Distributed Cloud Monitoring Using Docker as Next Generation Container Virtualization Technology

Suchit Dhakate
Dept. of Computer Engineering
Sardar Patel Institute of Technology
Andheri(W), Mumbai-400058(India)
Email: suchit_dhakate@yahoo.com

Prof. Anand Godbole
Dept. of Computer Engineering
Sardar Patel Institute of Technology
Andheri(W), Mumbai-400058(India)
Email: anand_godbole@spit.ac.in

***Abstract*—Paradigm shift is happening these days that organizations are dismissive towards capital investment on owning infrastructural setup, rather going for cloud services offered by various cloud providers. Cloud providers are accountable to their service that should always be available, reliable and scalable. They speculate lot of time and money on human resources who test the entire cloud relentlessly and make sure it is in compliance with anticipated service-level agreement (SLA). This approach is unacceptable for long term consideration keeping in mind the growing industry need and unpredictable customer loads. Hence the next immediate necessity would be to automate the process. Usually these cloud services are web based and public APIs (North-Bound APIs) are exposed for executing certain actions on Cloud Portal (CP). There can be multiple CP instances across the globe to serve regional requests. This paper proposes an architecture using containers to automate testing of entire cloud environment through North-Bound APIs (NBAPI) of respective CP instances spread across the globe and use it as a foundation for distributed cloud monitoring. A dashboard is developed to manifest a current global health status of cloud with capability of Proactive alert to forestall any undesirable state.**

***Keywords*—Cloud Monitoring, Docker, Test Automation, REST**

I. Introduction

Now-a-days every domain like E-commerce, banking, finance, telecommunication, health care, life sciences, travel, government and so on, demands servers with huge processing and storage capacity. Considering the cost of owning these resources (infrastructure) is too high. Large organizations can acquire their own infrastructure but same is not possible for start-ups, lack of capital investment being one of the potential reasons. Hence the concept virtualization evolved gradually. A platform where you dont need to buy your own physical infrastructure and maintain it. Companies like Amazon, Microsoft, VMware offers such virtualization solutions called as Cloud. Cloud providers allows you to create and manage your company network through CP with very minimum cost compared to owning your own physical infrastructure. Since there is a huge reliance on cloud providers to equip a complete reliable, available and scalable infrastructure, it should be tested and monitored thoroughly in all possible situations that may take it down from the service. In order to monitor current health status of cloud, testers tests the cloud manually and ensure the stability of cloud. There is a huge amount of capital and time investment in this process. Sometimes it may not be possible to test few scenarios manually, for e.g. spawning 1000 Virtual Machines (VM) at a time or hundreds of login request at the same time. Hence, the next step would be automating the entire manual testing. Test Automation helps to achieve critical test cases like mentioned as above and reduces human errors in the testing process. It also helps in quick discovery of bugs and fixing the same in considerably less time. To get the health status of cloud, these automated scripts has to run on each individual CP instances periodically. An environment has to be established from which these test will be triggered and executed on. Considering an option of deploying VMs to provide such environment is very time consuming and costly as it is very heavy-weight compute resource containing entire copy of guest OS, libraries and Apps. Hence we use next generation virtualization technology called Docker[10].

The paper is organized as follows: Section II literature survey, Section III background, Section IV proposed architecture for distributed cloud monitoring and smart alerts, Section V experimental results, Section VI future scope and Section VII conclusion.

