Second National Conference on Internet of Things : Solution for Societal Needs
In association with International Journal of Scientific Research in Computer Science, Engineering and Information Technology | ISSN : 2456-3307 (www.ijsrcseit.com)

Using Azure Kubernetes Services to Deploy a Micro Service Based Application

Dr. L. Srinivasan[1], Junaid Ahmed Nadaf[2]
[1]Associate Professor, ISE Department, New Horizon College of Engineering, Bengaluru, Karnataka, India
[2]M. Tech. Scholar, Cyber Forensics and Information Security, ISE Department, New Horizon College of Engineering, Bengaluru, Karnataka, India

ABSTRACT

Micro services are an Independent Services which works perfectly on the architectural approach. Micro services helps in building the applications with its independent behaviour which is well known and also known as small services. This also helps in deploying and managing a service in the form of tiny services where each service is coded separately. The Micro service Architecture is designed in such a way that, if one component fails it won't effect on other running services and ensure that the application services are hosting normally. This Micro service architecture is designed in such a way that the components are loosely coupled and the API's are in well-defined contracts. These benefits in reducing the stress-overhead in the Business Markets. This Micro service feature is offered in one of the open source domain called Kubernetes. This Kubernetes along with Containers and pods helps in deploying, managing and scaling applications called containerized applications. Kubernetes also helps in providing support during any service failure and helps in recovering by improving the situation. In this paper we are going to experiment the Micro services performance with the help of Kubernetes along in its default configuration by enabling High Availability (HA). We had intentionally stopped many services which has helped us to get to know and understand the working mechanism of the same.

Index Terms—Micro services, Containers; Pods, Kubernetes, Failure

I. INTRODUCTION

The Micro services support polyglot programming. And these services communicate with each other by using well- defined APIs. These services are implemented in such a way that they are hidden to each other. The Micro service architecture has management (orchestration) component, which helps in identifying failures, recovering the services across the nodes and deploying the nodes as well. And the API Gateway allows clients to enter inside instead of calling the services, this client when requests for an service, the API redirects to back end and serves the client's request.

The Benefits of Micro services are agility, small teams, and focused teams, small code camp, cross platform technologies, isolating the faults, scalability and data isolation. Some of the Identify applicable funding agency here. If none, delete this.

Major challenges of Micro services are complexity, develop- ment and testing, lack of governance, network congestion and latency, data integrity, management, versioning, and skill set.

Micro services addresses the monolithic approach, which is a drawback. Micro services helps in increase the standard and speed. Micro services always stay low, so that they can easily restart, recover whenever there is service failure. Micro services are known for

reducing the Application downtime value because the services are loosely attached and they don't communicate with each other at all. In order to avail these benefits, there should be technology configured with the existing architecture style.

Containers are known for airy weight and they turn on very quickly than those of Virtual Machines(VMs). Whenever there is a Service Failure, upgrading or in failure cases the containerization of micro services would do its job. Here, we have used Docker, which is mind-blowing container platform. There would be requirement of orchestration platform as well, which helps in deployment of the containers. Kubernetes is an open source technology platform. Apart from healing and restarting the failed containers, Kubernetes has been designed in an automated manner, which identifies the failed states and recovers them automatically. Hence, this feature helps in improvement of application's availability.

The Micro service Architecture style is generic and the practitioners and investigators refers this style as a Model. And with the help of this Architecture style the investigators comes out with different perspectives. After assessment the effectiveness of the High Availability mechanism with the help of Kubernetes, the result logs have been captured. And unfortunately, the results didn't meet the requirement. And the HA effectiveness has been investigated in the Kubernetes in its default configuration mode.

The remaining portion in this paper has been explained in the organized manner as follows. Section II helps us to familiarize Architecture about deploying containerized applications with Kubernetes. Section III explains the experiments which are present. Section IV helps in analysis of the results. And the Conclusion parts have been commented on Section V.

II. IN A PRIVATE CLOUD ENVIRONMENT THE CONTAINERIZED APPLICATION IS DEPLOYED WITH KUBERNETES CLUSTER

It is much needed to know the difference between the Private Cloud and Public Cloud Environment. Kubernetes cluster should be implemented in a public cloud as it requires less effort. Whereas it will take more effort when deploying in Private Cloud. Here the main difference is exposing the Application in a better way externally. We will get to know in details about exposing the applications which is deployed in Kubernetes cluster running in a private cloud. Kubernetes also has the ability to run the Virtual Machines and helps in creating a Single Cluster view. And here the application has been composed of single micro service.

