**, CMP results for the manual deployment procedure for machine learning service, manual deployment process of the threshold-based auto-scaling system, and our model-driven method to deploy the predictive auto-scaling system are represented.

Category	NDBench	
	Azure	AWS
Manual Auto-scaling	492	478
Manual Machine Learning	117	1103
MDD Predictive Auto-scaling	1181	1181

Table 33: Notation for CMP Calculation

CMP types	Symbol
Database Changes	$T1$
API Changes	$T2$
Create / instantiation	$T3$
Add attribute	$T4$
Edit /change service	$T5$
Infrastructure level	$T6$
Application level	$T7$

5.4 Summary

In this chapter, we answered the last research questions:

RQ 4. How to evaluate the effectiveness of the proposed approach?

Motivation: As effort is required for the deployment of the predictive auto-scaling on a target cloud platform and the amount of effort required is diverse, the effort estimation can illustrate the effectiveness of our proposition. We need to decide which metric to use in order to calculate the effectiveness of the proposed approach.

Approach: We calculate the effort in terms of the impact of changes in multi-clouds. We consider different scenarios to calculate deployment effort across multiple cloud environments. The evaluation of our model-driven framework focuses on the effort in terms of the impact of changes required between calling auto-scaling service with and without the support of our model-driven framework.

Table 34: Manual Threshold-based Auto-scaling Deployment Actions for AWS

Tasks	Types (Complexity)	Weight	CPM
Log in Aws	T7 (Low) /T4 (Low)	1 , 1	CMP (Ins) =1*1=1 CMP (code) =2*1=2
Instantiate Instance	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =3*9= 27 CMP (code) =7*3=21
Create Image (AMI)	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =2*3= 6 CMP (code) =3*3=9
Create Auto-scaling Group	T7 (Average) / T4 (Average)	2 , 3	CMP (Ins) =2*4= 8 CMP (code) =6*3=18
Create Launch Configuration	T7 (Average) / T4 (Average)	2, 3	CMP (Ins) =2*3= 6 CMP (code) =4*3=12
Create Scaling Policy	T7 (Average) / T4 (Average)	2, 3	CMP (Ins) =2*3= 6 CMP (code) =4*3=12
Total	CMP(code) =74		
	CMP (Ins) = 54		
	CMP (total) =74*5+ 54*2= 478		

Table 35: Manual Threshold-based Auto-scaling Deployment Actions for Azure

Tasks	Types (Complexity)	Weight	CPM
Log in Azure	T7 (Low) /T4 (Low)	1 , 1	CMP (Ins) =1*1=1 CMP (code) =2*1=2
Instantiate Instance	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =3*9= 27 CMP (code) =5*3=15
Create Managed Image	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =3*3= 9 CMP (code) =3*3=9
Create Resource Group	T7 (Average) / T4 (Average)	2 , 3	CMP (Ins) =3*4= 12 CMP (code) =5*3=15
Create Profile	T7 (Average) / T4 (Average)	2, 3	CMP (Ins) =2*3= 6 CMP (code) =5*3=15
Create Rules	T7 (Average) / T4 (Average)	2, 3	CMP (Ins) =2*3= 6 CMP (code) =6*3=18
Total	CMP(code) =74		
	CMP (Ins) = 61		
	CMP (total) =74*5+ 61*2= 492		

Table 36: MDD Auto-scaling Deployment Actions on AWS

Tasks	Types (Complexity)	Weight	CPM
Create CloudService Broker Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create AutoScaling Group Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =1*3 + 1*8 =11
Create Instance Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create BasicImage Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create ScalingPolicy Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*2 =3
Install Ansible	T7 (Average)	2	CMP (Ins) =1*2=2
Install Packer	T7 (Average)	2	CMP (Ins) =1*2=2
Total	CMP(code) =23		
	CMP (Ins) = 4		
	CMP (total) =23*5+ 4*2= 123		

Table 37: MDD Auto-scaling Deployment Actions on Rackspace

Tasks	Types (Complexity)	Weight	CPM
Create CloudService Broker Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create AutoScaling Group Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =1*3 + 1*8 =11
Create Instance Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create BasicImage Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create ScalingPolicy Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*2 =3
Install Ansible	T7 (Average)	2	CMP (Ins) =1*2=2
Install Packer	T7 (Average)	2	CMP (Ins) =1*2=2
Total	CMP(code) =23		
	CMP (Ins) = 4		
	CMP (total) =23*5+ 4*2= 123		