II. Literature Survey

Virtualization is the core technology behind cloud infrastructure. VMs are the backbone of cloud services offered and utilized. There is a fundamental need of cloud monitoring for Cloud providers to offer uninterrupted cloud service to its tenants. Cloud monitoring is an active research area in recent years. Monitoring cloud applications with high level of correctness and less communication overhead is very crucial to meet key business priorities. A WISE[3] framework has been proposed for state monitoring in datacenter and efficiently manage cloud applications. Alerts are raised only when the system is continuously violating the speculations within a specified time window. OLA[4] is proposed in order to provide best in class monitoring service for SLA-oriented cloud services. Tenant expectations are increasing these days about SLAs to guarantee user experience rather than just

978-1-4673-6540-6/15/$31.00 ©2015 IEEE

indirect measures like speed, delay, jitter etc. OLA[4] helps providing SLA-oriented monitoring suitable for an organization. To deliver a distributed cloud monitoring solution, numerous identical VMs has to spun up all over the geo-locations. Tests has to run from these VMs and results are collected back for further data analysis. VM spin up process includes installing guest OS, installing libraries and then on top of it apps and then finally automated scripts. As a result, it uses lot of resources(CPU,Memory,Storage), it is time consuming and ultimately a costly alternative. To overcome these drawbacks we use next generation virtualization platform called Docker[6][10].

Docker is a new-edge container virtualization technology. It is a light-weight virtual machine. For an ordinary VM which is running on hypervisor(Type-1,Type-2), not only it contains applications and necessary libraries and binaries but also an entire copy of guest OS. This makes VM to be of several GBs. On the contrary, Docker eliminates hypervisor and guest OS layer, even the libraries can be shared by different applications running on top of it(Figure 1[8]). This makes Docker a very lightweight VM and easy to deploy in less time(Figure 2[9]).

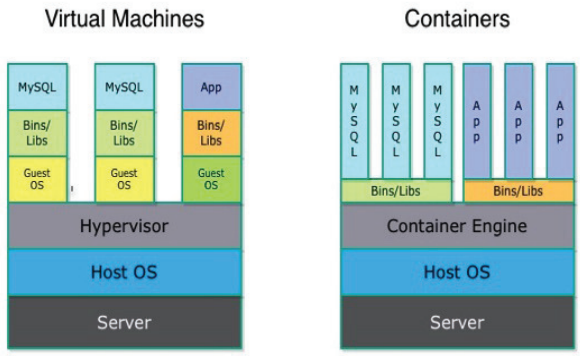

Fig. 1. VM vs. Container

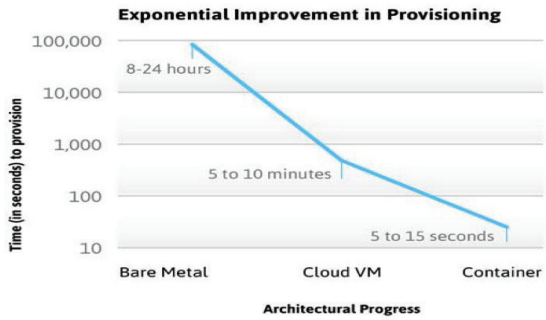

Fig. 2. Performance of VM vs. Container

Basically, it creates portable packages of applications and its dependencies. The packages are called container images and it can be published to a container repository called Docker Hub, so that it can be spawned and executed on any other machine having Docker engine. Docker uses union file system[5][6] where layers of file system are built up. Only the specific changes added to parent image are saved. Hence, size of image is very less(few KBs to MBs). As a result, publishing an image becomes very light weight and takes only few seconds. Also, spawning this image and getting the exact replica of VM takes only few seconds whereas ordinary VM cloning takes up to few minutes(Figure 2[9]). Docker can be used for continuous integration and continuous delivery(CI/CD). Developers can create their own replica of production environments and trigger actions by just spawning multiple containers in matter of seconds. A service framework[7] is proposed for parallel test execution on developers local development environment. We will be considering this for our distributed test case execution to achieve distributed cloud monitoring.

III. Background

In order to automate entire testing process of cloud services, there should be a precise way of implementation to make it more stable and adoptable over the span of time to underlying infrastructural changes. A standard framework[1][2] has to be defined and followed throughout the development process to develop a test library of core functionalities and use them as per test cases required. CP exposes public APIs called North-Bound API (NBAPI) and those can be utilized using any HTTP POST/GET tools. NBPAI defines a well structured HTTP Header and Body format that includes REST API, authentication and parameters. Executing this will in turn carry out the specified action. For e.g. Creating VM in specified datacenter, deleting datacenter, configuring firewall rules on datacenter etc. The library is developed by implementing all such REST actions and then referenced it in actual test cases(Figure 3).

