Implementation of Enterprise Application Using Micro services on Kubernetes High Availability Cluster

Rishabh Pandey[1], Kartik Kumbhakarna[2], Gitesh Brahmankar[3], Mayur Baravkar[4], Vishakha Bhadane[5]
[1, 2, , 4] Dept of Computer Engineering
[5]Assistant Professor, Dept of Computer Engineering
[1, 2, 3, 4, 5] Jawahar Education Society'sInstitute of Technology, Management & Research, Nashik

Abstract- *Small and loosely coupled modules can be developed and deployed independently to compose an application this is represented by a new architectural style called Microservices. Maintainability, flexibility and scaling can be achieved using this method and also this aims at decreasing downtime in case of upgrade or failure. Kubernetes is one of the Enablers, which is an open-source platform that provides mechanisms including deployment, maintenance, and scaling containerized applications across a knot of hosts. To improve the availability of applications, Kubernetes allows healing through failure recovery actions. As our ultimate goal is to originate architectures to enable high availability (HA) with Kubernetes for micro service based application.*

Keywords- Micro services; Containers; Orchestration; Docker; Kubernetes; Failure; Availability

I. INTRODUCTION

By using APIs, a micro service can be built around a separate business functionality, it runs in its own process and communicates through lightweight mechanisms. Micro services, comprising of an application, can be written using different programming languages and different storage technologies can be used. The drawbacks of the monolithic approach can be approached by Micro services, where the application is a single deployable unit suffering from" dependency hell" and creates barriers for scalability and high availability. Micro services increase velocity and quality. By being small, they can restart faster after upgrade or for failure recovery. Micro services are loosely coupled and failure of one micro service will not affect other micro services of the application. These factors impact the availability of applications as they decrease inaction.Moreover, the fine-grained architecture makes scaling flexible as each service can evolve at its own workload pace. To leverage all these benefits, one needs to use technologies lined up with the characteristics of the architectural style. Containers are lightweight and have faster start up than virtual machines (VMs). Thus, the containerization of micro services can help to speed up the restart after upgrade or for failure recovery. In our experiments, we use Docker, the leading container platform. There is also a need for an orchestration platfom to manage the deployment and operations of containers. Kubernetes is an open-source platform that manages Docker containers in a cluster. Along with the automateddeployment and scaling of containers, the healing is provided by Kubernetes automatically and it restarts failed containers and rescheduling them when their hosts die. This capability improves the application's availability. The architectural style of micro services is being adopted by practitioners and investigated from different perspectives by researchers in academia as well. In, the authors assessed the effectiveness of the HA mechanisms offered by Kubernetes while setting its monitoring intervals to their minimum values. The authors concluded that Kubernetes still needs improvement to provide HA. Such a configuration is not recommended and may lead to false node failure detection.

II. LITERATURE SURVEY

1)The paper titled OpenStack and Docker: building a high-performance IaaS platform for interactive social media applications describes about the Nova-Docker plugin which enables the fast and efficient provisioning of computing resources which can run as a Hypervisor that helps to manage the growth of application users. This is built using an OpenStack IaaS which enables to control data centers for cloud computing. OpenStack standard architecture contains three important roles: Nova, that manages the computation, storage resources are managed by Cinder. The entire networking resources are managed by Neutron across multiple data center. NUBOMEDIA is another approach which enables (PaaS) interactive social media through cloud. The major technologies adopted are Kornet Media Server (KMS) which provides interactive communications through WebRTC media server. Open Baton which manages the lifecycle of media server capabilities using Docker containers. In order to host applications which consumes media server capabilities, Open Shift Origin is enabled. Developers and Administrators are interested more in Docker container than Kernel-based Virtual machine mainly for its Fast Boot time, Direct Access to containers, it can be run on any hardware that supports Linux based OS. Docker containers are lightweight, minimizing the bandwidth needed for deployment using required resources

2) The paper titled Evaluation of Docker as Edge Computing Platform describes about how to overcome problems such as High latency, network bottleneck and network congestion. We can achieve this from moving centralized to decentralized paradigm, Edge computing will be able to reduce application response time for better user experience. Edge computing is enabled with Docker, a platform of container-based technology that has more advantages over VM based Edge computing. This paper mainly evaluates the fundamental requirement for EC that are 1) Deployment and Termination which mainly describes the platform that provides an easy way to manage, install and configure services to deploy the low-end devices. 2) Resource and Service Management that allows users to use the services even when the resources are out of limit. 3) Fault Tolerance which relies on the High availability and reliability to the user. 4) Caching allows the user to experience better performance where Docker images can cache at the Edge. One such that enables Docker concept which was applied on Hadoop Streaming which reduces the setup time and configuration errors. Overall, there are areas of improvement yet, it provides elasticity and good performance
3)The paper titled Model-Driven Management of Docker Containers focuses on management of docker containers, mainly where users findlow-level system issues and it describes how modelling Docker containers helps to achieve sustainable deployment and management of Docker containers. Indeed, Docker system has more advantages than cloud-computing like Azure, Amazon; the Docker containers attains drawbacks in synchronizing between deployed and designed containers. This paper provides a model driven approach to manage not just to design the containers architecture but also represents the deployed containers in target systems. The motivation for this model-driven approach is that present docker containers lacks verification and resource management. The overview of the architecture described in this paper tells us about the three components Docker Model, Connector and Execution Environment.

