National College of Ireland

Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool

MSc Research Project
Cloud Computing

Shinoj Pittandavida
Student ID: x17169704

School of Computing
National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shinoj Pittandavida
Student ID:	x17169704
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Victor Del Rosal
Submission Due Date:	20/12/2018
Project Title:	Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool
Word Count:	6537
Page Count:	20

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

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

Signature:	
Date:	28th January 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool

Shinoj Pittandavida
x17169704

Abstract

As Cloud computing has evolved to a new level, every organization needs to adopt the changes to support the new technology and design an infrastructure that supports fault tolerance. Any organization uses virtualization technology for their infrastructure should develop a feature to support the reliable, fault-tolerant and high-available resources. We propose a new model named Autodeployer, to mitigate the failures of Amazon Web Services (AWS) Elastic Cloud Compute (EC2) instances with a batch process to reduce the launch time. The Autodeployer will help to recover the failed instances in seconds. The available tools in the market are either limited with instance creation or batch process and do not support auto-recovery for the batch of EC2 instances. They require additional licenses and more domain knowledge for the automation. Our approach is based on the Application Programming Interface (API) iteration process which greatly reduces the time of the manual process. This process is different from the method being used by AWS CloudWatch technic and saves more space for additional log files. The proposed Autodeployer model can make the batch process with failure recovery with 70 to 85 percent faster than the other services. It can optimize the additional time and effort needed to configure a large number of instances at once and loss of service availability issues. In this paper, we present the solution for mitigating the failures in the batch EC2 instance with auto recovery.

1 Introduction

Cloud computing today provides wide exposure to different types of computing resources like instance that is necessary for any organization. This instance can be servers or virtual machines running on the distributed cloud. Companies used on-premise infrastructure are now moved to faster and reliable cloud infrastructure. But the equal responsibility policy in the AWS computes services to make the customers more responsible in case of failures. Mitigating failure in AWS is bit complex than the on-premise infrastructure because the complexity of the infrastructure leads increased chances of failures Samak et al. (2012). To provide the high availability or failure recovery requires configuration of EC2 instances in multiple regions. This configuration adds additional cost for the services used by AWS customer. Introducing a model to address this challenge and mitigate the failure is worth in the complex and distributed cloud.

Continuous disaster tolerance is the ability of a computing infrastructure system to endure from the loss of connection, hardware or software failure, loss of power, etc. The outage of service may take a short period or sometimes uncertain to recover from the failure Caraman et al. (2012). An efficient failure recovery system can mitigate the above outage process and recover automatically in seconds. The available services currently used by the cloud service providers have distributed management system with redundant components that are located on the different regions or continent Anarado and Andreopoulos (2016). Amazon Web Services offers Elastic Load Balancer (ELB) to provide high availability and redundant service to their customers.

Amazon Web Service offers a large set of EC2 instance types for growing infrastructure needs. Selecting a suitable instance type for running peak hours requires batch instance creation or batch instance processing Chard et al. (2016). Batch instance launching is a time-consuming process in case of the instance with multiple configurations. There are multiple chances for failures of instance due to the configuration mismatch of chosen instance type, Amazon Machine Image (AMI), incorrect Virtual Private Cloud Configuration (VPC) and incorrect network connections Varia (2010). Our proposed automation tool fill this space by auto-recovering the failed batch instances with desired configurations.

The current trends called automation using profiling services helps the cloud infrastructure to manage, deploy and operates an inefficient manner. It uses Infrastructure as Code (IaC) concept and it will be the future of automation since the code can be reused for other application. The profiling service can be used in AWS to define the computing power, type of network, type and capacity of the storage device and higher level security Chard et al. (2016). The main disadvantage of these profiling systems are the chances of failure on the configuration change during peak working hours. In order to check the performance of the instances created on the cloud, the benchmarking needs to be done by running several scientific application on top of it Jackson et al. (2010).

In order to mitigate the challenges of batch processing and failures in the AWS EC2 instances, a script based Autodeployer tool can be implemented. The proposed tool can reduce the time needed to create a batch of EC2 instances in seconds with the resistance to failure and auto recovery in case of bulk instance launch. The proposed functional diagram of Autodeployer is shown in Figure 1

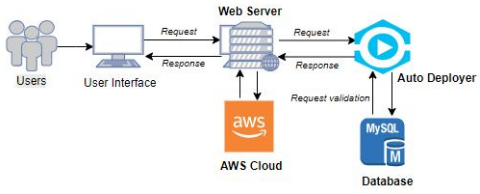

Figure 1: Proposed Autodeployer Functional diagram

This research paper presents the batch processing of EC2 instance with a solution to auto recovery in case of failure in AWS. This research paper is organized as following seven sections. Section 2 describes the current research works on the failure recovery and automation. Section 3 defines the research methodology and the evaluation methodology. Section 4 introduces the design specification including the architecture and framework. Section 5 presents the implementation of the proposed solution. Section 6 highlights the

evaluation of the research by analyzing the results and findings. Section 7 concludes this research with the future scope of work.

2 Related Work

Technology is evolving day by day and every business needs to be up to date. Over the years, companies maintained their data center to keep their data safe are now moved to cloud infrastructure. Some of them are maintaining owned infrastructure connected with the cloud to expand their infrastructure. Regardless of the compute and storage service, the cloud offers more application-based services and technologies. Rapid change in the field of cloud needs more precise and easy to set up infrastructure tools for provisioning and recovery in case of failure. Cloud automation and DevOps are the way to provisioning infrastructure but few are used for failure recovery. Among the different approaches used in the cloud to manage the instance creation and failure mitigation are briefing below sections.

2.1 Instance failure mitigation and recovery

Amazon EC2 instance is the most popular virtual compute service, which provides Infrastructure as a Service (IaaS) in terms of high performance, ease of setup and quality of service. Cloud users can create, configure, launch, terminate and configure their instance based on the requirements. Amazon offers an interface in the form of APIs and SOAP, through which users can develop an application to automate their server instances Caraman et al. (2012). One of the methods used to mitigate the failure was by implementing disaster tolerance solutions using live virtual machine migration with storage replication. This is a more expensive method to provide high availability since it requires multiple virtual servers on the cloud that support live migration Cully et al. (2008).

Another approach used for instance failure recovery was by using the seven-stage disaster tolerant (DT) algorithm. DT algorithm involves two-stage servers which are very similar to the normal data center failure setup. It uses two host systems one primary, secondary and these hosts are placed in different geographic locations with high-speed connection Caraman et al. (2012). It associate the seven stages including, disk replication with network protection, virtual machine check point, check point synchronization, backup replication, backup synchronization, failure detection and fail-over. Every organization can't afford the prize of configuring such system for mitigating the failures and this method is very similar to the AWS Elastic Load Balancer (ELB). The seven stage disaster tolerant algorithm is shown in Figure 2

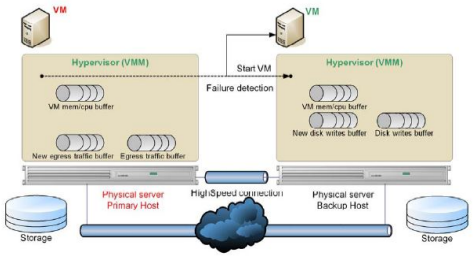

Figure 2: Seven stage DT algorithm Caraman et al. (2012)

Graph-based mitigation framework was another solution for the server instances to recover from the failure. Based on the annotated attack graph technique it identifies the potential threads to failure. This mitigation feature does not provide the recovery of the server instance but offers the mitigation to the threats Datta and Goyal (2014). These systems are more into secure the instances from attack and reduce the chance to fail. The disadvantage of the above systems is their inability to recover in case of failure Datta and Goyal (2014). The detection algorithm approach uses a different method to detect the failures. It has a virtual machine monitor that always checks the status of the system and report any information to fault tolerant manager (FTM) Gokhroo et al. (2017). Limitation of this method was once the error has been found, the service provider needs to take any further action and there is no automatic recovery system in this method.

2.2 Distributed coordinated check point for automatic recovery

In order to achieve better results and performance for scientific applications, High-Performance Computing (HPC) uses fault tolerance systems. HPC systems are needed to improve the computation of large scientific applications. In HPC system a reduced Mean Time To Repair(MTTR) is essential to provide high availability of service. Fault-Tolerant systems are the only solution to achieve this. The research paper Villamayor et al. (2017) says that fault-tolerant manager (FTM) with distributed checkpoint systems can be used for automatic recovery in computing node failures. Their approach to recover the failures is based on saving the checkpoints on the local storage and distribute them on the computing nodes in the systems.

Roll-back recovery method is another type of widely used technique where a frequent snapshot of the system status is taken. Here the snapshots are considered as checkpoints and whenever a failure happens, the most recent snapshot will be taken to restore the system Villamayor et al. (2017). The main implication in this approach is whenever a hard failure occurs in the system a human intervention needed to restore the system back to work. This problem will increase the MTTR value and it causes a break in Service Level Agreement (SLA) and the firm should pay for the outage. An automatic recovery mechanism is required to solve this issue and the solution is fault tolerant system. Villamayor et al. (2017).

The performance evaluation of the checkpoint or restart technique is detailed in the paper Azeem and Helal (2014). Where the performance of the systems in distributed architecture and single mode is tested. The analysis shows the applications running on the distributed systems or in the cloud takes less time for the execution. But unexpected applications failure happens due to the unpredictable computing system failures Azeem and Helal (2014). The most widely used method called coordinated checkpoint uses system snapshot of running processes and distribute them in the system.