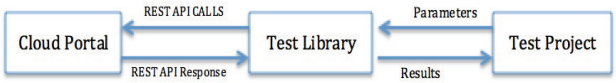

Fig. 3. Test Automation

Test Library will provide an advantage that if at any instance of time the underlying API calls got changed, then we just need to update Test Library and not at all the places in all test projects where API got called. Hence all the projects can run seamlessly.

IV. Proposed Architecture

CP gives an interface to manage cloud offerings to customers. Throughout the globe there can be many such locations(CP Nodes). In order to make sure all portals are running and meeting SLAs, there is a need of rigorous test suites(bunch of test cases that achieves certain usecases) execution periodically on all the CP nodes.

Using centralized monitoring approach, the test suites can be triggered remotely and results can be captured accordingly. In this approach there is a centralized system that has entire

automated test scripts and libraries. The drawback of this approach is that test suites are executing remotely, hence the returned performance results may not be accurate because of some network delay that might have introduced in between the data transfers. To avoid this we propose a distributed approach where the test suites will be executed locally on those target locations and then results are sent back to source system where further data analytics can be done.

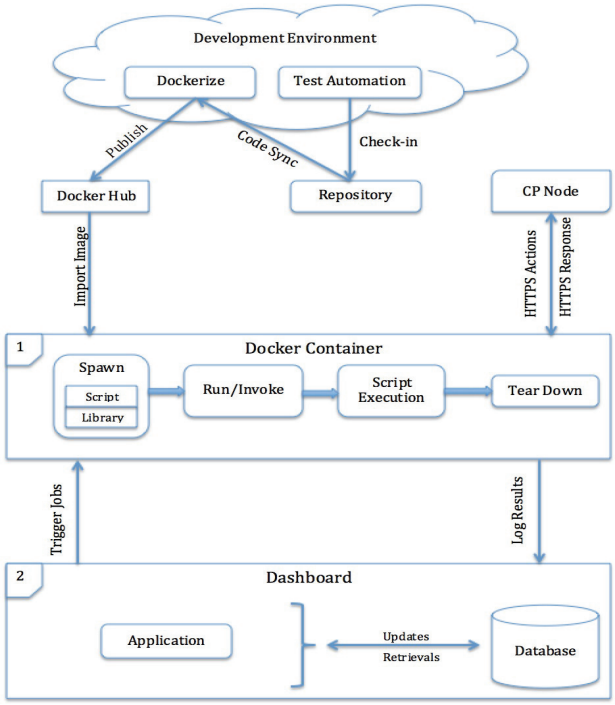

Fig. 4. Cloud Monitoring Dashboard Architecture

Consider the Cloud Monitoring Dashboard Architecture(Figure 4), automated test scripts and libraries(Figure 3) are developed in development environment, the code is checked in to some version control tool like svn[11], Perforce[13] or Git[12]. Latest codebase of every test suite and its artifacts are Dockerized and published in Docker Hub[5]. Dashboard consist of a scheduler through which test suites are periodically triggered on several Docker Containers located remotely. When test suite is triggered, a script is executed on target Docker that imports respective test suite container image from Docker Hub and spawns it. It executes the automated script inside the newly spawned container and simultaneously triggers HTTPS actions on respective CP nodes. Once the HTTPS actions are triggered, the respective CP nodes starts executing those actions such as creating VMs, Deleting a Network, Adding Firewall Rule to a Network etc. and returns the HTTPS response. During the course of test script execution, every test case level logs are captured and sent back to dashboard using TestNG[14] annotations like @BeforeSuite, @AfterSuite, @BeforeTest, @After Test and so on. These logs contains test case execution details such as test result,total execution time, failure causes if any, warnings etc. Once the script finishes its execution, the container tears down itself hence resource is utilized optimistically.

