YOUNG RESEARCHER
ISSN: 2277-7911Impact Factor: 5.077 (SJIF 2021)
Peer-Reviewed, Refereed, Indexed Journal
Volume - 12, Issue - I, Jan-Feb-March 2023

Designing A Cloud-Based Software Testing Environment For Enhancing The Reliability Of Distributed Systems

Sumathi Rajkumar
Assistant Professor,
Department of Computer Science,
Thakur Ramnarayan College of Arts and Commerce, Mumbai, Maharashtra, India
Corresponding Author: ***Sumathi Rajkumar***

DOI - 10.5281/zenodo.10947539

ABSTRACT:

The reason for this work is to propose a product testing climate that we term Cloud. This climate will utilize cloud registering innovations and virtual machines that are outfitted with shortcoming infusion offices. Then again, the meaning of high constancy in a product framework has recently developed. Thorough testing of programming frameworks is getting more expensive and tedious, and in many examples, it isn't plausible to do sufficient programming testing. In case, it is frequently difficult to test equal and circulated frameworks in reality after they have been sent, regardless of the way that trustworthy frameworks, like high-accessibility servers, are instances of equal and dispersed frameworks. The cloud-based framework known as Cloud is fit for overseeing virtual machines and incorporates a shortcoming infusion capability. As per a foreordained situation, Cloud will consequently run various tests whenever it has laid out a test climate on the cloud assets by utilizing a framework setup record that has been given. We found that the Cloud framework not just simplifies it for a client to set up and test a disseminated framework on the cloud, however it likewise fundamentally eliminates how much time and cash expected for testing.

Keywords: Cloud Computing, Distributed System, Cloud Technologies

INTRODUCTION:

An elevated degree of steadfastness is an exceptionally huge viewpoint in an assortment of data frameworks in the data society that we live in today. Because of the difficulties related with keeping up with trustworthiness, the extension of the size of programming, and the significance of programming testing, it isn't possible to do satisfactory programming testing in many occurrences. In occurrence, regardless of the way that high-accessibility servers and other trustworthy frameworks are instances of equal and conveyed frameworks, it is at times testing to test an equal and dispersed framework in reality whenever it has been sent. Not exclusively should profoundly solid frameworks have some strategy for adaptation to non-critical failure, however they should likewise can endure equipment

disappointments notwithstanding programming issues. To test the framework concerning legitimate activity, which incorporates the administration of equipment disappointments, we really want to test the framework under a wide range of equipment disappointments. By and by, it isn't not difficult to harm a particular part of a genuine piece of equipment or to make an extreme tension on an actual gadget. Both of these things are challenging to do.

Likewise, it is extremely difficult to create shortcomings that are steady all through a test on a conveyed framework that is comprised of various actual server hubs, and deciding the justification behind the failure is additionally difficult. Giving the shortcoming infusion ability through the utilization of virtual machine innovation is the ongoing methodology that has been executed. A virtual machine that we have constructed, which is named FaultVM and depends on QEMU [2], is fit for mimicking the equipment disappointments of numerous gadgets at the level of the virtual machine climate. Through the use of FaultVM, the client can bring blunders into the visitor working framework while the program being tried is being run. Our Eucalyptus cloud registering innovation is being acquainted all together with work with the administration of virtual machines.

In this paper, we present a dispersed framework for a testing climate that we call Cloud. This framework offers equal programming testing conditions for trustworthy dispersed and single frameworks by utilizing virtual machines with shortcoming infusion. Using the Eucalyptus cloud registering framework [1] as a fundamental system, we have effectively conveyed Cloud . Coming up next is an outline of the commitments that every one of these papers makes:

- As an innovative use of cloud computing technology, we suggest the creation of a software testing environment for distributed systems.
- It is possible for the user to verify the functions of fault tolerance in dependable distributed system software by using a virtual machine that is equipped with a fault injection feature.
- Because we wanted to automate the process of setting up and running tests, we established the format for the description of the system configuration and the test scenario.