2.3 Roll-forward failure mitigation approach

This approach differs from other failure mitigation methods in terms of a solution that does not require any type of check-sum or duplicate results from the virtual machines. These type of mitigation system are mostly used for high-performance clusters. AWS offers spot instances with less cost attract users to set up high-performance clusters. It is very much needed to set up a failure tolerant system whenever using spot instances because AWS has the right to terminating a spot instance anytime. Roll-forward method

guarantees the recovery from the failure without re-computation of the processor cores. Forward error recovery system uses a predefined and stored data from the check-sum. The major disadvantage of this approach is, this method is more detailed to the computational level failure recovery rather than server instance level. Forward error recovery approach has more overhead when comparing to other processor level approaches due to the high storage and processing of check-sums Anarado and Andreopoulos (2016).

2.4 Automation in batch compute instance creation

Most of the commercial application widely used public cloud services like AWS for reliable and cost-effective deployments. Growing infrastructure leads the companies to add more server instances to their infrastructure to run the applications smoothly. In order to reduce the time constraints in launching multiple server instances at once requires batch processing. There are some research happened on this area to cut-down the cloud computing cost for batch jobs. The main reason behind this approach is the cost for the IT infrastructure mainly lays on the running server instances. Maintaining cost-effective and efficient infrastructure is a tedious task and it needed to verify the type of instance before launching them. The research paper Zhang et al. (2010) proposed an idea of brokerage service to maintain a pool of server instances.

The broker service accepts the inputs from the web service providers and provides optimal strategies which minimize their cost for instances. This approach can be used for both reserved and spot instance batch processing. Even without using any of the spot instances, using the broker service it can significantly reduce the cost of infrastructure. In order to satisfy the computation, the broker will distribute its load in a time window. By distributing the load it reduces the peak hour demands increases the efficient usage of the reserved instances and hence reduce the cost for new instances Fox et al. (2009). The framework for the cloud brokerage service is shown in Figure 3

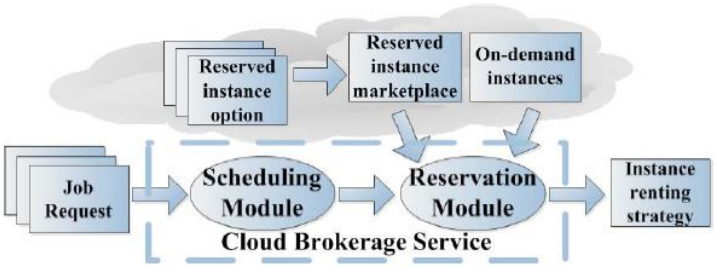

Figure 3: Framework for cloud brokerage service Fox et al. (2009)

Regardless of the other research discussed above on the instance automation and batch process, the objective of this batch process scheme is only to optimize the instance renting cost. The Yao et al. (2014) proposed research does not solve any failures of single or batch instances. This approach has chances of failure if any distributed nodes down since the scheduling module always used the distributed nodes to meet the peak demand. If the rate of failure in the distributed nodes is high then this approach can't be used to exploit the batch processing. This leads new approach for both batch process with failure tolerance.

2.5 Concluding remarks on related work

All the above research on the failure recovery, batch processing and automation of server instances on the cloud services gives strength and weakness of the different approach. Most of the above research is either limited to fault tolerance or to create an instance in public or other cloud environments. The problem identified here is none of the script automation gives a solution to create batch instances with auto recovery. The problem of replication and auto recovery can be addressed with our proposed Autodeployer automation tool.

3 Methodology

Based on the knowledge and experience gained from other research on the instance automation, our research will focus on developing a new script based Autodeployer automation tool. Autodeployer will target the identified problem on the previous research and solve the batch instance process with fail-over. The proposed script automation tool can address the issues identified in the related work and greatly optimize the instance launching time and recovery in case of failures. The previous research on related work in section 2 clearly mention that the existing automation tools are either limited to the instance profiling service or fault-tolerant. None of them are particularly helping in batch AWS EC2 instance creation process with automatic recovery. Our proposed approach has several phases including procuring cloud data, analyzing data, choosing an appropriate script and some activities for batch instance creation and automatic recovery. Also to test the feasibility of Autodeployer tool that runs on top of AWS EC2 instance, we conduct some experiments to analyze the optimal strategy for failure recovery in batch instance process.

3.1 Research Question

The research question addressed by this paper is:

What is the effect on instance launch time using script based Autodeployer tool in Amazon Web Services to mitigate the EC2 instance failure with automatic recovery?

3.2 Procurement of cloud user data

The initial process of Autodeployer automation tool is to collect the cloud user data. A cloud user data can be an associated ID and type of the instance, status or configurations of any instance. It is very important to gather the appropriate instance data because batch processing and auto-recovery are based on the instance data. The instance data is the input for Autodeployer, it can be instance id for the particular instance having the issue, configurations of the instance or it can be a group of different instance type. Once the input has identified, the Autodeployer will read the value from the data and perform appropriate actions.

In Table 1 is provided with the cloud user data which is the input to the Autodeployer tool. The input user data can be changed based on the customer requirements.

The value of the AMI ID is always different for the type of Operating System used. Users have the freedom to choose AMI based on the application requirements. Instance type can vary based on the availability zone. Some type of instances won't be available

Table 1: AWS EC2 Cloud user data.

Input variables	Data type
AMI ID	String
Availability zone	String
Instance Type	String
Min Count	Number
Max Count	Number
Monitoring	Boolean

in some regions. Min and max count give users to specify the number of the instance and monitoring will keep on check the status.

3.3 Analyze and rendering of data

Analyzing deployed instances is mandatory to detect and provide continuous fault-tolerance and automatic recovery in case of failures. Autodeployer keeps on checking status and configurations of each instance to analyze the anomaly and respond accordingly. The first process starts with the user input to examine the data provided are sufficient to create and manage instances in batch processing. If not the system will halt and ask the user to input the valid data values. Table 2 shows the input of the analyzed data for the batch process.

Table 2: Batch process data inputs.

AMI ID	Instance Type	O.S
ami-00035f41c82244dab	t1.micro	Ubuntu Server
ami-08935252a36e25f85	t1.small	Amazon Linux
ami-0e12cbde3e77cbb98	t2.micro	Red Hat
ami-07e2f0b6f6b5aeea3	t2.large	Windows server 2016
ami-050889503ddaec473	t2.small	Suse Linux

The AMI ID of the virtual machine is not associated with a particular instance type, users can choose their own instance type based on demand. Rendering is the action carried out once the input data is valid and set. Autodeployer reads the data from the excel files or CSV files where all the configuration are written. Rendering module has a set of packages that reads the data raw by raw from the sheet. The automation script in the Autodeployer will process the data and launch the instances quickly. Once the instance starts running, each instance status will be monitored continuously to predict the uncertain behavior and restart the failed instance.

3.4 Applying desired automation script

Batch instance processing and automatic recovery process are working hand in hand but they are two distinct processes in the Autodeployer. Instance creation and batch instance processing are initial script procedures. Continuous disaster tolerance and auto recovery methods work after successful instance creation steps. Autodeployer has an additional feature that gives users the freedom to select only the instances that require attention.

Therefore the Autodeployer select the appropriate script based on the user request. Major script models in the Autodeployer are briefing below.

Instance creation script function at the initial step of Autodeployer model. It accepts the user request and starts the instance on a single click. The selection of the instance type purely lies on the customer side. The script reads the input data of the particular request and starts the instance using AWS API interface less than 10 seconds. Batch Instance replication script performs the bulk instance creation depend on the data in the excel sheet. This operation carried out by the script is purely based on the data provided in the manual. If the organization requires hundreds of the different instance's type with the different configurations, batch instance script will be running. Instance auto recovery script functions after the instance creation. It always keeps on tracking the running instances with keep-alive status on. Keep-alive is used to control the instances that require attention. Users can choose the instance ids that require the keep-alive active for auto recovery.

3.5 Instance creation activity

Instance creation activity is for launching a single instance with configuration. The instance creation module reads the user input after user authentication with the database. After successful authentication, it selects the script and calls the AWS API to create an instance. Back-end application passes the control to AWS to create the requested instance only if the output is a success. Web API for Autodeployer display the created instance with ID. Instance creation activity is shown in Figure 4

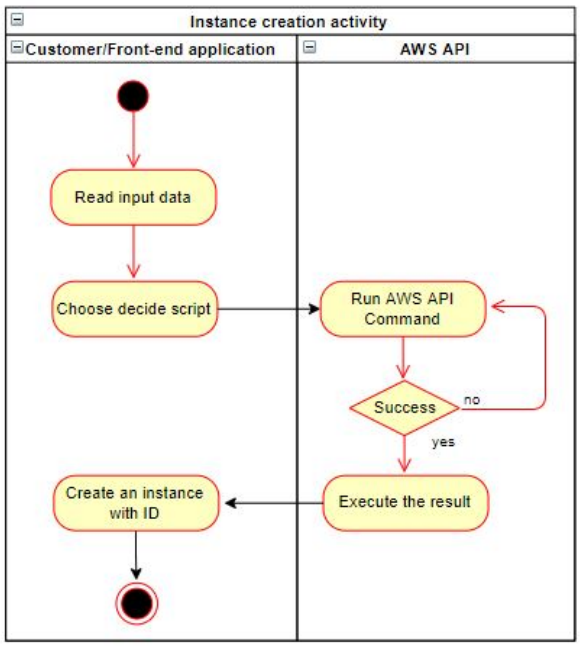

Figure 4: Instance creation activity

3.6 Batch instance creation activity

Batch instance processing is for dynamic infrastructure deployment. After successful authentication batch script module waits for user input in the form of excel file. Autodeployer will check the file format after uploading the input data file. After successful validation batch processing module call the AWS API for start bulk instances based on the uploaded input data file. If the process pass through it creates multiple instances with a unique id. It repeats the process until all the instance creation. Batch instance process is shown in Figure 5