Table 38: Manual Machine Learning Deployment Actions for AWS

Tasks	Types (Complexity)	Weight	CPM
Log in Aws	T7 (Low) /T4 (Low)	1 , 1	CMP (Ins) =1*1=1 CMP (code) =2*1=2
Set up Security Group	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =2*3=6 CMP (code) =3*1=3
Set up VPC	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =2*3=6 CMP (code) =4*1=4
Create Lambda function	T6 (Average)/ T4 (Low)	3 , 1	CMP (Ins) =5*3=15 CMP (code) =5*1=5
Configure Setting for Lambda	T7 (Average) / T4 (Average)	3, 3	CMP (Ins) =4*3=12 CMP (code) =2*3=6
Configure API for Lambda	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2=4 CMP (code) =2*1=2
Create CloudWatch Monitoring	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =4*9= 36 CMP (code) =5*3=15
Configure API for Monitoring	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2= 4 CMP (code) =2*1=2
Create S3 Bucket storage	T1(High) / T4 (Low)	7 , 1	CMP (db) =3*7= 21 CMP (code) =2*1=2
Configure the API for Storage	T2 (Average) / T4 (Low)	3 , 1	CMP (db) =2*3= 6 CMP (code) =2*1=2
Create Machine Learning Instance	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =3*9= 27 CMP (code) =7*3=21
Create Image for Machine Learning Instance Instance	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =2*3= 6 CMP (code) =3*3=9
Deploy Python code for Machine Learning Algorithms	T7 (High) / T4 (High)	7 , 6	CMP (Ins) =3*7= 21 CMP (code) =6*10=60
Install Apache Server	T7 (Average)	2	CMP (Ins) =1*2= 2
Configure API for Machine Learning Instance	T7 (High) / T4 (Low)	7 , 1	CMP (Ins) =4*7= 28 CMP (code) =2*1=2
Configure API for Prediction	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2= 4 CMP (code) =2*1=2
Create DynamoDB table	T1 (High) / T4 (Low)	7 , 1	CMP (db) =3*7= 21 CMP (code) =2*1=2
Configure API DynamoDB	T2 (Average) / T4 (Low)	3 , 1	CMP (db) =2*3= 6 CMP (code) =2*1=2
Total	CMP(code) =141		
	CMP (db) = 54		
	CMP (Ins) = 172		
	CMP (total) =141*5+54*1+ 172*2= 1103		

Table 39: Manual Machine Learning Deployment Actions for Azure

Tasks	Types (Complexity)	Weight	CPM
Log in Azure	T7 (Low) /T4 (Low)	1 , 1	CMP (Ins) =1*1=1 CMP (code) =2*1=2
Set up Network Security Group	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =3*3=9 CMP (code) =4*1=4
Set up Route Table	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =3*3=9 CMP (code) =3*1=3
Create AppFunction function	T6 (Average)/ T4 (Low)	3 , 1	CMP (Ins) =5*3=15 CMP (code) =4*1=4
Configure Setting for AppFunction	T7 (Average) / T4 (Average)	3, 3	CMP (Ins) =4*3=12 CMP (code) =4*3=12
Configure API for AppFunction	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2=4 CMP (code) =2*1=2
Install Grafana Monitoring	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =5*9= 45 CMP (code) =6*3=18
Configure API for Monitoring	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2= 4 CMP (code) =2*1=2
Create Blob Storage Account	T1(High) / T4 (Low)	7 , 1	CMP (db) =3*7= 21 CMP (code) =2*1=2
Configure the API for Storage	T2 (Average) / T4 (Low)	3 , 1	CMP (db) =2*3= 6 CMP (code) =2*1=2
Create Machine Learning Instance	T6 (High) / T4 (Average)	9 , 3	CMP (Ins) =3*9= 27 CMP (code) =6*3=18
Create Image for Machine Learning Instance	T6 (Average) / T4 (Low)	3 , 1	CMP (Ins) =3*3= 9 CMP (code) =3*3=9
Deploy Python code for Machine Learning Algorithms	T7 (High) / T4 (High)	7 , 6	CMP (Ins) =3*7= 21 CMP (code) =6*10=60
Install Apache Server	T7 (Average)	2	CMP (Ins) =1*2= 2
Configure API for Machine Learning Instance	T7 (High) / T4 (Low)	7 , 1	CMP (Ins) =4*7= 28 CMP (code) =2*1=2
Configure API for Prediction	T7 (Average) / T4 (Low)	2 , 1	CMP (Ins) =2*2= 4 CMP (code) =2*1=2
Total	CMP(code) =142		
	CMP (db) = 27		
	CMP (Ins) = 190		
	CMP (total) =142*5+27*1+ 190*2= 1117		