Dockers are deployed distributedly in such a way that it gives platform to spawn containers having test scripts and its dependencies to execute HTTPS actions on repective CP node. Consider Figure 5, 1-C1 indicates Docker Container which executes test suites locally on CP Node 1 by spawning containers from appropriate Docker Hub Images. For e.g. Image 1 is Docker image for test suite 1 and all its artifacts and so on. Similarly 1-C2 will indicate Docker Container which executes test suites locally on CP Node 2 and so on. There will be only one central 2-C i.e. Cloud Monitoring Dashboard Container that will periodically trigger test suites remotely and monitor all CP Nodes using distributed Dockers 1-C1,1-C2,1-C3. . . 1-Cn (Figure 6). Since dashboard is getting current health status of cloud by periodically executing test suites on all the CP Nodes across the global locations, it becomes the central platform for all testing, reporting, monitoring and alerting.

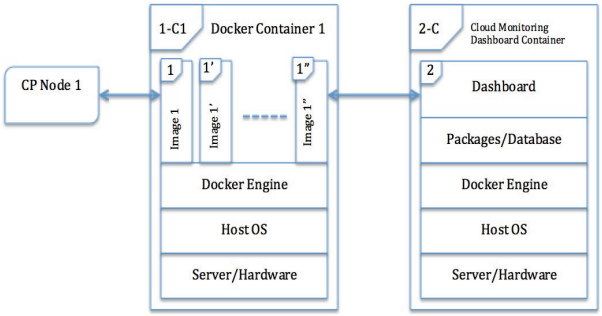

Fig. 5. Container stack. Tag 1 and Tag 2 indicates same architectural setup as mentioned in Figure 4. 1' and 1" are similar containers spawned as 1.

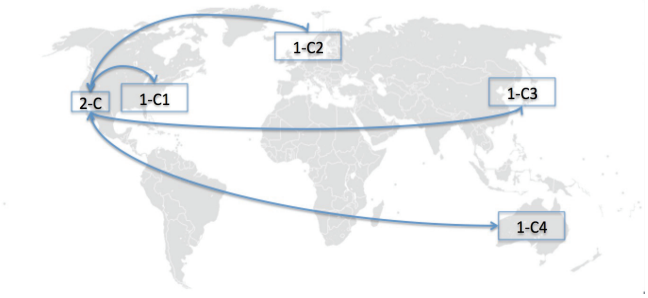

Fig. 6. Distributed Cloud Monitoring. The blocks displayed in graph are in reference to Figure 5.

Smart Alerts:
When any test case is taking more time than expected i.e. it is violating SLAs or it is failing continuously, then respective service owners are notified via email or ticketing system so that operations team can resolve the issue. This is called reactive alerting. Dashboard also alerts Proactively i.e. when

test case results matches some series of events or patterns in prescribed time frame[3] that may bring system to a state of failure or any undesired state as per the norms defined, then alert is raised to avoid such possible failures.

V. Experimental Results

Docker image is created from a Dockerfile. Dockerfile is a text file having set of commands to install test scripts and its dependencies in that Docker image. Command to create Docker image from Dockerfile present in current directory is:

```
$docker build -t newdockerimage .
```

Once the image is created, it is pushed to Docker Hub using command:

```
$docker push newdockerimage
```

The dashboard periodically triggers remote commands on target Docker to spawn test suite image and execute it on its local CP. The image is installed from Docker Hub on local Docker using:

```
$docker pull newdockerimage
```

After the newdockerimage container is spawned at target Docker, the JAVA command to start Test Suite execution is triggered inside it. The command is:

```
$docker run newdockerimage bash c "java
-classpath /Users/cloudmonitoring/Desktop/
test_suites_automation-jar-with-dependencies.jar
-Dlogin.username=cloudUser1
-Dlogin.password=cloudUser1#
-Dlogin.host=country-1-location-1.cloudspace.com
org.testng.TestNG /Users/cloudmonitoring/Desktop/
paas/testng.xml"
```