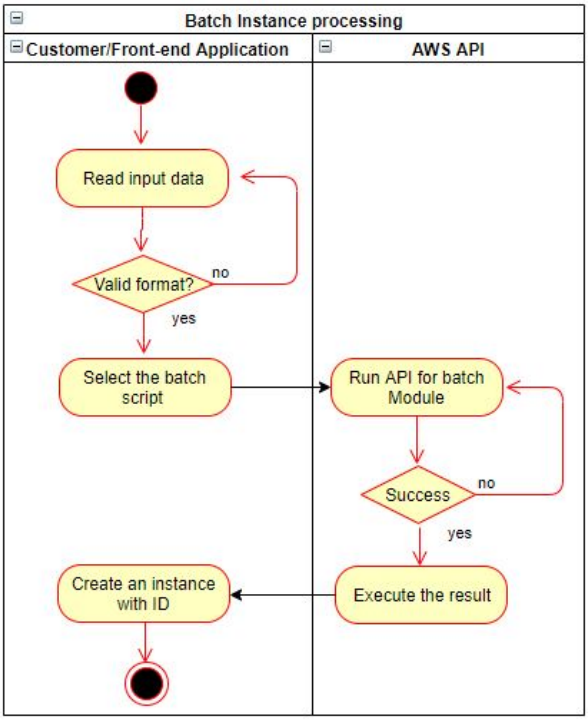

Figure 5: Batch Instance creation activity

3.7 Instance auto-recovery activity

Instance auto-recovery process starts only if the system finds an unusual behavior in the instance status. The recovery module in the Autodeployer monitoring status of all the instance with keep-alive value ON. Whenever status of an instance or multiple instances fail, the recovery module flag the error and recover the instance and restart in case of failure. Unlike AWS CloudWatch alarm, Autodeployer won't wait for the status to report to the user. It performs the recovery process right after the instance status fails. Hence this model greatly reduces the failure time and recover the instance automatically. Instance auto-recover activity is shown in Figure 6

4 Design Specification

This research is focused on developing a new automation model for Amazon Web Services EC2 instance failure recovery. Our proposed model is called Autodeployer which is cap-

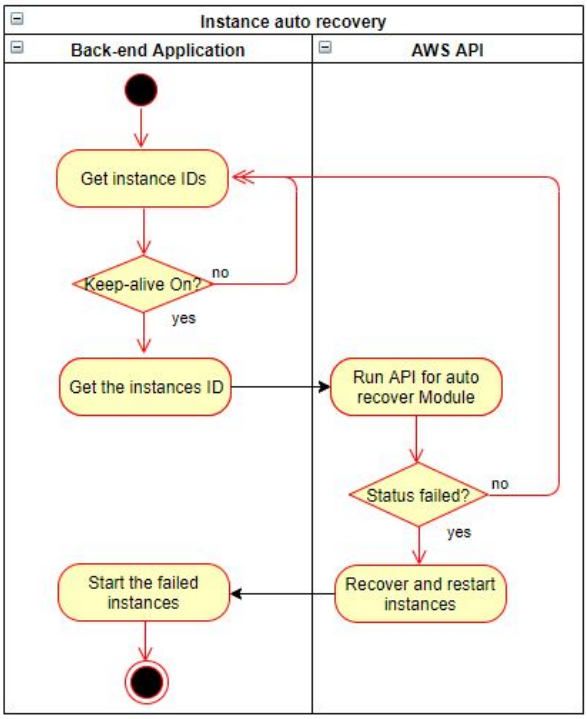

Figure 6: Instance auto-recovery activity

able of automatic failure recovery with batch instance processing for growing cloud-based infrastructure. We followed the Docker and container architecture model design for our application. Since Docker containers are capable of providing Operating System level virtualization and that is more relevant for our approach. Proposed model mainly have three containers including Nginx web server, Php FPM (Fast common gateway interface Process Manager) and MySQL Database. The framework used to implement our model is Laravel which is the most popular framework for Php. The flexibility and dynamic process of Laravel is essential for our proposed model.

4.1 Data functional architecture

The comparative framework and methods of previous research give the idea of developing a new model to mitigate AWS EC2 failure recovery using the Laravel framework. Our proposed architecture contains four stages or modules. A user interface, front-end application, back-end or server-side application, and AWS API interface. Each module requires input and produce output that forwarded to the next module. Data functional architecture for proposed Autodeployer automation tool is shown in Figure 7

The first module, a web interface or web-based application will act as a user interface. Users of companies will be provided with user id and password through users have access to our application. The web interface is based on the front-end application JavaScript, HTML (Hypertext Markup Language) and CSS (Cascading Style Sheet). Where HTML and CSS are core technology used to develop a web application. The parallel execution provides in web browsers helps users to use JavaScript for any web-based application Wenzel and Meinel (2015). Autodeployer has a user interface, the first module of Autodeployer which request users to register with their email id and registered AWS credentials.

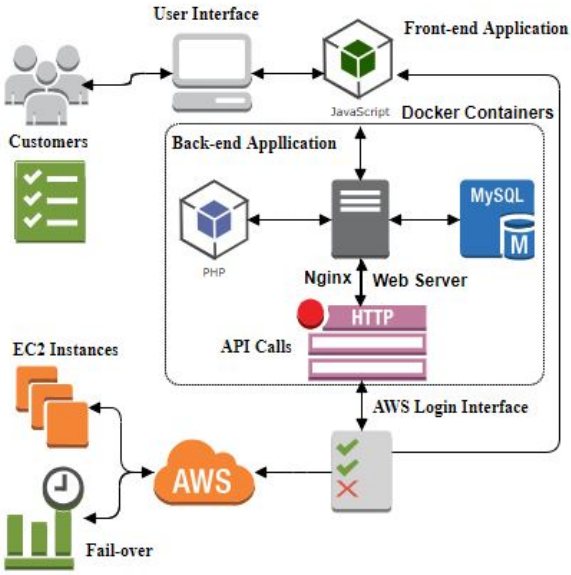

Figure 7: Autodeployer Functional Architecture

Once this has been done, users can access their account through Autodeployer.

Second module front-end application, that is JavaScript act as a client-side data processing element, that process user data with web server and database. User authentication and request are processed through the JavaScript with the help of back-end applications. JavaScript as front-end application is a general approach to function the request from client side to the server side for processing Wenzel and Meinel (2015).

Back-end or server-side application is the third working module in our automation tool. This module is the major part of Autodeployer since all the three scripts for batch instance processing and automatic recovery are running on this module. It accepts the valid request from the front-end application through the web server and verify the user with the database and finally run the appropriate script with Php FSM. This unit provides results in a response from the final module AWS API.

The fourth module is the AWS API interface which accepts requests from the back-end scripts to automate the batch process and continuously monitor the instance for failure detection. AWS provides API calls for a different process. We have API calls for instance creation, start, stop and termination.

4.2 Laravel Framework

The main analysis for the automatic failure recovery in AWS EC2 made of using the comparative and popular framework Laravel. We follow the framework Laravel, which can standardize the web development process Chen et al. (2017). Php is considered to be the most popular server-side scripting language used in developing web-based application because of its dynamic nature. The construction of the front-end page for Autodeployer using the flexible blade module that integrates with the front-end framework jQuery and Bootstrap to design our dynamic interactive interface. We follow the MVC (Model View Controller) Anif et al. (2017) process simulation in the Laravel framework to manage our architecture.

4.3 Docker and container

The proposed design of Autodeployer automation model is a more specific approach using Docker and containers. Docker is sort of computer program used to achieve O.S level virtualization also called containerization. Docker containers are used to solve the dependency issues in the software application development Abdelbaky et al. (2015). Hence using Docker approach helps to customize the interface and environment perfectly to our proposed automation tool Autodeployer. As the Dockers are open source platform Preeth et al. (2015), dependencies for developing our tool was quietly reduced. Three major container modules in our Autodeployer model are,

- Nginx Web Server Container: All request from the user to the Autodeployer is going to the Nginx Web server. Starting from the user authentication to the batch processing requests are managed by the Nginx server. The main advantage of Nginx is its immense concurrent processing capability and performance Chi et al. (2012). On the other side, Nginx is working with event-driven architecture, it can handle many HTTP connections concurrently Data et al. (2017).

- Php FSM Container: The main user request process and script are managed by this container service. All the user authentication request from the web server are pass to Php FSM and it validates data with MySQL database. All the controllers are running under this container service.

- MySQL Database Container: The third container service in the Autodeployer helps to manage user credentials. All registered user information's are stored on MySQL database and validated on request. It is a popular open source RDMS (Relational Database Management System) used to store and manage user data Fahad and Uddin (2016).

5 Implementation

A web application is developed for users to perform batch processing and mitigate from failure recovery. The final implementation follows the MVC framework and consists of components like framework, methods, configuration files and some of the AWS services. There are some software tools and languages involved in the development of Autodeployer automation tool. We followed the new technology Docker and containers to eliminate the dependencies may occur during implementation and testing of our application. Docker and containers have the ability to access and install the dependent components from the internet. The proposed Autodeployer have a user registration and login page through users can have access to AWS EC2 instance services.

5.1 Model View and Controllers (MVC) Architecture

The architecture model used for Autodeployer automation tool is MVC. It helps to model and implement a user interface for Autodeployer. This software architecture pattern helps in proper designing and promote the system modular and ease of use. MVC logic divides the application into three separate parts. The first part model that defines the data structure of the application and updates the application to added items to it. The view defines the UI (User Interface) for better user experience and shows how users can perform

their action on Autodeployer. Controller logic works based on the user request and it accepts the data from the view module and process them and notify the model module to add them.