Table 40: MDD ML Auto-scaling Deployment Actions on AWS

Tasks	Types (Complexity)	Weight	CPM
Create S3 (Data Storage) Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 2*1 =3
Create DynamoDB (Data Storage) Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 2*1 =3
Create Lambda (Controller) Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create GBR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 9*4 =45
Create SVR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 9*4 =45
Create LR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 5*4 =29
Create Predictor Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 6*4 =33
Create CloudWatch (Monitor) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =2*3 + 5*4 =26
Create CloudService Broker Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create AutoScalingGroup Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =1*3 + 1*8 =11
Create Instance Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create BasicImage Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create ScalingPolicy Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*2 =3
Create Stack Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Install Ansible	T7 (Average)	2	CMP (Ins) =1*2=2
Install Packer	T7 (Average)	2	CMP (Ins) =1*2=2
Total	CMP(code) =233		
	CMP (Ins) = 8		
	CMP (total) =233*5+8*2= 1181		

Table 41: MDD ML Auto-scaling Deployment Actions on Azure

Tasks	Types (Complexity)	Weight	CPM
Create Blob (Data Storage) Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 2*1 =3
Create FunctionApp (Controller) Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create GBR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 9*4 =45
Create SVR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 9*4 =45
Create LR (MachineLearning) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 6*4 =33
Create Predictor Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =3*3 + 6*4 =33
Create Grafana (Monitor) Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =2*3 + 5*4 =26
Create CloudService Broker Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 3*1 =4
Create ResourceGroup Object	T4 (Average)/ T3 (Average)	3 , 4	CMP (code) =1*3 + 1*6 =9
Create Instance Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Create BasicImage Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*4 =5
Create ScalingPolicy Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*2 =3
Create Stack Object	T4 (low)/ T3 (low)	1 , 1	CMP (code) =1*1 + 1*3 =4
Install Ansible	T7 (Average)	2	CMP (Ins) =1*2=2
Install Packer	T7 (Average)	2	CMP (Ins) =1*2=2
Create EGL template	T4 (High) / T3 (High)	6 ,9	CMP (code) =1*6 + 1*9 =15
Total	CMP(code) =233		
	CMP (Ins) = 8		
	CMP (total) =233*5+8*2= 1181		

Chapter 6

Conclusion and Publications

In this chapter, we summarize the research contributions. We explain challenges and limitation. Also, we address the future work. Then, we list the publications related to this work.

6.1 Threats to Validity

Change of Platform: The architecture should be generally applicable on different cloud platforms. Currently, we have implemented our proposed architecture on the AWS and Azure. If we want to implement this architecture on other cloud platform, we need to map the architecture to target cloud services. Finding the right modules with right assignment of responsibilities with well-designed interfaces is a challenge and each service has some limitations and open issues. For instance, each cloud provider offers different type of monitor service and there are some limitation for each one. Amazon Cloudwatch monitoring service provides hypervisor-specific metrics. To have OS-specific metrics such as memory utilization, we need to add an extra monitoring script. So, we can collect the percentage of system memory as a custom metric. While, Azure monitor service has some limitations for retrieving the monitor metrics including limits the number of rows that can be retrieved in one call to 1000 rows.

Change of Application: In this work, the Ansible tries to reduce the manual tasks when the new instance is added to the cluster. However, the new applications may have some limitations for using the Ansible for reconfiguring the cluster. In

addition, deploying the new application on target Cloud platform can introduce new issues. Application versions also are changed regularly and sometimes new versions introduce limitation.

Continuous Training: Continuous model training allows models retrain and updates over a period of time. The advantage of training over the arbitrary intervals is that the model has a higher probability to catch more workload patterns and always the model is trained by recent workload history. In this paper, we have two weeks window time for training because two weeks is enough to catch the pattern. However, if we want to have an accurate model, we need to reconfigure our process in order to have an arbitrary intervals for training stage.

System Level Metric: Our work is limited to collect only system level metrics (CPU usage and memory), so we did not consider application level metrics like response time or number of requests.