CLOUD PROGRAM TESTING ENVIRONMENT:

To facilitate the development of a system that is exceptionally dependable, Cloud offers a virtual machine environment that is tailored specifically for the purpose of fault

injection. Following is a list of the features that are included in Cloud .

- Using the fault injection feature that is provided in the virtual machine layer, Cloud makes it possible to test fault-tolerant functions in relation to hardware failures that take place in a real computer.
- The management of the computer resources is able to be flexible. The execution of test cases may be accomplished in a short amount of time by concurrently using the resources, provided that they are accessible.
- For the purpose of automating testing on cloud computing systems, Cloud makes use of descriptions of the system setup and the test scenario with the intention of carrying out tests.

FAULT INJECTION IN A VIRTUAL MACHINE:

To complete framework tests, Cloud utilizes a virtual machine. It is feasible to reproduce disappointments on the visitor working framework by using the virtualized equipment gadget. Also, shortcoming infusion through the utilization of virtual machines makes it conceivable to run framework tests without altering the program. It is feasible for Cloud to test programming that is working in the userland layer as well as in the part layer by utilizing the virtual machine usefulness. If programming deserts on the part layer are found during a framework test, the working framework may consequently balance up because of a piece alarm. It is hard to assemble important data for the issue fix in this situation while the framework is running on a genuine PC. This is because of the way that the client can't change the working framework when the piece alarm is happening. A blemish in the working framework that is running on a virtual machine, then again, doesn't affect the host working framework that is running on an actual PC while using a virtual machine. Subsequently, the analyzer can proceed with framework testing, and regardless of whether the visitor working framework comes up short, the analyzer is as yet ready to accumulate data to investigate. As an extra advantage, the preview of the earlier state in the visitor working framework makes it feasible for the cycle to ceaselessly return until it arrives at the ideal state.

MANAGEMENT OF COMPUTING RESOURCES:

To plan frameworks that are trustworthy, it is fundamental that framework tests be completed for however many various situations as practical. This will take into consideration the recognition and remedy of however many shortcomings as would be prudent. Likewise, to do countless tests, it is important to deal

with a significant measure of assets in a way that is both productive and adaptable. A cloud registering framework is liable for the administration of assets in the cloud. For instance, a ton of frameworks that are expected to have an elevated degree of steadfastness and dependability are comprised of a few hubs that are associated by an organization. Under these conditions, Cloud can test disseminated frameworks by using numerous visitor working frameworks.

AUTOMATING SYSTEM CONFIGURATION AND TESTING:

Cloud has the capacity to mechanize the method involved with setting up the framework that is being tried as well as the most common way of testing, which incorporates shortcoming infusion, contingent upon a situation that has been created by an analyzer. The Cloud consequently establishes reasonable test conditions and runs reasonable tests when the analyzer puts various setups of framework test conditions in a situation portrayal document. This permits Cloud to develop appropriate test conditions naturally. Along these lines, Cloud makes it conceivable to lead broad tests on the way of behaving of constancy capabilities on the framework and furthermore makes it conceivable to perform framework tests in a short measure of time.

DESIGN OF CLOUD:

As a virtualization programming, Cloud uses QEMU, and Eucalyptus, which is an open-source execution that has a similar application programming point of interaction (Programming interface) as Amazon EC2 [3]. An overall outline of Cloud is displayed in Figure 1. The cloud is comprised of the accompanying classifications of parts:

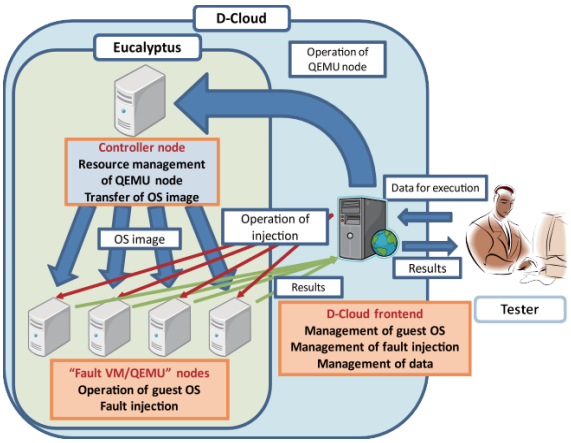

Fig.1. Configuration of Cloud

- There are QEMU nodes that are accessible using the fault injection function.
- Eucalyptus is used by the controller node, which is responsible for managing the QEMU nodes.
- A frontend hosted in the cloud that is responsible for issuing test and fault injection instructions and transferring input and output data with QEMU nodes.