The **welcome.blade** Php function under the resources are used to design our proposed web application. Autodeployer has a login and register tab in the application design. Users need to register with their e-mail id with a password and their AWS credential. The title, background, positions of the user login and registration tabs and positioning of the image are done at this page. Whereas the **home.blade** Php function is the extension to the welcome blade function. It helps to various status information to the users by giving some alerts.

Controllers are main functions in the Autodeployer model which holds major three controllers and the user authentication functions. The function **Controller.php** class is the base controller which authorizes user requests, dispatches jobs and validates all requests. The **Homecontroller.php** class extends the controller class and used to access the AWS API services to perform certain actions based user request. Setting user in AWS, getting instance, checking whether user request for auto-recovery, starting, stopping and terminating instance are performed under this controller class. The major controller in Autodeployer is **BulkController.php** it has all the controls for the AWS API services to start, stop, terminate with bulk instance upload and launching. It always checks the keep-alive value for each running instance in order to track their status. Auto recovery of any instance is possible only if the particular instance is active with keep-alive status on.

5.2 Framework, Methods and Configurations

As mentioned in the design section, the framework used for the implementation of Autodeployer is **Laravel**. The framework is defined under **composer.json** file. We have included AWS SDK service for Php and package service module for Php. It also specifies the dependencies for running the particular Laravel framework and the version of Php tool.

Autodeployer uses HTTP service to access the user data and perform associated task. HTTP uses some methods to access the data from users and interface. Get is an HTTP method which can be used to read the data from the input. We have home controller index for user authentication method and bulk controller index function for reading the batch data. The other HTTP methods used for bulk instance processing are bulk controller start, upload, stop, terminate, keep-alive on and keep-alive off functions. Where bulk controller upload uses HTTP method post.

All the configurations for proposed Autodeployer tool are stored on the file **.env**. It stores application name, storage location, key, the URL to access the web application, database connection for user credentials, port address, host id, user name, and password.

5.3 AWS Services

Batch instance creations and auto recovery of failed instances are performed under the AWS services. Autodeployer actions are based on the API services offered by Amazon. They have a wide range of API calls for developers to access their services on the background. The main advantage of these services is they immensely reduce the time to access a service from AWS. The major services used for our proposed research tool are,

- setUser(): After authentication, this function helps to set up the user in AWS. It set the user to a particular region and can specify the region of the instance to launch. The user credentials like AWS Key and AWS Token are the major user input to authorize the user in AWS account.

- gets Instances(): This function describes the instances from the AWS to access the instances as reserved.

- startInstance(): This API helps to start a new EC2 instance from the AWS interface. You can specify the type of instance and storage before starting an instance. Users can specify the AMI type, min and max count, instance type, availability zone, etc based on their requirements.

- stopInstance(): Instances can be put on stop state if that is not used for some time. You have to keep them on a stop in order to reduce the usage cost. Terminating an unused instance leads to inaccessible and impossible recover hence you can put them on stop state until it needed again.

- terminateInstance(): Instance which is not in use can be terminated in order to reduce the cost of running instance. Whenever the additional instances launched to meet the peak demand expires, then you can terminate the additional instances.

- start bulkInstances(): Access the input from the user as excel file or associated file to launch multiple instances at once to reduce the instance launch time for growing infrastructure applications.

- restartInstance(): This function requires whenever an instance need to restart again or the case when instance stop working due to any failure.

5.4 Tools and languages

This section discusses the main tools and language used for the implementation of proposed script automation tool Autodeployer. We have discussed modules of Autodeployer on the design section which briefs the tools incorporate. Each module uses different tools,

Front-end : The Table 3 below shows all the tools and language used for front-end side

Table 3: Web application tools.

Front-end application		
Tools	Packages	Version
Framework front-end	Bootstrap	4
IDE	Php Storm	2018.3
Container	Docker	18.09
Scripting	JavaScript	1.6
Library	jQuery	3.1

Back-end: The tools and language used for server side scripting are shown in Table 4

Software's : Software's and languages used for implementing proposed script automation tool are shown in Table 5

Table 4: Back-end application tools.

Server-end application		
Tools	Packages	Version
Framework back-end	Laravel	5.6
IDE	Php Storm	2018.3
Build	aws-sdk-php	3.72
Scripting	Php	7
Package	Php excel	2.0

Table 5: Languages and software used for implementation.

Software applications	
Tools	Version
JavaScript	1.6
CSS	2.1
HTML	5
Php FPM	7.2
MySQL	5.7
Nginx	1.14

6 Evaluation

To evaluate the performance of our proposed model, we have tested the Autodeployer automation tool with the popular AWS CloudWatch alarm. For the purpose of reducing the cost of implementation, all test have conducted on AWS t1 and t2.micro EC2 instances. To identify the time metric and to reduce the launch time we conducted various test analysis using different workloads. The same workload has been applied to Autodeployer and AWS CloudWatch to analyze the results. We have evaluated the time metric for both models by applying the same data set. The evaluation takes four experiments, one for the batch processing with the manual process, second for the automatic recovery for batch instances, recovery in various instance types and final experiment for the single instance failure. Apart from the above four experiments, we have conducted a failure recovery test on single instances of the different type in order to calculate the recovery time of each instance type. In the following section, we discuss the various results achieved on testing the instance batch processing and automatic failure recovery in AWS EC2 cloud instances.

6.1 Experiment 1: Batch processing with manual process

The first experiment was conducted on AWS was to calculate the time for a batch of different AWS EC2 instances. By considering the AWS On-Demand limit of 20 instances at a single time, we have conducted the test with a batch of 15, 25, 50 and 75 instances by terminating some of the instances and run the batch process again. From the results achieved on batch processing using our proposed model was better than the current AWS GUI interface and API approach. They are a significant time difference between manual instance launching vs processing of batch instance processing. The time comparison graph for batch AWS EC2 instance launching with manual launching is shown Figure 8

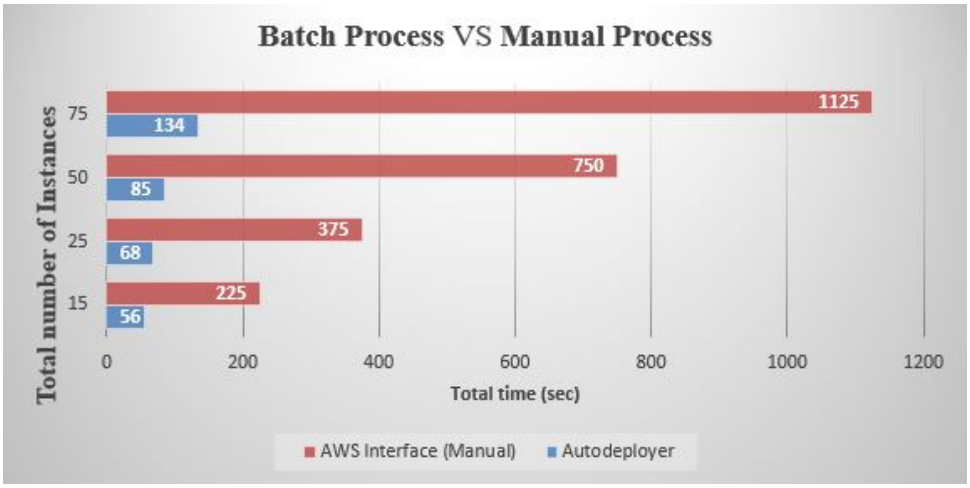

Figure 8: Batch Process VS Manual Process

To run and launch a batch of 15 instances using proposed Autodeployer took just 56 seconds on an average of 4 seconds per instance. The results were varying depending on the type of instance and Operating system running on it. For the batch of 75 instances, it took 134 seconds in Autodeployer and 1125 second in AWS GUI. We have calculated all average values only after running the test multiple times in both testing environments. Some test was showing a bit different than the previous one but the average of all the conducted test shows Autodeployer gives good performance in terms of reducing the time required to launch a batch of instances.

6.2 Experiment 2: Automatic recovery in batch instance

The major objective of this research was to analyze whether it possible to reduce the batch of AWS EC2 instance launch time and mitigate a number of failures. After conducting some test on the failure recovery on a batch of instances, we got some good results. The result was compared with currently available mitigation approach provided by Amazon, CloudWatch alarm. We applied the same type of workloads to for this test as well. For a batch of 15 mixed types of EC2 instance taken an average of 259.5 seconds to automatically recover from failures.

The test considered worst case scenario of auto recovery after 100 percent instance failure. The same scenario took 900 seconds for auto recovery for AWS CloudWatch alarm. The final result shows our proposed architecture model Autodeployer is 3 times faster in recovering a failed instance than AWS CloudWatch alarm by considering on workloads of 15, 25, 50 and 75.