III. IDENTITY, RESEARCH AND COLLECT IDEA

A software requirements speciation (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement speciation document consistent of all necessary requirements required for project development. To develop the software system, we should have clear understanding of Software system. To achieve this, we need to continuous communication with customers to gather all requirements. A good SRS dene the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios. Using the Software requirements specication(SRS) document on QA lead, managers create test plan. It is very important that testers must be cleared with every detail species in this document in order to avoid faults in test cases and its expected Results

3.1.1 Project Scope

The main futuristic perspective behind this project is to develop a Kubernetes enables healing through its failure recovery actions and they are often evaluated through internal operations. For these types of operations, Kubernetes reacts reasonably well in comparison with its reaction to failures resulting from external triggers. According to our experiments, in the latter, the downtime is significantly higher

3.1.2 User Classes and characteristics

1)The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.
2)The class diagram describes the attributes and operations of a class and also.

3.1.3 Assumptions and Dependencies

Assumptions An assumption is something that you assume to be the case, even without proof. For example, people might make the assumption that you’re a nerd if you wear glasses, even though that’s not true or very nice. What are Dependencies in Project Management? Each item relies on the output of another activity in some way and contributes to the end result of the project. The relationship between two tasks is dened as the dependency between them.

3.2 FUNCTIONAL REQUIREMENT

A functional requirement denes a function of system or its component, where function is described as speciation of behavior between output and input. It involve technical details data manipulation and processing and other specific functionality that dene system is supposed to accomplish.

3.2.1 Pods failure Detection and Identification

This is the basic and initiative functionality of the proposed idea.this functional requirement mainly deals The common way of showing Kubernetes' reaction to pod failure is to delete the pod using the administrative commands in the CL.

3.2.2 Scale down and Scale up

In the pod failure Scenario I, the reaction time is 0.041 seconds which is significantly better than the 0.496 seconds of the pod process failure (Scenario II). The reason is that in the former, the termination is triggered from inside of Kubernetes, which then reacts according to the termination procedure, while in the latter it is up to the Kubelet's health check to detect that the pod is no longer present and this depends on how close to the next health check the failure happens.

An important observation on the experiments is shown in Fig. 2. Although the pod process is failed forcefully in case of Scenario II the orphaned application container of the pod receives a graceful termination signal. Thus, the pod process failure is detected by the Kubelet, which waits for Docker and will not start the repair process before it makes sure that the application container of the pod is terminated as well. This means graceful termination of the application container whose duration depends on Docker's configuration, impacts and delays the service recovery time begin center

3.2.3 High Availability cluster

High availability clusters are groups of hosts (physical machines) that act as a single system and provide continuous availability. High availability clusters are used for mission critical applications like databases, eCommerce websites, and transaction processing systems. High availability clusters are typically used for load balancing, backup, and failover purposes. To successfully configure a high availability (HA) cluster, all hosts in the cluster must have access to the same shared storage. In any case of failure, a virtual machine (VM) on one host can failover to another host, without any downtime. The number of nodes in a high availability cluster can vary between two to dozens of nodes, but storage administrators should be aware that adding too many virtual machines and hosts to one HA cluster can make load balancing difficult.

4.2 FUNCTION MODEL AND DESCRIPTION

4.2.1 Data Flow Diagram

4.2.2 Data Flow Diagram Level 0

A level 0 data flow diagram (DFD), also known as a context diagram, shows a data system as a whole and emphasizes the way it interacts with external entities. This DFD level 0 example shows how such a system might function within a typical retail.

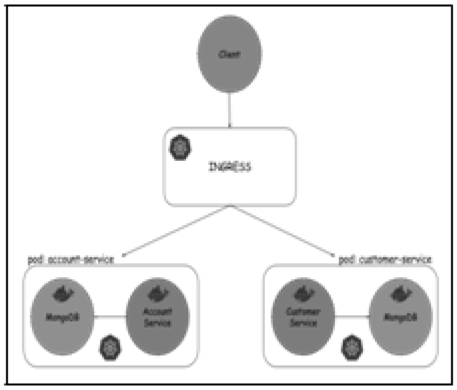

Figure 4.3: Data Flow Diagram Level 0

4.2.3Data Flow Diagram Level 1

As described previously, context diagrams (level 0 DFDs) are diagrams where the whole system is represented as a single process. A level 1 DFD notates each of the main sub-processes that together form the complete system. We can think of a level 1 DFD as an view" of the context diagram.