FAULT INJECTION USING QEMU:

The virtualization programme that Cloud employs is known as QEMU. The following is a description of the benefits that come with utilising QEMU.

- An extensive variety of hardware devices may be emulated using QEMU. As a result, QEMU has the ability to regulate many hardware problems in the guest operating system.
- There is open-source software that can be used to access the QEMU source codes. This enables modifications to be made to the codes that are used for hardware emulation in order to include the fault injection function.
- QEMU has the capability to separate the host operating system from the guest operating system. In addition, QEMU has the potential to protect the computer host from any abnormal behaviour shown by the guest operating system while it is being tested.

Table 1: Types of Fault Injection

device	contents	value
Hard disk	Error of specified sector Specified sector is read-only Error detection by ECC Received data contains error Response of disk becomes slow	badblock readonly ecc corrupt slow
Network	1bit error of packet 2bit error of packet Error detection by CRC Packet loss NIC is not responding	1bit 2bit crc loss nic
Memory	Bit error Byte at specified address contains error	bit byte

MANAGING RESOURCES USING EUCALYPTUS:

By utilizing Eucalyptus, Cloud can handle the assets of virtual machines. In the cloud processing engineering known as Eucalyptus, PC assets are overseen in an adaptable way by means of the utilization of a virtual machine. The cloud is comprised of various QEMU hubs, which are liable for running visitor working frameworks, and a regulator hub, which is responsible for dealing with all of the visitor working frameworks.

The means that should be taken to deal with the machine's assets are as per the following.

- Operating system images are uploaded to the controller node by a tester, who then registers the machine images with the Cloud frontend software.
- At the controller node, operating system images are sent to QEMU nodes.
- On QEMU nodes, the operating system images are booted as a guest platform.

After the controller node receives the request to boot a guest OS, the controller node transfers the OS images to QEMU nodes, which are

available to run the OS images. Thus, the tester does not need to be aware of computing resources on Cloud.

CLOUD FRONTEND:

For the purpose of system testing, the cloud frontend is responsible for managing guest operating systems, configuring system test environments, and transferring different data from the tester to a guest operating system that is being run.

A cloud frontend will carry out its duty in the following manner:

- A test scenario, a test programme, input data, and an execution script are all sent to the cloud frontend by a tester via the cloud frontend.
- In the next step, the cloud frontend sends a request to the controller node to boot a guest operating system.
- Afterwards, the test programme, the input data, and the execution script are sent to the guest operating system via the cloud frontend.
- After that, the cloud frontend sends the fault injection command to the guest operating system that is the target.
- In conclusion, the Cloud frontend is responsible for gathering the output data, logs, and snapshots.

The tester is able to retrieve this data whenever they want since the Cloud frontend gathers the data that is acquired throughout the test. If the tester examines the output and tracks the process by utilising snapshots that have been stored, the tester will be able to find certain issues and study these faults in further depth.

SYSTEM CONFIGURATION AND SCENARIO DESCRIPTION:

A number of different system tests are carried out simultaneously by the tester while they are explaining the scenario. On display in Figure 2 is a comprehensive illustration of a scenario statement. A scenario statement is comprised of four components, each of which defines the test in the following manner:

- Detailed explanation of the hardware environment
- Detailed explanation of the software environment
- Failures in injection are defined as follows:
- The complete procedure for the examination

CONFIGURATION FOR THE HARDWARE ENVIRONMENT:

The "machine Definition" component contains the determination of the equipment climate that is expected to appropriately work. The things that are incorporated inside the "machineDefinition" component are recorded in Table II. Following that, the accompanying will give a portrayal of

the determination of the equipment climate. It is feasible for the "machineDefinition" component to incorporate mutiple "machine" component. There are five parts that make up the "machine" component: the name, the focal handling unit (central processor), the memory, the organization interface card (NIC), and the identifier. Every one of these parts is expected for the improvement of an equipment climate. It is the "name" part.