We also considered the test with few failures and calculated the average time for recovery in both tools. The main advantage gained for auto recovery using Autodeployer is the status reporting time. Our proposed model checks error and report the status in every 5 seconds and CloudWatch sending a status report at a minimum of every 60seconds. Failure recovery for single instance of different instance type, the result shows t1.micro instance took 45 seconds and t2.micro captured 80 seconds on Autodeployer. The same test took 130 and 120 seconds respectively on AWS CloudWatch alarm for automatic recovery. The amount of time taken for auto-recovery using Autodeployer and Amazon CloudWatch Alarm in batch instance processing is shown Figure 9

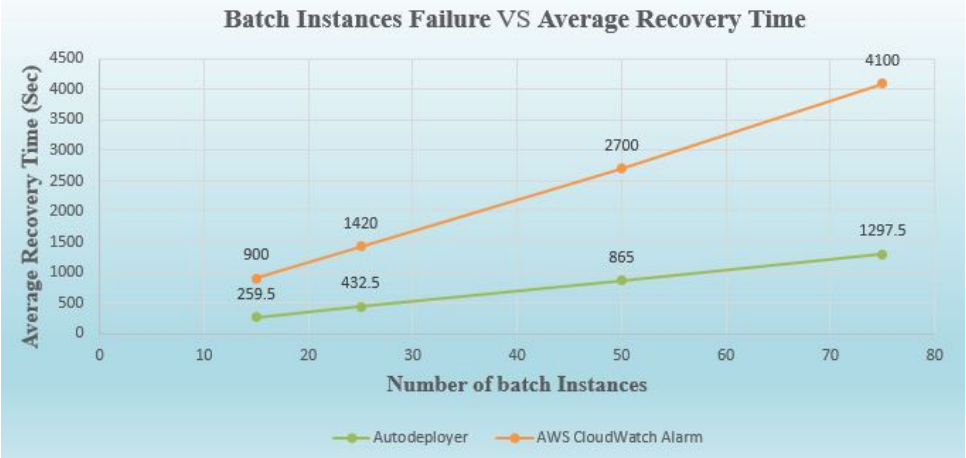

Figure 9: Batch Instances Failure VS Average Recovery Time

6.3 Experiment 3: Recovery with varying instance type

The third experiment was conducted to analyze the effect of launching time and recovery time on various instance type. We performed the test on top of free tier AWS instance t1.micro and t2.micro with different AMI or Operating System running. The results show both instance type running with Linus O.S is recovering faster than the Windows systems. This test also takes less time for recovering from failure than CloudWatch alarm. Recovery time comparison between Autodeployer and CloudWatch Alarm for varying instance and AMI type are shown in Figure 10

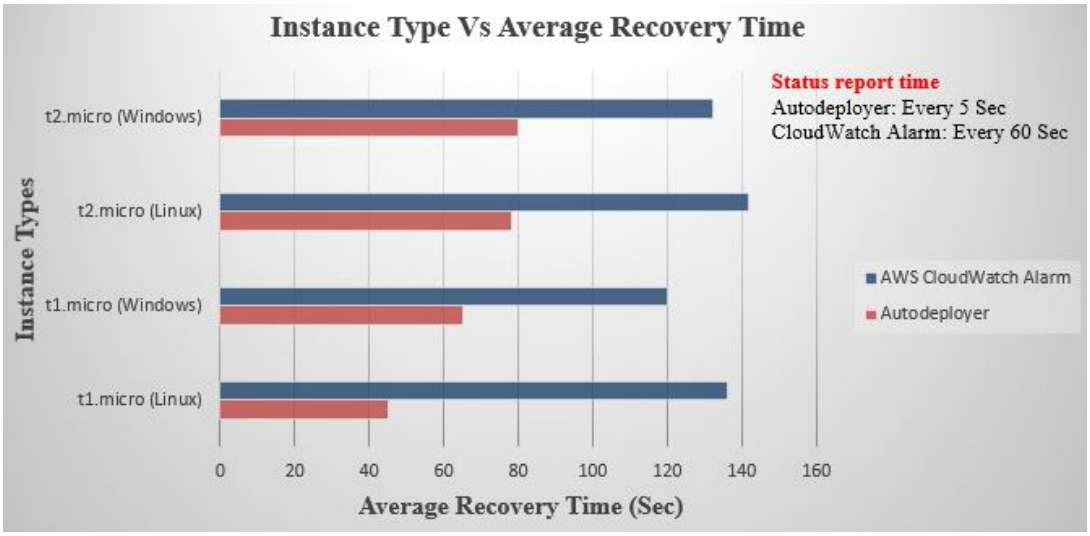

Figure 10: Instance Type Vs Average Recovery Time

6.4 Experiment 4: Recovery from single instance failure

This experiment was conducted to test the effect of time on single instance failure recovery on different instance type. The test result shows t1.micro instance takes less time for

recovery than t2.micro in Autodeployer. Which is 2 times lesser than the time taken by CloudWatch approach. The comparison graph for recovery time is shown in Figure 11

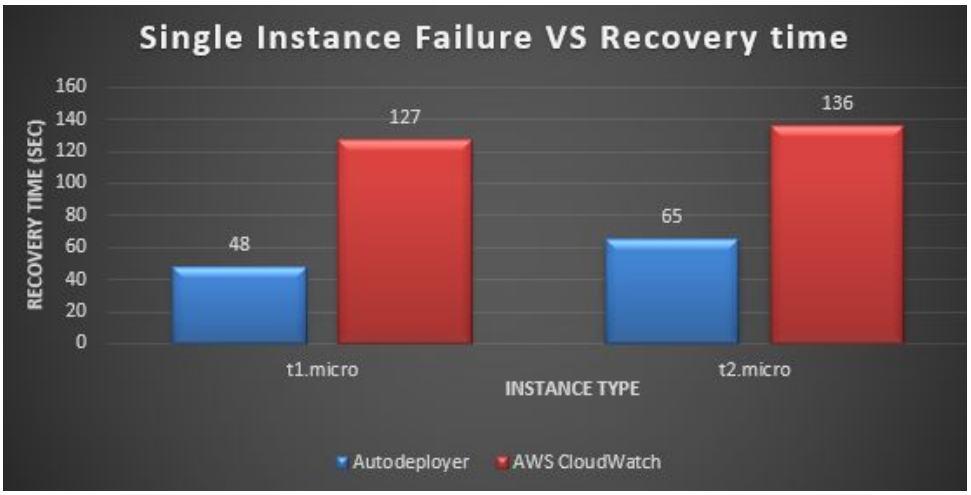

Figure 11: Single instance failure vs recovery time

6.5 Discussion

Based on the detailed analysis and test on the research work, the proposed system saves more time for batch instance process and recovery. But lacking the time needed to read the data from input files for a batch process. From the previous research on the batch processing and recovery techniques, we are able to achieve an average of 3 times better results than the available approach or model in the market today.

Many of case studies on previous works were lacking either batch processing technique or failure mitigation. Some of the approaches require more cost and infrastructure for implementation. But our approach can reduce these cost for implementation with better performance. We implemented two approaches for batch processing and automatic failure recovery for single and batch instances. This will reduce the time for creating server instances to infrastructure for any growing organization. This research also helps in limiting the time spending for recovering failed instances.

Although our web-based application Autodeployer was able to greatly reduce the time for batch AWS EC2 instance launching and recovery. It also has some limitations, our proposed approach can even make better if it can process more instance at the same time including different regions.

7 Conclusion and Future Work

This research work proposed a new script based automation model to automatic recover and gives continuous tolerance to AWS EC2 instances. As part of this research, our objective was to optimize the time needed to process the batch of EC2 instance and recovery in case of failures. After the successful implementation of the research project, we were able to reduce the time needed for automatic recovery in the business perspective. Our proposed research work was able to optimize the time by improving the faster recovery method, which is 3 times faster than the AWS CloudWatch approach.

The implemented research method will help the infrastructure engineers in any growing organization to cut down the major impact of failures particularly by using EC2 instances for their infrastructure. This approach makes business continuity for any small and medium scale companies from unexpected instance failures and loss. The lack of different input format can be eliminated by adding the newer package support system to the proposed model.

The performance and the efficiency of the system can be improved further by adding the feature to support different regions and hundreds of the On-Demand instances at the same time. Addition to this in future, an AI-based system can be added with Autodeployer to predict the system failure by learning the failure status from other instances and mitigate failures before it happens.

References

Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M. and Steinder, M. (2015). Docker containers across multiple clouds and data centers, *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*, IEEE, pp. 368–371.

Anarado, I. and Andreopoulos, Y. (2016). Core failure mitigation in integer sum-of-product computations on cloud computing systems, *IEEE Transactions on Multimedia* **18**(4): 789–801.

Anif, M., Dentha, A. and Sindung, H. (2017). Designing internship monitoring system web based with laravel framework, *2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, IEEE, pp. 112–117.

Azeem, B. A. and Helal, M. (2014). Performance evaluation of checkpoint/restart techniques: For mpi applications on amazon cloud, *Informatics and Systems (INFOS), 2014 9th International Conference on*, IEEE, pp. PDC–49.

Caraman, M. C., Moraru, S. A., Dan, S. and Grama, C. (2012). Continuous disaster tolerance in the iaas clouds, *Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on*, IEEE, pp. 1226–1232.

Chard, R., Chard, K., Ng, B., Bubendorfer, K., Rodriguez, A., Madduri, R. and Foster, I. (2016). An automated tool profiling service for the cloud, *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, IEEE, pp. 223–232.

Chen, L., Huang, W., Sui, A., Chen, D. and Sun, C. (2017). The online education platform using proxmox and novnc technology based on laravel framework, *Computer and Information Science (ICIS), 2017 IEEE/ACIS 16th International Conference on*, IEEE, pp. 487–491.

Chi, X., Liu, B., Niu, Q. and Wu, Q. (2012). Web load balance and cache optimization design based nginx under high-concurrency environment, *Digital Manufacturing and Automation (ICDMA), 2012 Third International Conference on*, IEEE, pp. 1029–1032.

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. and Warfield, A. (2008). Remus: High availability via asynchronous virtual machine replication, *Proceedings of*

the 5th USENIX Symposium on Networked Systems Design and Implementation, San Francisco, pp. 161–174.

Data, M., Luthfi, M. and Yahya, W. (2017). Optimizing single low-end lamp server using nginx reverse proxy caching, *Sustainable Information Engineering and Technology (SIET), 2017 International Conference on*, IEEE, pp. 21–23.