6.2 Future Work

This thesis presented contributions in the area of model-driven machine learning for auto-scaling system. Yet, there exist research directions for the future:

Clusters are sets of servers that are managed together and participate in workload management. Through the thesis, we assumed that the workloads are balance in a cluster. It means workloads are distributed across all members of a cluster equally. This raises the challenge of scaling the resources with an unbalanced workload. It would be interesting to investigate the forecasting-based auto-scaling to manage resource of cluster with an unbalance workload.

This thesis for predictive auto-scaling is using infrastructure level (system-level) metrics such as CPU/memory/network utilization. However, auto-scaling uses not only infrastructure, but also application-level monitoring data. In the future, it would be interesting to study on how effective predictive auto-scaling can use application-level metrics such as throughput and response time.

6.3 Conclusion

In this work, we present a model-driven framework to automate the operations of predictive auto-scaling service design, deployment and launching on multiple clouds. Our approach contains two parts of models: one for modeling machine learning models and auto-scaling service independent of a cloud platform; and the other for model cloud-specific predictive auto-scaling process. To connect the design level models with the deployment of a cloud platform, we propose the transformation from models to deployment scripts and launch cloud management tools within a single integrated modeling environment. A practical case of this framework is evaluated on both AWS, Rackspace and Azure clouds. Our contribution is to hide the technical details of developing cloud provider specific auto-scaling operations with machine learning techniques.

Also, we present the microservice architecture for forecasting-based auto-scaling process that adaptively monitors the workload based on multi metrics and schedules multiple machine learning models to learn the workload pattern online and predict the workload classification at runtime. The process of model training, model validation, model selection, and prediction are decoupled into separate microservices. For evaluation, we use Dell DVD Store and Netflix Data Benchmark that are designed to explore the performance impact generated from microservices. We applied the proposed solution on Amazon Web Services and Azure cloud environment, also three machine learning regression algorithms. We demonstrate the real-time prediction is integrated to the auto-scaling configuration of a cloud infrastructure to add or remove computing resources. This forecasting-based solution is independent of distributed framework and thus is applicable to other cloud infrastructures.

6.4 Publication

I listed our publication here:

- Alipour, Hanieh, Yan Liu, and Abdelwahab Hamou-Lhadj. "Analyzing auto-scaling issues in cloud environments." In Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, pp. 75-89. IBM Corp., 2014.

- Alipour, Hanieh, Yan Liu, Abdelwahab Hamou-Lhadj, and Ian Gorton. "Model driven performance simulation of cloud provisioned Hadoop MapReduce applications." In Proceedings of the 8th International Workshop on Modeling in Software Engineering, pp. 48-54. ACM, 2016.

- Alipour, Hanieh, and Yan Liu. "A model driven method to deploy auto-scaling configuration for cloud services." In Proceedings of the 4th International Workshop on Release Engineering, pp. 23-23. ACM, 2016.

- Alipour, Hanieh, and Yan Liu. "Online machine learning for cloud resource provisioning of microservice backend systems." In 2017 IEEE International Conference on Big Data (Big Data), pp. 2433-2441. IEEE, 2017.

- Alipour, Hanieh, and Yan Liu. "Model Driven Deployment of Auto-Scaling Services on Multiple Clouds." In 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 93-96. IEEE, 2018.

- Alipour, Hanieh, and Yan Liu. "Microservice Orchestration to Inference-based Cloud Workload Auto-scaling", IEEE Transactions Cloud Computing, 2019 (Submitted[0]).

- Alipour, Hanieh, and Yan Liu, Abdelwahab Hamou-Lhadj."Model-Driven Machine Learning for Predictive Cloud Auto-scaling", IEEE Transactions Software Engineering, 2019 (Submitted).

.

[0]We submitted our journal paper in IEEE Transaction Service Computing (TSC) on 27-Jun-2018 and we received the first review on 13-Nov-2018. So, we submitted the revision version on 08-Dec-2018 and unfortunately because of the leave of absence of the chair, we didnt receive the answer after 6 months. So, we decided to withdraw our paper and submit it in IEEE Transaction Cloud Computing.

Bibliography

[1] Ansible. https://www.ansible.com/.

[2] Aws auto-scaling. http://aws.amazon.com/documentation/autoscaling.

[3] Azure. https://azure.microsoft.com.

[4] Bi-lstm. www.cl.cam.ac.uk/~pv273/slides/LSTMslides.pdf.

[5] Cassandra. http://cassandra.apache.org.

[6] Cassandra configuration. https://www.digitalocean.com.

[7] Counting practices manual. http://www.ifpug.org/.

[8] Dvd store. http://linux.dell.com/dvdstore/.