Kubernetes Ingress is a kind of resource which is designed in a structured way to showcase the services. Fig. 1 represents a architecture which is generic design of Ingress which ex- poses the service in a cluster running in a private cloud. A Cluster IP gets generated and helps in redirection of requests to the pods and this Ingress resource is used completely in back end. The Ingress controller always helps in redirecting the incoming requests to the ingress resource, and maps to the appropriate back end service. Only One Pod is used here and Ingress Controller have been deployed here. The rules have to be defined prior on the deployment controller and ensure that the ingress controller has been scheduled on a specific node. The Kubernetes has the possibility to connect to a pod directly to a port on the node which is hosting the pod. We know that we have defined a rule where Ingress controller pod has to be on the same node, and it is safe method where it can always receive the requests from outside of the cluster and that cluster has public IP of that node. Deploying the Ingress Controller and Kubernetes Cluster which is running in the Private Cloud is little tricky and there is no proper documentation available

yet. Understanding and deploying a Ingress Controller needs a lot of trial and errors to understand the working mechanism of the same. The previously described architecture is designed based on our own understanding.

III. EVALUATION RESULTS OF THE MICRO SERVICE BASED APPLICATIONS WHICH IS DEPLOYED WITH KUBERNETES

The nonfunctional requirement which is nothing but Avail- ability is measured usually when there is power outage time from the given period of time. And the Value of High Availability should be calculated when there is a up-time of 99.999 per cent, and we also need to ensure that there should be no power issue more than 5 minutes overall per year. The improvement of the availability of micro services and containers helps in Kubernetes healing capability because they are small and lightweight by natural. So here we are going to describe the experiments which we have conducted to evaluate from an availability perspective and the deployment of the micro service based application in a Kubernetes cluster which is running in a private cloud (Please refer Fig.1).

Whenever there is a Pod Failure, Kubernetes delivers the new pod automatically and therefore this ensures that the improvement of availability of the service is provided by the pod. The evaluation of Kubernetes service failure or pods failure helps in identifying the administrative operations (e.g. pod deletion, node deletion) with the help of Kubernetes command line interface (CLI) and also evaluates how fast Kubernetes delivers the new pods when there is a pod failure. And when there is more usage of Kubernetes administrative operations, where ”failure” is only not to be screened by Kubernetes. Instead, the Kubernetes administrative operations performs it tasks in a graceful manner. Therefore, there won't be any common execution failure scenarios, because there will be a perfect steps followed and these happen spontaneously which is the outcome of Events such as External failure events (e.g. physical node crash). Evaluating the conclusions based on the administrative operations is not accurate. And this is important to know that how to identify and measure the external execution failure and also measuring the availability of the above simulation results before making any decisions.

A. Measuring Availability

From the Availability Perspective, Measuring the Availability helps in evaluating the Kubernetes and some have been defined below.

a) Reaction Time: When there is a failure event detected the time calculated between the introduced failure event and the first reaction of Kubernetes.
b) Repair Time: The time calculated between the repair of the failed pod and the first reaction of Kubernetes.
c) Recovery Time: The time calculated between when the service comes online and the first reaction of Kubernetes.
d) Outage Time / No Power Issue: The time duration in which the service was offline. And hence this Outage Time represents the sum of reaction time and recovery time.

B. The Experiments

The Architecture Diagram can be referred in Fig.1. Here we have deployed three VMs in our Cluster. And this same is hosted in Open Stack Cloud. We have used Open Source OS i.e Ubuntu 16.04 and all the VMs have been deployed here. Container Engine has been introduced by Docker 17.09. And the Main Engine i.e Kubernetes 1.8.2 is providing the nodes which are completely running. Network Time Protocol (NTP) which runs on Port Number 123 is used for time synchronization between the nodes. The Micro service that is officially used here for

experimentation is VLC Video Streaming. The Template used in Pods helps in the deployment controller and this Pod Template also contains the Image of the Containers of the streaming server. And this Container Image when deployed will start to refer the streaming file. And in our Experiment, the main task of deployment controller to maintain only one pod properly amongst all other pods. This also helps in achieving the availability of the Kubernetes Healing Services. And the Video Streaming micro service is emigrant and when there is such a failure, the video streaming service will start to read from the beginning of the file.