Datta, E. and Goyal, N. (2014). Security attack mitigation framework for the cloud, *Reliability and Maintainability Symposium (RAMS), 2014 Annual*, IEEE, pp. 1–6.

Fahad, S. M. F. and Uddin, M. S. (2016). Cloud-based solution for improvement of response time of mysql rdbms, *Computational Intelligence (IWCI), International Workshop on*, IEEE, pp. 7–10.

Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A. and Stoica, I. (2009). Above the clouds: A berkeley view of cloud computing, *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* **28**(13): 2009.

Gokhroo, M. K., Govil, M. C. and Pilli, E. S. (2017). Detecting and mitigating faults in cloud computing environment, *Computational Intelligence & Communication Technology (CICT), 2017 3rd International Conference on*, IEEE, pp. 1–9.

Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J. and Wright, N. J. (2010). Performance analysis of high performance computing applications on the amazon web services cloud, *2nd IEEE international conference on cloud computing technology and science*, IEEE, pp. 159–168.

Preeth, E., Mulerickal, F. J. P., Paul, B. and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization, *Control Communication & Computing India (ICCC), 2015 International Conference on*, IEEE, pp. 697–700.

Samak, T., Gunter, D., Goode, M., Deelman, E., Juve, G., Silva, F. and Vahi, K. (2012). Failure analysis of distributed scientific workflows executing in the cloud, *Proceedings of the 8th international conference on network and service management*, International Federation for Information Processing, pp. 46–54.

Varia, J. (2010). Migrating your existing applications to the aws cloud, *A Phase-driven Approach to Cloud Migration* .

Villamayor, J., Rexachs, D. and Luque, E. (2017). A fault tolerance manager with distributed coordinated checkpoints for automatic recovery, *High Performance Computing & Simulation (HPCS), 2017 International Conference on*, IEEE, pp. 452–459.

Wenzel, M. and Meinel, C. (2015). Parallel network data processing in client side javascript applications, *Collaboration Technologies and Systems (CTS), 2015 International Conference on*, IEEE, pp. 140–147.

Yao, M., Zhang, P., Li, Y., Hu, J., Lin, C. and Li, X. Y. (2014). Cutting your cloud computing cost for deadline-constrained batch jobs, *2014 IEEE International Conference on Web Services (ICWS)*, IEEE, pp. 337–344.

Zhang, Q., Cheng, L. and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges, *Journal of internet services and applications* **1**(1): 7–18.

National College of Ireland

Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool

MSc Research Project
Cloud Computing

Shinoj Pittandavida
Student ID: x17169704

School of Computing
National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shinoj Pittandavida
Student ID:	x17169704
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Victor Del Rosal
Submission Due Date:	20/12/2018
Project Title:	Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool
Word Count:	1415
Page Count:	10

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

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

Signature:	
Date:	28th January 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Auto-recovery and continuous disaster tolerance in Amazon Web Services instance using Autodeployer script automation tool

Shinoj Pittandavida
x17169704

1 Introduction

As part of our research work, a model of the proposed automation tool is implemented. In order to avoid the conflict of dependencies of different applications, we used Docker and Container approach. Using Docker method we were able to install and run the required software application easily. To optimize the cost of implementation, this model has developed in Linux platform. A web-based prototype model called Autodeployer was developed to demonstrate our research solution.

This research project used different software tools and languages for the development. The open-source Linux platform is used as an environment for developing, testing and deploying the Autodeployer application. The leading container platform Docker has been used as a method to set up the environment for developing an application and it reduces the application dependencies. The whole development system was followed the Linux platform and the installation and configuration are described in the following sections.

2 System Environment and Configurations

Setting up the environment for developing an application is very important. The whole application development was taken place on a system running with Ubuntu 18.04.1 LTS (Long Term Support) Operating System. The minimum recommended system requirement to run this O.S are Petersen (2018) that is given below,

- 2 GHz Dual-core processor or above
- 2 GB RAM
- 25 GB free hard disk space
- Installation media either DVD or USB

The development and testing of the web application have taken place on the system running with the Ubuntu has the following features. The system should be configured with an internet connection to update and install the dependent applications on the development and testing phase. It is mandated to run the O.S update before configuring the development environment in Ubuntu.

- 2.50 GHz Core i5 CPU
- 4 GB RAM
- 500 GB free hard disk space
- Installation media with DVD or USB

3 Software tools and languages

As part of the design, some of the software tools and languages are employed in the development and testing of the proposed research work. The tools and languages are divided into 3 models. Front-end application, Back-end or server-side scripting tools, and developing languages.

- Front-end: It is mostly used for designing the web page, look and feel of the user interface, user request handler and supporting library files. The Table 1 below shows all the tools and language used for front-end application in the development process.

Table 1: Web application tools.

Front-end application		
Tools	Packages	Version
Framework front-end	Bootstrap	4
IDE	Php Storm	2018.3
Services	AWS SDK Php	3.72
Container	Docker	18.09
Scripting	JavaScript	1.6
Library	jQuery	3.1

- Back-end: The user request processing happens in the back-end of our application. The main Php script and framework is employed in this part. The tools and language used for server-side scripting are shown in Table 2

Table 2: Server side application tools.

Back-end application		
Tools	Packages	Version
Framework back-end	Laravel	5.6
IDE	Php Storm	2018.3
Build	aws-sdk-php	3.72
Scripting	Php	7
Package	Php excel	2.0

- Software's and Languages: The purpose of the software is to perform actions based on the user requests and the language helps to design and model them. The Software's and languages with a version that take part of the implementation of proposed script automation tool are shown in Table 3

Table 3: Languages and software used for implementation.

Software applications	
Tools	Version
JavaScript	1.6
CSS	2.1
HTML	5
Php FPM	7.2
MySQL	5.7
Nginx	1.14

4 Installation of Docker and containers

Autodeployer used a specific approach to design the environment for development. The Docker and containers are easy to set up for developing an environment and that helps to reduce the overhead of installing different application Preeth et al. (2015). The installation can be triggered by using the following command in the user terminal.

sudo apt-get install docker

This command will install the docker into the system, sudo command help user to get the proper permission to execute the command as superuser. The installation of docker is shown in Figure 1

Figure 1: Installation of docker

The next step for setting the development environment is to install the docker-compose. Docker-compose is a tool used for defining and running multiple Docker containers. This helps docker containers to define an appropriate configuration into a docker-compose YAML file Klinbua and Vatanawood (2017). Then using a single command you can create and start all services that needed from your configuration. Using below command you can install the docker-compose.

sudo apt-get install docker-compose

The above command will install the docker-compose to the system and create better and efficient deployment environment for our application. The installation of docker-compose shown in Figure 2

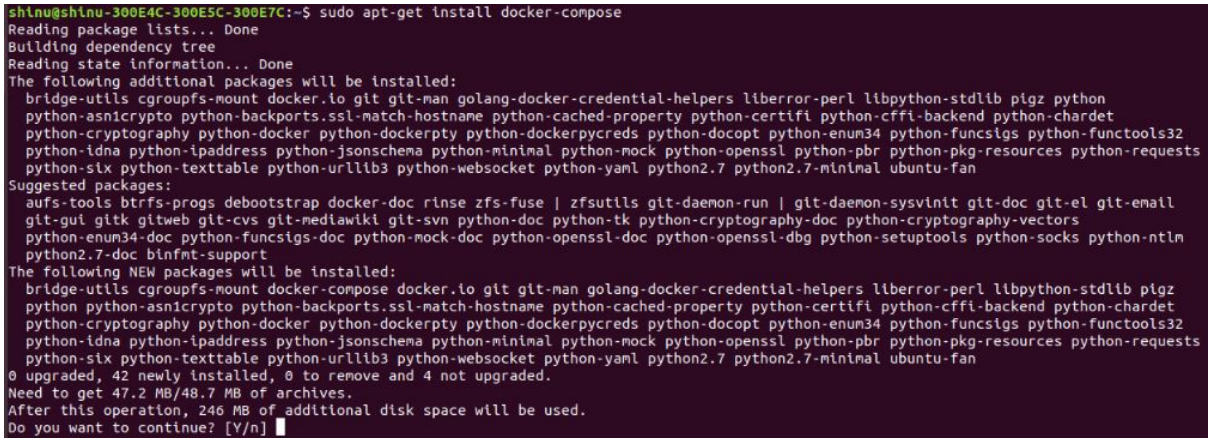

Figure 2: Installation of docker-compose

To install the all dependent components needed for our proposed model including MySQL, Nginx and Php FPM are configured on the docker-compose. The below can be used to run the docker-compose for our application. The file can be found on the installation folder after the docker-compose installation.

sudo docker-compose up -d mysql php-fpm nginx

Next step in the docker configuration is installing composer tool. The Composer which is the dependency tool used for Php application development. Installation of composer is shown in Figure 3

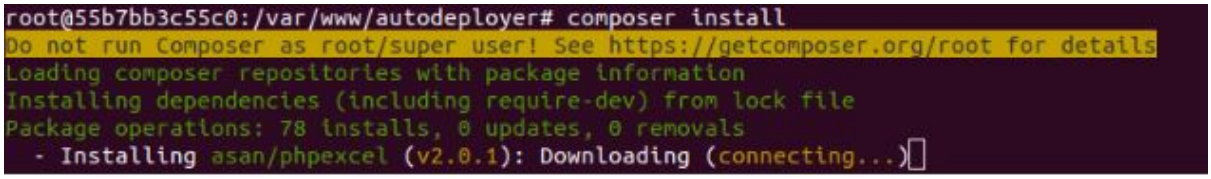

Figure 3: Installation of Php composer

5 Implementation Components

The implementation components mainly focused on the framework, design, controllers, methods and configuration files.