[9] Gartner. http://www.gartner.com/technology/home.jsp.

[10] Grafana. https://grafana.com/.

[11] Implementation codes. https://github.com/haniehalipour/Online-Machine-Learning-for-Cloud-Resource-Provisioning-of-Microservice.

[12] iperf3. https://iperf.fr/iperf-download.php/.

[13] Lstm. www.mathworks.com/help/deeplearning.

[14] Netflix. https://medium.com/netflix-techblog.

[15] Packer. https://www.packer.io.

[16] Prometheus. https://prometheus.io/.

[17] Scikit-learn. https://scikit-learn.org/stable/.

[18] What is auto-scaling? `http://docs.rightscale.com/faq/What_is_auto-scaling.html`.

[19] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[20] Timo Aho, Bernard Ženko, Sašo Džeroski, and Tapio Elomaa. Multi-target regression with rule ensembles. *Journal of Machine Learning Research*, 13(Aug):2367–2407, 2012.

[21] Samuel Adesoye Ajila and Akindele A. Bankole. Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment. In *Transactions on Machine Learning and Artificial Intelligence 4*, 2016.

[22] Hammam M AlGhmadi, Mark D Syer, Weiyi Shang, and Ahmed E Hassan. An automated approach for recommending when to stop performance tests. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 279–289. IEEE, 2016.

[23] Hammam M AlGhmadi, Mark D Syer, Weiyi Shang, and Ahmed E Hassan. An automated approach for recommending when to stop performance tests. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 279–289. IEEE, 2016.

[24] Hanieh Alipour, Yan Liu, and Abdelwahab Hamou-Lhadj. Analyzing auto-scaling issues in cloud environments. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, pages 75–89. IBM Corp., 2014.

[25] Danilo Ardagna, Elisabetta Di Nitto, Giuliano Casale, Dana Petcu, Parastoo Mohagheghi, Sébastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, et al. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Proceedings of the 4th international workshop on modeling in software engineering*, pages 50–56. IEEE Press, 2012.

[26] Matthias Becker, Markus Luckey, and Steffen Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures*, pages 117–122. ACM, 2012.

[27] Dario Bruneo, Thomas Fritz, Sharon Keidar-Barner, Philipp Leitner, Francesco Longo, Clarissa Marquezan, Andreas Metzger, Klaus Pohl, Antonio Puliafito, Danny Raz, Andreas Roth, Eliot Salant, Itai Segall, Massimo Villari, Yaron Wolfsthal, and Chris Woods. CloudWave: Where adaptive cloud management meets DevOps. In *2014 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2014.

[28] Chris Bunch, Vaibhav Arora, Navraj Chohan, Chandra Krintz, Shashank Hegde, and Ankit Srivastava. A pluggable autoscaling service for open cloud paas systems. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 191–194. IEEE Computer Society, 2012.

[29] Faruk Caglar, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. Model-driven performance estimation, deployment, and resource management for cloud-hosted services. In *Proceedings of the 2013 ACM workshop on Domain-specific modeling*. ACM Press, 2013.

[30] Lequn Chen Pedro Fonseca Tianqi Chen Chern Cheah Karan Gupta Ramesh Chandra Cano, Ignacio and Arvind Krishnamurthy. Adares: Adaptive resource management for virtual machines. In *arXiv preprint arXiv:1812.01837*, 2018.

[31] Salvatore Capra. Cloud computing trace characterization and synthetic workload generation. 2013.

[32] Krishna Varaynya Chivukula. Monitoring and analysis of cpu load relationships between host and guests in a cloud networking infrastructure: An empirical study, 2015.

[33] Filippo Lorenzo Ferraris, Davide Franceschelli, Mario Pio Gioiosa, Donato Lucia, Danilo Ardagna, Elisabetta Di Nitto, and Tabassum Sharif. Evaluating the auto scaling performance of flexiscale and amazon ec2 clouds. In *2012 14th*

International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pages 423–429. IEEE, 2012.

[34] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013.

[35] Nicolas Ferry, Hui Song, Alessandro Rossini, Franck Chauvel, and Arnor Solberg. CloudMF: Applying MDE to tame the complexity of managing multi-cloud applications. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014.

[36] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing ({ICAC} 14)*, pages 57–64, 2014.

[37] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Model-driven optimal resource scaling in cloud. *Softw. Syst. Model.*, 17(2):509–526, 2018.