Table 2. Machine Definition ELEMENTS

element name	contents
machine	unit of definition of hardware environment
name	name of hardware environment
cpu	number of CPU
mem	size of memory
nic	number of NIC
id	ID of an OS image

Table 3. System Definition ELEMENT

element name	contents
system	unit of definition of software environment
name	name of sonfware environment
host	unit of definition of a testing host
hostname	name of each host
machinename	used machine element
config	used file of environment configuration

The term that is utilized to allude to an equipment climate. The setting of an equipment climate is achieved by means of the utilization of the "name" component inside the "systemDefinition" component. A memory distribution size is characterized by the "memory" component, while the "computer chip" and "nic" parts each mirror the quantity of focal handling units (central processors) and organization interface regulators (NICs), separately. A determination ID of a working framework picture that will be booted is assigned by the "id" component. Eucalyptus doles out an exceptional identifier to each working framework picture, requiring the analyzer to give a specific identifier. The analyzer can boot working framework pictures with various settings since a virtual machine might be worked with different central processors, network interface regulators (NICs), and memory sizes. Inside the setting of the circumstance displayed in Figure 2, the expression "nodeA" is conceptualized as "machineDefinition." The working framework picture known as id-1 is utilized by "NodeA," which has three organization interface regulators, 512 megabytes of memory, and one focal handling unit center.

CONFIGURATION OF THE SOFTWARE ENVIRONMENT:

The "systemDefinition" component contains the determination of the product climate that is pertinent to the application. The things that are

incorporated inside the "systemDefinition" component are recorded in Table III. various "framework" components might be incorporated inside the "systemDefinition" component, and each of these "framework" parts can contain a "name" component as well as various "have" components. What's more, a "framework" component is essential for each product climate that has been depicted. For additional data, see to Segment IV-D. The "name" component is liable for characterizing the name of a product climate. It is likewise utilized in the "testDescription" component to choose target hubs for execution of a framework test. The component known as "have" is contained three parts: the hostname, the machinename, and the setup. These parts are fundamental for each portrayal of a tried host. The "hostname" component is liable for characterizing the name of a host, and the "machinename" component is browsed the "machineDefinition" component to design the equipment climate that is depicted by the "machineDefinition" component. Using the name that is determined in the "machineDefinition" component, Cloud takes care of the arrangement of the important equipment climate. A few ecological variables might be chosen from a record by utilizing the "config" component, which is utilized to pick the document. Ahead of time, the document portrays the applications that have been introduced as well as the arrangement of the organization. The document is first transferred to the Cloud Frontend by the analyzer.

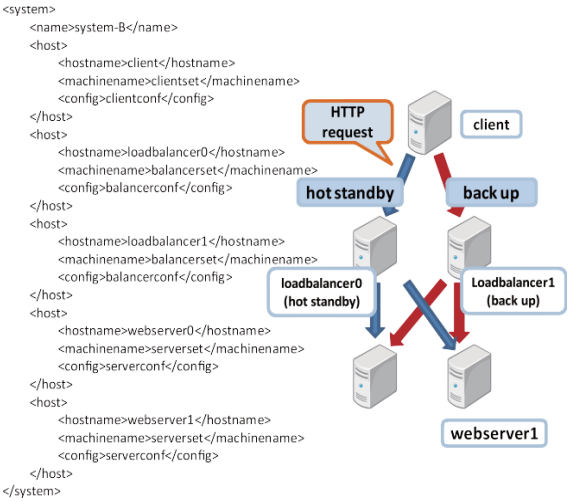

Fig. 2. Example scenario and configuration of the HA server

Utilizing the equipment climate "nodeA," the visitor working framework that is recognized as "node0" starts activity in the model XML situation is displayed in Figure 2. Various boundaries on the visitor working framework are designed as per the setup record known as "nodeconf," which is made ahead of time by the analyzer. This happens after the visitor working framework has been fired up.