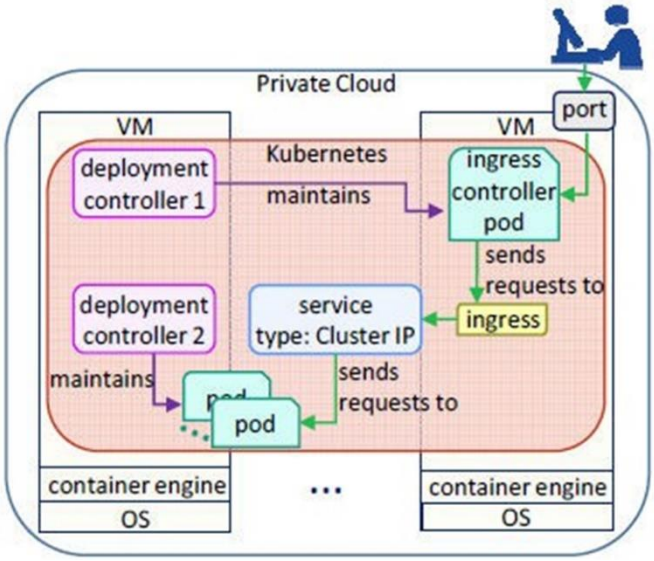

Fig. 1. Private Cloud - Exposing services via Ingress

After successfully passing one round of test, we have identified that there are two sets of Failure Scenarios. The First set is an Application Failure, and this occurs when there is a Pod Failure. The Second set of Failure is known due to Node Failure. In each of these, we will identify and differentiate between these two failure scenarios. Scenario I identifies a Failure reproduced in parallel to an administrative operation which is related to Kubernetes. And this Scenario II is in parallel to external of the Kubernetes. And below we are going to see the Failure scenario of the same in details. And the results of the experiments have been displayed here which represents the failure scenarios and those results can be viewed in Table (I) and Table (II). And all these experiments have been performed say 10 times to get the accurate results. And the results have been filled in the table rows and columns with an average output results taken of those 10 repeated times. And all these Measurements of this experiment have been considered in seconds only.

a) Summary of Pod Failure:

- Plan-I - Pod Failure: With the help of Administrative commands in the CLI the failure pods can be deleted and hence this can also be a showcase to Kubernetes. As explained above that with administrative mode in CLI the failure pods can be removed from the endpoints. And, pods have been given 30 seconds of refined termination period. And at this scenario, the pods won't receive any new requests, and this will also keep serving the requests which was assigned previously. And this also gives an sufficient time to Kubernetes which can be scheduled to a new pod and this also helps in dealing of incoming requests. And the responsible person for maintaining only one replica of pod is deployment controller, and this will bring up the events which always marks the reaction time. And when there is a repair time then the new pod comes up. And this new pod as soon as its landed it starts executing and it starts streaming the video, and this wont be published to users unless and until it has been registered in Endpoints list of the service. And hence, this scenario makes us understand that streaming services are recovered whenever there is a new pod registered on the endpoint list of the service. And the outcome value of this same is presented in Table I.
- Plan-II - Pod Process Failure: Whenever there is a pod which contains some containers i.e Application Con- tainers and when these have

been deployed, parallel to that another container is created which is also the Pod's Container. And these pods are the processes which are running in the OS, may get expelled which ends up resulting in failure of the whole pod. And, to overcome from the pods failure we need to kill the OS Process of the pod container. And the main agent who identifies the failure of Pods process is Kubelet. This Kubelet helps in the reaction time. Similarly, the Kube-proxy helps in removing the pod from the endpoints service list. And when there is a crash in the process, a refined termination signal comes into existence. And this Signal is sent to the application container and the Docker will wait for 30 seconds before being kicked out. And as we have observed in the experiments that whenever there is a Pod Failure and there is Crash over of the pods, the streaming service won't be available. Kubernetes also waits for new pod to start until there will be a termination of the Application Container. simultaneously, the deployment controller will re initiate the pod this is also known as repair time. And the recovery time is calculated when there is a new pod added in the endpoints list of the service. And the measurement of this is represented in Table I.