5.1 Laravel Framework

Laravel framework is defined in the composer.js file inside the Php FPM. The composer also shows the AWS SDK for Php and Php package used for excel. We have used the Laravel framework version 5.6 Anif et al. (2017) for our application. The configurations are shown in Figure 4

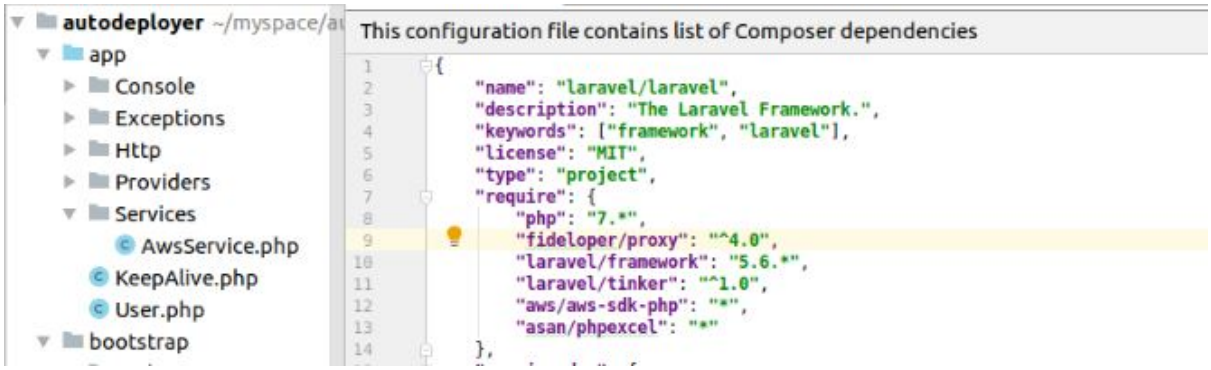

Figure 4: Laravel framework for Php

5.2 Autodeployer User Interface Design

The design of the user interface is a more important part of our model since all the process begins with the user request. Welcome.blade and Home.blade file contains the code for user interface using simple HTML. The main tabs included in the front page are login and user registration. We have used MVC architecture for the designing of the web interface. The user interface is shown in Figure 5

Figure 5: Simple html design for UI

5.3 Autodeployer Controllers

Class controller authorizes user requests, dispatch jobs and validates requests with the database. A bulk controller that accepts the input as an excel file contains instance reads and execute. The main controller class is shown in Figure 6

5.4 Methods and configuration

All the HTTP methods to access the AWS services for batch processing are written here. HTTP methods for batch processing are shown in Figure 7

Autodeployer configurations are written on the .env configuration file, that included the database connection. The configuration file are shown in Figure 8

```php
public function aliveon(string $instanceId)
{

    $this->aws->setUser();
    $ka = KeepAlive::where('instance_id', $instanceId);

    if ($ka) {
        $ka = new KeepAlive();
        $ka->instance_id = $instanceId;
        $ka->user_id = auth()->user()->id;
        $ka->save();
    }

    return redirect()->route( route: 'home')->with(['status' => "Instance $instanceId will be kept alive now"]);

}
```

Figure 6: Bulk controller class

```php
Auth::routes();

Route::get('/home', 'HomeController@index')->name('home')->middleware('auth');

Route::get('/bulk', 'BulkController@index')->name('bulk.index')->middleware('auth');

Route::get('/bulk/start', 'BulkController@start')->name('bulk.start')->middleware('auth');

Route::post('/bulk/upload', 'BulkController@upload')->name('bulk.upload')->middleware('auth');

Route::get('/bulk/stop/{instance_id}', 'BulkController@stop')->name('bulk.stop')->middleware('auth');

Route::get('/bulk/terminate/{instance_id}', 'BulkController@terminate')->name('bulk.terminate')->middleware('auth');

Route::get('/bulk/aliveon/{instance_id}', 'BulkController@aliveon')->name('bulk.aliveon')->middleware('auth');

Route::get('/bulk/aliveoff/{instance_id}', 'BulkController@aliveoff')->name('bulk.aliveoff')->middleware('auth');
```

Figure 7: Http methods for batch processing

```
APP_NAME=AutoDeployer
APP_ENV=local
APP_KEY=base64:4v1RiAXhk6pv1vVDPNTAXDHyQ9W5rkzVhUuh6yorfb8=
APP_DEBUG=true
APP_URL=http://autodeployer.test

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=default
DB_USERNAME=root
DB_PASSWORD=root
```

Figure 8: Autodeployer configuration file

5.5 AWS Services for Autodeployer

Whenever user request for a batch process service, start, stop, terminate to Autodeployer, the service routed to AWS API to run particular user request. The main function of batch instance process AWS service is shown in Figure 9

```php
public function startBulkInstances($servers)
{
    $instances = [];

    $is = [];

    foreach($servers as $server) {
        $instances = [
            'DryRun' => false,
            'ImageId' => $server[0],
            'MinCount' => 1,
            'MaxCount' => 1,
            'InstanceType' => $server[1],
            'Placement' => array(
                'AvailabilityZone' => 'eu-west-1a',
            ),
            'Monitoring' => array(
                'Enabled' => false,
            )
        ];
        $result = $this->aws->runInstances($instances);

        $is[] = $result->get('Instances')[0]['InstanceId'];
    }
```

Figure 9: AWS service for batch processing

6 Proposed Web application - Autodeployer

The first process to run the Autodeployer on the production system is by running the command ./start.sh under the Laradock directory. That directory contains all the installation of the docker and containers steps we did in the previous section. Starting the Autodeployer service on the local machine is shown in Figure 10

```
shinu@shinu-300E4C-300E5C-300E7C:~/myspace/laradock$ ./start.sh
[sudo] password for shinu:
Starting laradock_workspace_1 ...
Starting laradock_php-fpm_1 ...
Starting laradock_workspace_1
Starting laradock_mysql_1 ...
Starting laradock_php-fpm_1
Starting laradock_php-fpm_1 ... done
Starting laradock_nginx_1 ...
Starting laradock_nginx_1 ... done
root@55b7bb3c55c0:/var/www#
```

Figure 10: Starting Autodeployer automation service

As per the research implementation, we have developed a web-based application through the users or client request for AWS service. All users are needed to register themselves with their email id with AWS login credentials. Below shows the steps to register the users in Autodeployer. Web-based Autodeployer automation tool is shown in Figure 11

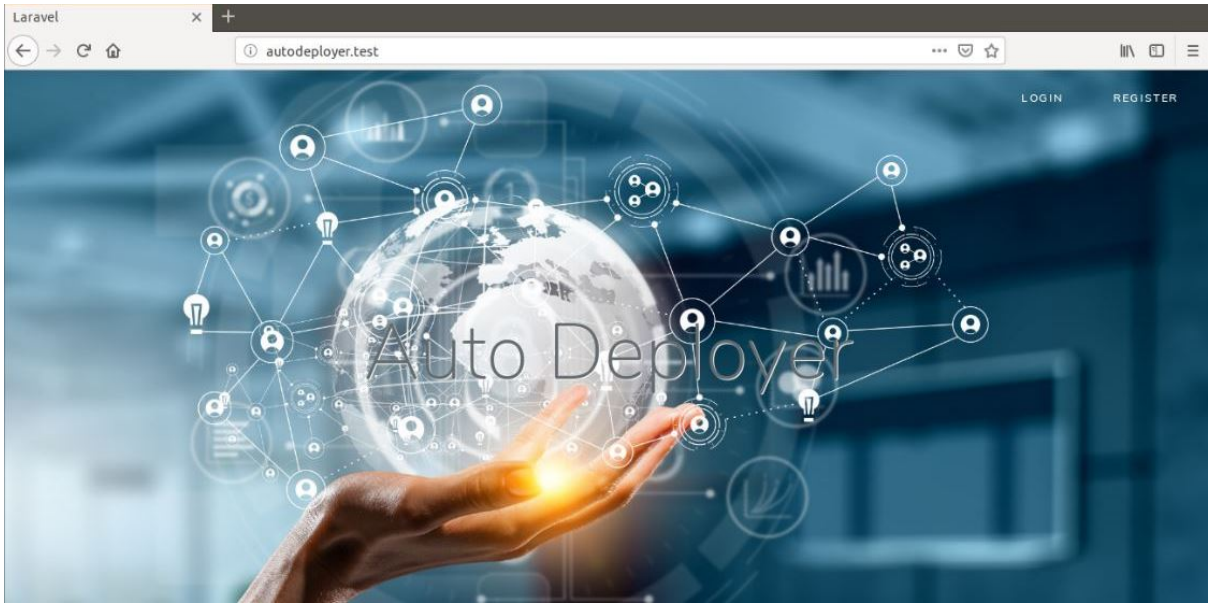

Figure 11: Proposed Autodeployer automation tool

Step 1: Register on clicking the top of the Autodeployer registration tab with user email id and their AWS key. The registration process is shown in Figure 12