An illustration of a circulated framework that contains an exceptionally accessible server framework (otherwise called a HA server framework) is displayed in Figure 3. At the point when a client sends a HTTP solicitation to the HA server framework, the solicitations are shipped off two back-end servers (webserver0 and webserver1) in a cooperative style by "loadbalancer0."

This is finished to guarantee that the solicitations are handled accurately. Besides, if "loadbalancer0" is coincidentally suspended in an odd condition, the held hub "loadbalancer1" will start load adjusting administrations as opposed to "loadbalancer0." During this trial of the appropriated framework, every arrangement data for every one of the hosts is down on paper. The amount of portrayals of host parts is then used to decide how quick visitor working frameworks fire up. Inside the setting of the delineation displayed in Figure 3, the "client" is booted as a client hub, it is liable for sending HTTP demands. It is an equipment natural component known as "clientset" and a product design record known as "clientconf" that are liable for the arrangement of the "client" hub. Then again, "loadbalancer0" and "loadbalancer1" start their tasks as loadbalancers. These loadbalancers are changed as per an equipment ecological component known as "balancerset" and a product design document known as "balancerconf." Following that, "webserver0" and "webserver1" start their tasks as web servers and are likewise arranged similarly as "client" and "loadbalancer[0-1]".

CONFIGURATION OF FAULT INJECTION:

The "injec-tion Definition" component contains the meaning of shortcoming infusion that is made accessible to the client. Coming up next is a portrayal of the meaning of the meaning of shortcoming infusion. various "infusion" components might be incorporated inside the "injectionDefinition" component, and every one of these parts has a "name" component as well as various "shortcoming" components independently. Characterizing the singular shortcoming infusions is achieved by the utilization of the "infusion" parts. The "name" component is liable for characterizing the name of a shortcoming infusion and giving the means by which the shortcoming infusion is determined in the "area," "target," "kind," and "time" are the four parts that make up the shortcoming component, which is liable for "characterizing" issue infusions. Picking an objective gadget, for example, a hard plate drive (HDD), organization, or memory, is achieved through the utilization of the "area" component. For hard circle drives (HDDs), the "target" component gives a gadget name, for example, "hda" or "hdb," or for network interface regulators (NICs), "eth0" or "eth1." A sign of the determination of shortcoming infusion components that are given in Table I is given by the "kind" component. A portrayal of the span time for shortcoming infusion is given by the "time" component. As found in Figure 2, the shortcoming infusion is meant by the letter "injectionA," and the shortcoming

infusion for bundle is signified by the letter "injectionB."

CONFIGURATION OF TEST EXECUTION:

Definite data on the execution of the test might be seen as in the "testDescrip-tion" component. The items in the "testDefinition" component are the ones that are recorded in Table V. Each of the "run," "systemname," and "end" parts, as well as various "script" components, might be incorporated inside the "testDefinition" component, which can have different components. For every one of the different definitions that are utilized in the execution of the test, the "run" component is used. If an analyzer wishes to complete two particular framework tests, the analyzer will determine the two "run" pieces in a different way. "name" is the component that determines the name of the framework test that will be completed. A result information record is unloaded, and the substance of the "name" component is utilized to decide the filename of the result document. From the name component of the "systemDefinition" component, the "systemname" component is decided to be related with the framework. "Stop" is a component that means the time at which the framework test will be finished. This "when" indicates how much time that has elapsed from the very start of the method involved with booting a visitor working framework. "on," "putFile," "executive," and "infuse" are two of the four parts that make up the "script" component. These components extend when the content is executed. At the point when an analyzer portrays at least two hubs in a "systemDefinition" component, the analyzer should then characterize script components for every one of the hubs that they have depicted.