TABLE I POD FAILURE

Failure Plan (unit: seconds)	Reaction Time	Repair Time	Recovery Time	Outage Time
Plan-I	0.031	0.843	1.422	1.343
Plan-II	0.431	32.443	31.242	32.333

b) Summary of Node Failure:

- Plan-I - Administrative deletion of the Node: A node which gets deleted without draining performed by admin- istrative deletion is one of the method of simulating the node failure. And this node failure is managed by Kuber- netes CLI which is similar to the the Spontaneous Node Failure, and we are going to determine the behaviour of Kubernetes reactions. Here, A node when hosted a pod gets deleted from the Kubernetes CLI Command. And this results in removing the containers and processes which are related to Kubernetes on the active node gets triggered. And the unknown pod which is running on the node which was about to get deleted enters into a state where there wont be any new requests. Therefore the pod failure situation occurs. And we can see lots of different behaviour between this situation and the previous pod failure behaviour(PLAN-1). Also, the pod will serve the requests which were assigned previously just for one second (excluding default 30 seconds). Quickly, the pod is going to be deleted completely and the deployment controller tries to attempt to add a new pod on different node. Here, the repair time gets activated when there is a new born pod. And the recovery time is also taken into consideration when there will be a new pod and this will be added in the endpoint security list. And the calculations for this scenario are represented on Table II.
- Plan II - Externally Triggered Node Failure: As discussed previously, the Kubelet has the responsibility to report the status of the node actively to the master. And in the default configuration of the Kubernetes, the Kubelet gives an update about the Node's Status to the Master Node in every 10 seconds. And this allows the node to be un- responsive for about 4 consecutive status updates before it is confirmed as failed. And after getting confirmed as failed, the master node waits for another 4 minutes and 20 seconds before selecting the pods which are

running on the failed node and which are terminated and those pods are going to be rescheduled on another node. This also means that with the help of default configuration of the Kubernetes, this Kubernetes takes around 5 minutes to be recovered from the node failure. And in order to perform this task, the VM which is hosting the pod is going to be shutdown with the help of Linux Reboot Command.The Master node acknowledges itself that the node is not ready and when there is a missed node status update on the fourth attempt then that would be marked as the reaction time. After 5 minutes, there would be a new pod arrived by the deployment controller, and this pod would be considered and concluded as Repaired. Simultaneously, this new Pod's IP address is added on the endpoints list, and hence the service will be recovered. The assessment value of the same has been represented on the Table II.

IV. ANALYSIS

A. Pod Failure Summary

In the Plan-I case, the reaction time was 0.031 seconds which is very much better than the 0.431 seconds of the failure of pod process(Plan-II). The main reason is that the termination gets activated within the Kubernetes itself, and when there would be a reply appropriately to the termination procedure. And after observing the current state, the Kubelet's health determines that the pod is no longer available and also this helps in predicting how close the next health check failure happens.

TABLE II NODE FAILURE

Failure Plan (unit: seconds)	Reaction Time	Repair Time	Recovery Time	Outage Time
Plan-I	0.031	1.003	1.532	1.513
Plan-II	38.133	25.343	26.432	32.442

The keen observation on the experiments of the same is shown in Fig. 2. The Pod Process gets failed to perform forcefully even after applying the external force which can be read in Plan-II(Fig. 2(b)), and there will be a graceful termination signal when there will be a container of the pod.And hence, the pod process failure gets detected by the Kubelet, and until and unless the Docker gets confirmation from the application container stating that the pod has been terminated. And this leads to the impacts and delays of the SRT(Service Recovery Time) and until the Docker gets new configuration there will be a delay in servicing. And this also ensures that the fault has been occurred due to Bug or might be due to Real Fault. The principles which are required in the Fault isolation has to be responded with proper forceful cleanup of the application containers as soon as there is a confirmation stating that pod's process failure gets detected. And this Grace period gets increased i.e Plan-II is 38.133 seconds. And this is much more significantly longer when there is Plan-I which has Reaction time of 0.031 seconds. Previously, we have observed that in Fig. 2 (a)), Kubernetes always helps in performing the graceful termination and this helps in parallel to repair procedures. For the certain actions of procedures the ordering value is guaranteed for these kinds of procedures. Let's take an example, the terminating pods gets removed from the end points list which helps in recitation of the start of the repair procedures, and the pod gets terminated when there will be a completion and this follows the new pod to be added. This parallelization can be possible only when there will be an assumption that there wont be any fault in the

execution system. This also helps in refreshing that the points that were made earlier about the simulation of the failures which helps on the availability of metrics. And as observed, Plan-I reports an Loss or Outage time of 1.513 seconds and same in Plan-II it reports an loss of 32.44 seconds.