The Java command as specified above is executed inside newdockerimage container. Login.host specifies the target CP and testing.xml(Figure 7) file specifies which actions to be carried out on that CP and respective method to execute it. The newdockerimage container tears down itself once the

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="VMOperationsAutomatedScript">
 <test name="createVM">
  <classes>
   <class name="automation.vmoperations.createVM ">
    <methods>
     <include name="createVMTestMethod"/>
    </methods>
   </class> <!-- automation.vmoperations.createVM -->
  </classes>
 </test> <!-- createVM -->
</suite> <!-- VMOperationsAutomatedScript -->
```

Fig. 7. testng.xml

test suite finishes its execution. Likewise, different test cases are executed on all CPs. The information related to test suite execution such as location name, CP URL etc. is sent back to Cloud Monitoring Dashboard followed by it's test cases results using REST POST call(Figure 8). All the results received at Dashboard are logged into its database(Figure 9).

With the use of received test case results data, Graphs are plotted and cloud is monitored to measure the performance

```
HttpClient client = new DefaultHttpClient();
HttpPost post = new HttpPost('http://cloudmonitoring.cloudspace.com/api/test_results');
List nameValuePairs = new ArrayList();
nameValuePairs.add(new BasicNameValuePair('test_case_id', 24));
nameValuePairs.add(new BasicNameValuePair('test_run_id', 18));
nameValuePairs.add(new BasicNameValuePair('status', 'PASS'));
nameValuePairs.add(new BasicNameValuePair('start_time', '2015-09-24 08:56:09.541'));
nameValuePairs.add(new BasicNameValuePair('end_time', '2015-09-24 08:56:18.844'));
nameValuePairs.add(new BasicNameValuePair('log_string', 'TestCase Succeeded'));
post.setEntity(new UrlEncodedFormEntity(nameValuePairs));
HttpResponse response = client.execute(post);
```

Fig. 8. Sample POST Request

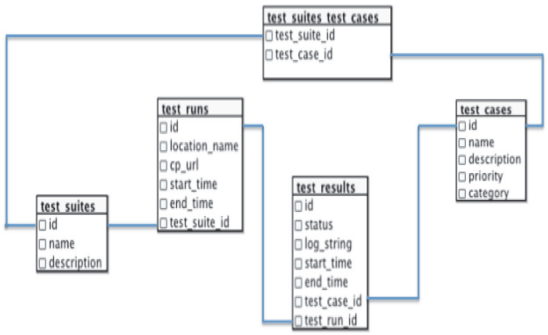

Fig. 9. Dashboard Database Schema

comparison of particular test cases across the different CP Nodes and to get the overall health status of cloud by knowing the failures if any. Figure 10 and Figure 11 are two sample test cases for which graphs are plotted.

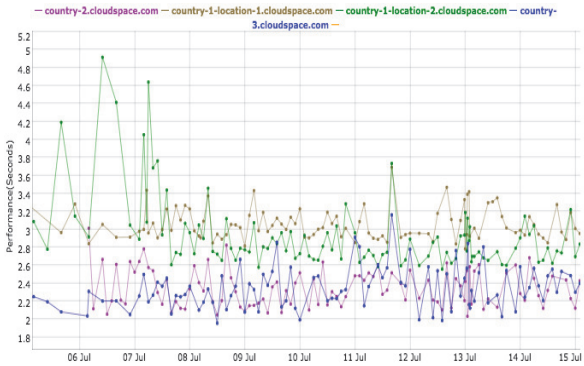

Fig. 10. Distributed Cloud Monitoring::Sample test case 1::createVM

VI. Future Scope

In addition to proactive alerting of dashboard, the capability of monitoring system can further be extended to self healing technology. With this capability , some known failures in cloud can be self repaired by the system automatically based on some artificial intelligence and data analytics on well learned Knowledge Base(KB). The KB is generated over the period of time to make system learn about possible failure causes and

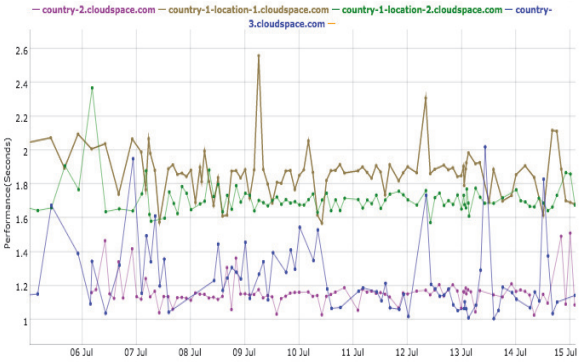

Fig. 11. Distributed Cloud Monitoring::Sample test case 2::tearDownVM