A determination is produced using the "hostname" component inside the "systemDefinition" component to figure out which "on" component to utilize. Indicating a transferred document is achieved by means of the utilization of the "putFile" component. The information documents of the program should be available for the program to be tried. Each hub in the framework is exposed to tests that utilize the transferred record. To characterize a content that will be performed during the test, the "executive" component is a fundamental part. A previous form of the content was submitted utilizing the "putFile" order. The component known as "infuse" is browsed the "name" component that is incorporated inside the "injectionDefinition" component. If an analyzer has any desire to do blame infusion, which is determined in the "injectionDefinition" component, the analyzer should initially determine the infusion name inside the "injectionDefinition" component.

Inside the setting of the delineation displayed in Figure 2, the

test is alluded to as "testA," and the result of the test that was gained by the analyzer is likewise alluded to as "testA." Inside the testing climate known as "systemA," the visitor working framework known as "node0" utilizes "documents" as contributions for the activity of the framework. The items in the "script" component, which depicts a succession of test processes, are then used to lead the framework test as per the directions. Over the framework test, the infusion that is assigned as "injectionA" is completed 200 seconds later "node0" has been booted, and "node0" is then halted 400 seconds after the boot.

CONCLUSION:

In the ongoing review, we recommended a framework for the Cloud programming test climate that is prepared to do consequently designing test conditions, naturally executing tests, and consequently infusing flaws into equipment parts that are contained inside a virtual PC. By utilizing Eucalyptus and QEMU, we had the option to effectively make a model Cloud. An analyzer can play out a program test for a circulated framework in significant settings by utilizing Cloud. This is finished as per a situation that is ready in XML style. Moreover, the analyzer can complete various tests by embedding issues into actual parts that are contained inside a virtual PC. Various deformities might be infused into a hard circle drive (HDD), network interface card (NIC), or memory on a virtual machine (VM), and the model Cloud can independently run a framework test by using a situation record.

It is our expectation to think about the repeatability of the framework test from here on out, as well as to foster a UI that will simplify it to compose situation records by utilizing a web-based interface.

REFERENCES:

[1]. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Yous- eff, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system,” in *CCGRID ’09: Proceedings of the 2009 9th IEEE/ACM Inter- national Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131.

[2]. QEMU, open source processor emulator. [Online]. Available: http://www.qemu.org/

[3]. Amazon elastic compute cloud (Amazon EC2). [Online]. Available: http://aws.amazon.com/ec2/

[4]. A. Duarte, W. Cirne, F. Brasileiro, and P. Machado, “Gridunit: software testing on the grid,” in *ICSE ’06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006,

[5]. pp. 779–782.

[6]. N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "OurGrid an approach to easily assemble grids with equitable resource sharing," in *9th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, vol. 2862, June 2003, pp. 61–86.

[7]. M.-E. Begin, G. D.-A. Sancho, A. D. Meglio, E. Ferro, E. Ronchieri,

[8]. M. Selmi, and M. urek, "Build, configuration, integration and testing tools for large software projects: ETICS," in *Rapid Integration of Soft- ware Engineering Techniques*, ser. Lecture Notes in Computer Science, vol. 4401, September 2007, pp. 81–97.

[9]. F. J. Gonza´lez-Casta no, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, and L. Anido-Rifo´n, "Condor grid computing from mobile handheld devices," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 2, pp. 18–27, 2002.

[10]. Open solaris test farm. [Online]. Available: http://opensolaris.org/os/community/testing/testfarm

[11]. S. Han, K. Shin, and H. Rosenberg, "Doctor: an integrated software fault injection environment for distributed real-time systems," *Computer Performance and Dependability Symposium, International*, p. 0204, 1995.

[12]. S. Potyra, V. Sieh, and M. D. Cin, "Evaluating fault-tolerant system designs using faumachine," in *EFTS '07: Proceedings of the 2007 workshop on Engineering fault tolerant systems*. New York, NY, USA: ACM, 2007, p. 9.