[38] Mostafa Ghobaei-Arani, Sam Jabbehdari, and Mohammad Ali Pourmina. An autonomic approach for resource provisioning of cloud services. *Cluster Computing*, 19(3):1017–1036, 2016.

[39] Michele Guerriero, Michele Ciavotta, Giovanni Paolo Gibilisco, and Danilo Ardagna. A model-driven DevOps framework for QoS-aware cloud applications. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2015.

[40] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S McKinley, and Björn B Brandenburg. Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 109–120. ACM, 2017.

[41] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*, pages 23–27, 2013.

[42] Waheed Iqbal, Mathew N Dailey, and David Carrera. Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier web applications. *IEEE Systems Journal*, 10(4):1435–1446, 2016.

[43] Yujuan Jiang and Bram Adams. Co-evolution of infrastructure and source code - an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015.

[44] Hui Kang, Michael Le, and Shu Tao. Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211. IEEE, 2016.

[45] Amin Karami. Utilization and comparison of multi attribute decision making techniques to rank bayesian network options, 2011.

[46] Mehran NAH Khan, Yan Liu, Hanieh Alipour, and Samneet Singh. Modeling the autoscaling operations in cloud with time series data. In *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, pages 7–12. IEEE, 2015.

[47] Dragi Kocev, Sašo Džeroski, Matt D White, Graeme R Newell, and Peter Griffioen. Using single-and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecological Modelling*, 220(8):1159–1168, 2009.

[48] Zheng Li, Liam O'brien, He Zhang, and Rainbow Cai. On a catalogue of metrics for evaluating commercial cloud services. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 164–173. IEEE Computer Society, 2012.

[49] Ying Liu, Navaneeth Rameshan, Enric Monte, Vladimir Vlassov, and Leandro Navarro. Prorenata: Proactive and reactive tuning to scale a distributed storage system. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 453–464. IEEE, 2015.

[50] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department*

of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09, 12:2012, 2012.

[51] Mohamed Mohamed, Mourad Amziani, Djamel Belaïd, Samir Tata, and Tarek Melliti. An autonomic approach to manage elasticity of business processes in the cloud. *Future Gener. Comput. Syst.*, 50:49–61, 2015.

[52] J.C. Munson and S.G. Elbaum. Code churn: a measure for estimating the impact of code change. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE Comput. Soc.

[53] Thanh HD Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 299–310. ACM, 2012.

[54] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 35–45. IEEE Press, 2015.

[55] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507. IEEE, 2011.

[56] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1-2):460–471, 2010.

[57] Weiyi Shang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using regression models on clustered performance counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 15–26. ACM, 2015.

[58] Weiyi Shang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using regression models on clustered

performance counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 15–26. ACM, 2015.

[59] Ritu Sharma and Manu Sood. Enhancing cloud saas development with model driven architecture. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, 1(3):89–102, 2011.

[60] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. Does your configuration code smell? In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM Press, 2016.

[61] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Transactions on Computers*, 63(11):2647–2660, 2014.

[62] Yu Sun, Jules White, Sean Eade, and Douglas C Schmidt. Roar: A qos-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software*, 116:146–161, 2016.

[63] Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Florian Dudouet, and Andrew Edmonds. An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, pages 19–24. ACM, 2015.

[64] Van TK Tran, Kevin Lee, Alan Fekete, Anna Liu, and Jacky Keung. Size estimation of cloud migration projects with cloud migration point (cmp). In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 265–274. IEEE, 2011.

[65] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Aikaterini Vrekou, and Ioannis Vlahavas. Multi-target regression via random linear target combinations. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 225–240. Springer, 2014.

[66] Aparna Vijaya, V. Neelanarayanan, and V. Vijayakumar. *Framework for Supporting Heterogenous Clouds Using Model Driven Approach*, pages 219–235. Springer International Publishing, 2015.

[67] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590. IEEE, 2015.

[68] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182. IEEE, 2016.

[69] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[70] Muhammad Wajahat, Anshul Gandhi, Alexei Karve, and Andrzej Kochut. Using machine learning for black-box autoscaling. In *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, pages 1–8. IEEE, 2016.

[71] Johannes Wettinger, Michael Behrendt, Tobias Binz, and all. Integrating configuration management with model-driven cloud management based on tosca. In *In Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER). SciTePress*, 2013.

[72] Chung-Hsing Yeh. A problem-based selection of multi-attribute decision-making methods. *International Transactions in Operational Research*, 9(2):169–181, 2002.