apply most suitable solutions to it by itself. After the potential solution is applied, we test the system again and if it still failed then rollback to the previous failure point and apply the next most suitable solution and so on. Initially KB won't know any failures but as they starts encountering in cloud space, the cause and solution for the failures are recorded in KB and hence KB is evolved over the period of time. If similar/same type of failures occurred in the future point of time, the system will self repair itself with the help of well learned KB.

VII. Conclusion

In order to survive/grow in the market, cloud providers have to test and monitor their cloud continuously to avoid possible failures and achieve expected or even better QoS(Quality of Service). Testing a cloud manually is very cumbersome and time-consuming process and so is the cloud monitoring. But it ensures scalability, reliability, availability and compliance with SLAs of the cloud. It is equally important to cloud providers and their customers that cloud services are up and running 24/7. This paper demonstrated the architecture for entire cloud testing, reporting, monitoring and alerting seamlessly in distributed cloud space. Containers are used for the implementation and considerable amount of resources, cost and time is saved. Cloud Monitoring Dashboard helps viewing entire cloud health status with any failures and alerts as per configuration if any. It saves tremendous amount of time, money and human resources. Proactive alerting helps to avoid possible failures in future point of time. This results in very less failures and helps to make cloud more stable, scalable, reliable, and available. It eventually improves overall uptime of cloud and ultimately attract more and more customers and helps organization to grow over the period of time rapidly.

References

[1] Fei Wang & Wencai Du, *"A Test Automation Framework Based on WEB"*, *2012 IEEE/ACIS 11th International Conference on Computer and Information Science (ICIS)*

[2] Vera Stoyanova, Dessislava Petrova-Antonova & Sylvia Ilieva, *"Automation of Test Case Generation and Execution for Testing Web Service Orchestrations"*, *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*

[3] Shicong Meng, Ling Liu & Ting Wang, *"State Monitoring in Cloud Datacenters"*, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 9, SEPTEMBER 2011*

[4] David Roxburgh, Daniel Spaven & Craig Gallen, *"Monitoring as an SLA-oriented consumable service for SaaS Assurance:A Prototype"*, *12th IFIP/IEEE IM 2011: Application Session*

[5] Di Liu & Libin Zhao, *"The research and implementation of cloud computing platform based on Docker"*, *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference*

[6] Charles Anderson, *"Docker: Software Engineering"*, *Software IEEE, VOL. 32, NO. 3, 2015*

[7] Mazedur Rahman, Zehua Chen & Jerry Gao, *"A Service Framework for Parallel Test Execution on a Developer's Local Development Workstation"*, *2015 IEEE Symposium on Service-Oriented System Engineering (SOSE)*

[8] http://patg.net/containers,virtualization,docker/2014/06/05/docker-intro/

[9] http://www.linuxjournal.com/content/containers%E2%80%94not-virtual-machines%E2%80%94are-future-cloud?page=0,1

[10] https://www.docker.com/

[11] https://subversion.apache.org/

[12] https://git-scm.com/

[13] http://www.perforce.com/

[14] http://testng.org/doc/index.html