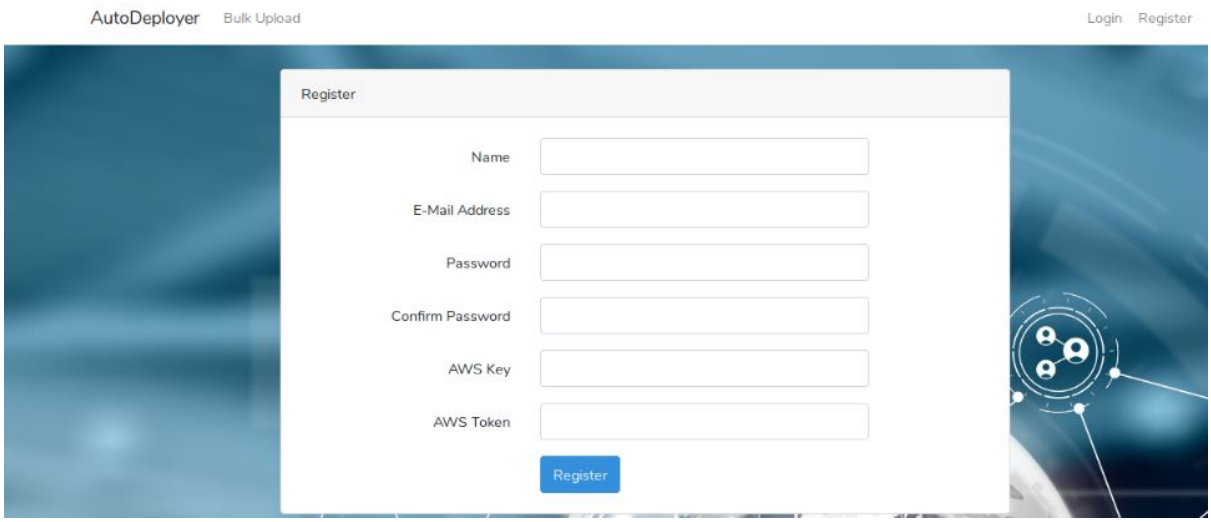

Figure 12: User registration in Autodeployer

Step 2: Select the Bulk Upload tab to select the batch process by uploading the excel file with user EC2 instances. User input upload is shown in Figure 13

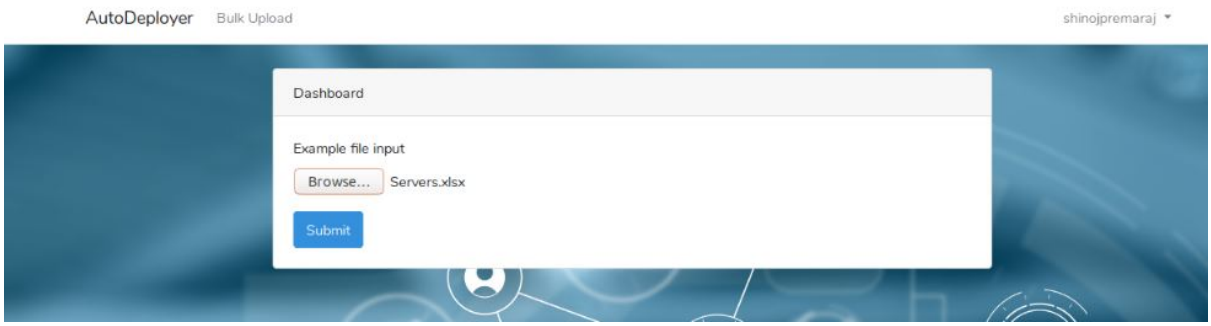

Figure 13: Batch instance upload function

Step 3: Once uploaded all the instances will be started running and timer start to indicate the instance starting time. Instance running process is shown in
Figure 14

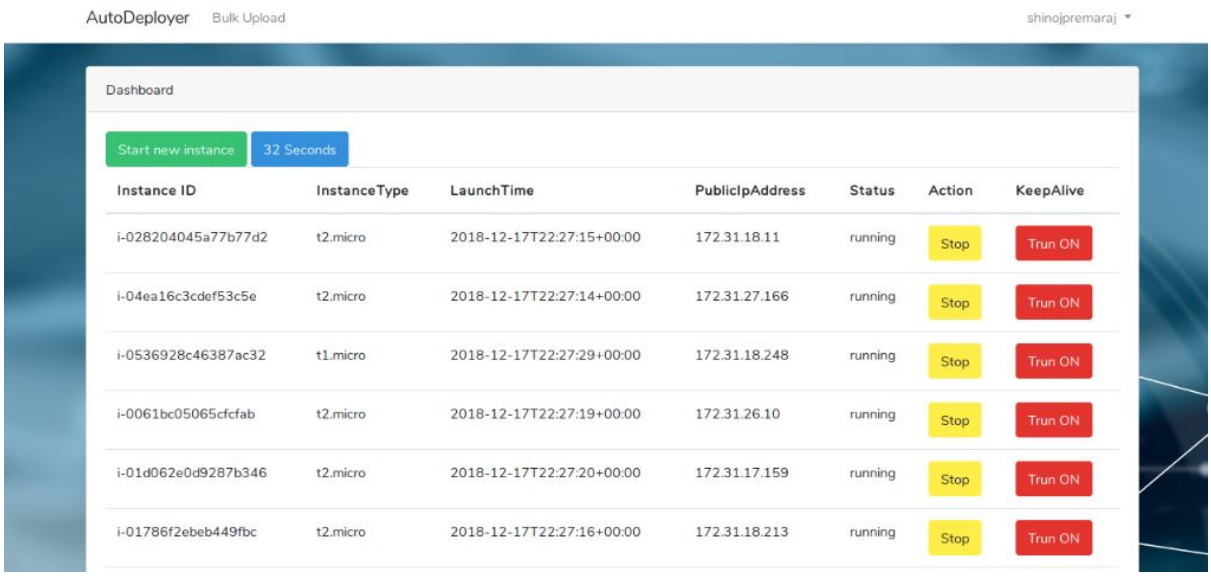

Figure 14: Batch instance running on Autodeployer

Step 4: Users or administrator of the organization needs to select the Keep alive button On for those instances to require auto recovery in case of failures. "Php artisan check: alive" function will keep on checking the status of the instance and recover from failure McCool (2012). The keep-alive module in our application perform the status checking in the following ways,

- Status check: Check the status for those instances with keep alive is ON.
- New instance check: It always checks for the new instances that created by the user and add the keep alive for new instance for failure mitigation.

Checking status with all the keep-alive value enable for instance for batch and single instance type is shown in Figure 15

Whenever the keep-alive module detects a failure in instances it automatically pushes the instance to start again from failure. Users or administrator of Autodeployer can

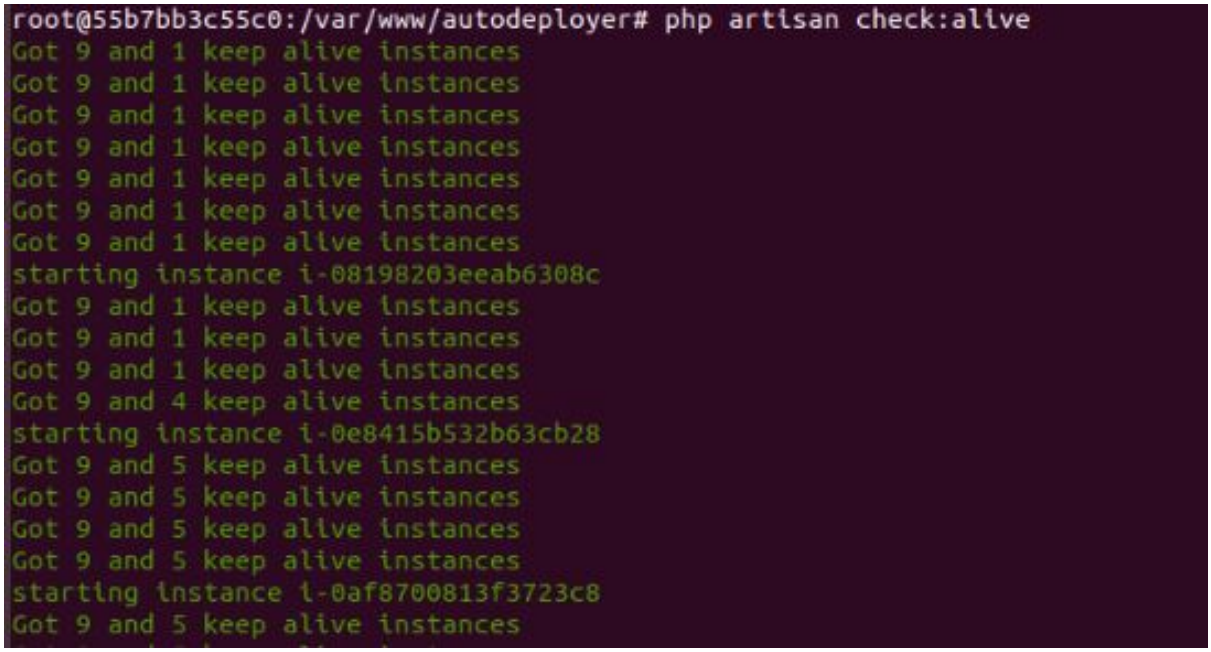

Figure 15: Autodeployer status checking for failure

view the retrieved instance changing its status from stopped to running after a successful recovery.

References

Anif, M., Dentha, A. and Sindung, H. (2017). Designing internship monitoring system web based with laravel framework, *2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, IEEE, pp. 112–117.

Klinbua, K. and Vatanawood, W. (2017). Translating tosca into docker-compose yaml file using antlr, *Software Engineering and Service Science (ICSESS), 2017 8th IEEE International Conference on*, IEEE, pp. 145–148.

McCool, S. (2012). *Laravel Starter*, Packt Publishing Ltd.

Petersen, R. (2018). *Ubuntu 18.04 LTS Desktop: Applications and Administration*, Surfing Turtle Press.

Preeth, E., Mulerickal, F. J. P., Paul, B. and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization, *Control Communication & Computing India (ICCC), 2015 International Conference on*, IEEE, pp. 697–700.