B. Node Failure Summary

And in the node Failure segments, there has been observed that there is an similar differences in all the framed availability output. And for Plan-I, the failure has been triggered from the Shell of Kubernetes, there will be a Reaction time of 0.031 seconds and it is much faster than that of the Reaction time captured in Plan-II i.e 38.133 seconds. As explained in the previously section that, the period of Kubelet's status depends on the period which gets updated by default in 10 seconds and the same is allowed in the number of Missed status by default in 4 seconds.

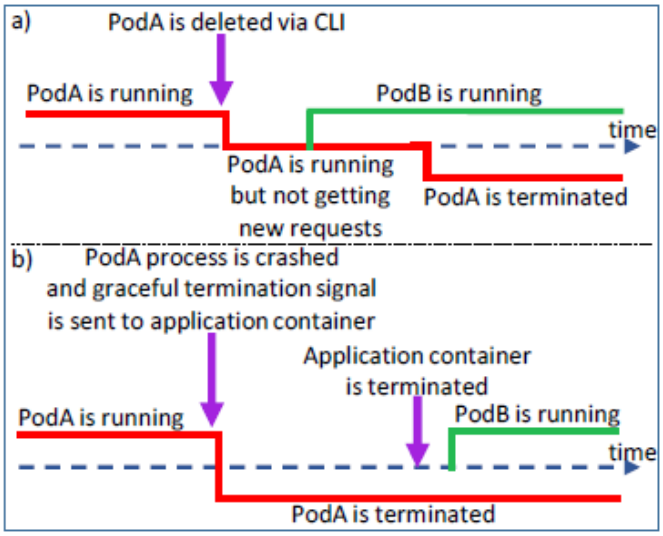

Fig. 2. Analysis of Pod Failures Plans. (a) Plan-I (b) Plan-II

And another important point which was observed in Plan-I is that, though the pod has failed to perform from the Kubernetes, the new pod is going to be initiated and the old pod is going to be terminated. And hence this was expected to behave properly similarly to that of the termination of the pods by administrative method(Plan-I), and hence the new pod will be procured prior to the old one which gets terminated.

V. CONCLUSION

Kubernetes helps in enabling the healing service whenever there is a recovery failure. And this is evaluated through internal operations. Hence, for these kinds of operations, Kubernetes helps to react will in comparison with its reaction to failures which results in performing the external triggers. And according to the survey experiment, the downtime of the Kubernetes service is significantly higher. And it is much important to know that the default configuration of the Ku- bernetes leads to trigger the node failure forcefully. And the output tables explains that these types of failures leads to downtime for 5 minutes, and this in results helps in the amount of downtime calculation for one-complete year and for the high availability requirements are not differentiated and these do not gets satisfied automatically even if an application or a micro service gets deployed in Kubernetes.

As we know that the default configuration can be altered in Kubernetes, analyzing how to reconfigure the Kubernetes and how the reaction would be when there is node failure. This helps in avoiding the network overhead and there would be fake reports which makes it complicated and which requires a better effort.

VI. ACKNOWLEDGMENT

This complete work has been demonstrated and implemented from the NSERC and Ericsson.

VII. REFERENCES

[1]. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2]. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol.2. Oxford: Clarendon, 1892, pp.68–73.
[3]. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4]. K. Elissa, “Title of paper if known,” unpublished.
[5]. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
[6]. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7]. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: Univer- sity Science, 1989.
[8]. S. Newman, “Building Microservices: Designing Fine-Grained Sys- tems,” O’Reilly Media, Inc., 2015.
[9]. N. Dragoni et al., “Microservices: Yesterday, Today, and Tomorrow,”in Present and Ulterior Software Engineering, Springer, Cham, 2017, pp. 195–216.
[10]. D. Jaramillo, D. V. Nguyen, and R. Smart, “Leveraging microservices architecture by using Docker technology,” in SoutheastCon 2016, 2016, pp. 1–5.