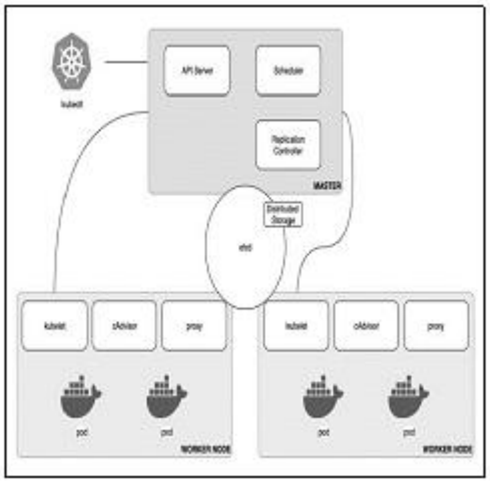

Figure 4.4: Data Flow Diagram Level 1

4.3 ACTIVITY DIAGRAM

An activity diagram visually presents a series of actions or ow of control in a system similar to aowchart or a data ow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram.

Activities modeled can be sequential and concurrent.
Client-side: Front-end App and Browser Extension.
API Gateways: Auth API and Main API
Micro services: Sign in, photo and payment Micro services

4.4 ARCHITECTURAL DESIGN

4.4.1 Architectural Description

Goal Initialization:- Users can have one of two goals:

1) Deploying containerized applications in Kubernetes cluster running in a private cloud
2) Evaluation of the availability of micro service based applications deployed with Kubernetes.

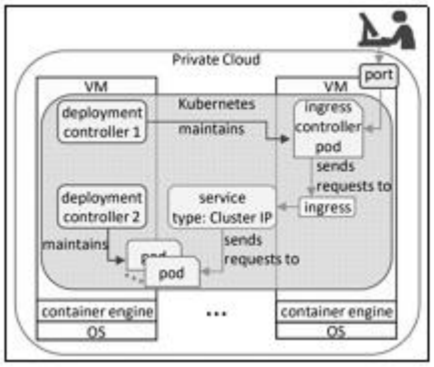

Figure 4.6: Architectural View

We used the architecture in Our cluster is composed of three VMs running on an OpenStack cloud. Ubuntu 16.04 is the OS on all VMs and Docker 17.09 is running as the container engine. Kubernetes 1.8.2 is running on all nodes. Network Time Protocol (NTP) is used for time synchronization between nodes in the cluster. The microservice we use in these experiments is VLC video streaming. The pod template provided to the deployment controller contains the container image of the streaming server, which once deployed will stream from a file. In our experiments, the desired number of pods that the deployment controller needs to maintain is one. This can help to understand the achievable availability through the Kubernetes' repair action by itself. Our video streaming micro service is stateless and in case of failure, the video will restart from the beginning of the file. We identified two sets of failure scenarios. In the first set, an application failure is due to a pod failure whereas in the second set it is due to a node failure. In each set, we distinguish between two failure scenarios. Scenario I designates a failure simulated by an administrative operation internal to Kubernetes while Scenario II is simulated by a trigger external to Kubernetes. Below, we explain each failure scenario in details. The results of our experiments for these failure scenarios are presented in

IV. APPLICATIONS DEPLOYED WITH KUBERNETES

Availability is a nonfunctional requirement which is measured as the outage time over a given period [10]. High availability is achieved when the system is available at least 99.999per year . Some characteristics of micro services and containers such as being small and lightweight naturally contribute to improving availability when supported by Kubernetes' healing capability . In this section, we describe the experiments we conducted to evaluate from an availability perspective the deployment of a micro service based application in a Kubernetes cluster running in a private cloud Kubernetes reacts to a pod failure by automatically starting a new pod and therefore it is expected to improve the availability of the service provided by the pod. The common practice to evaluate Kubernetes' reaction to failure is to simulate failures through administrative operations (e.g. delete the pod or the node) using the Kubernetes command line interface (CLI). Due to the use of Kubernetes' administrative operations, such a "failure" is not a spontaneous event that Kubernetes needs to detect and react to. Instead, the operation is executed by Kubernetes in due order often in a graceful manner. Therefore, these operations cannot reflect common execution failure scenarios, which are anything but graceful and happen spontaneously as a result of external failure events (e.g. process or physical node crash). Drawing conclusions based on such administrative operations would not be accurate.

A. Availability metrics

The metrics we use to evaluate Kubernetes from availability perspective are defined hereafter.

Reaction Time: The time between the failure event we introduce and the first reaction of Kubernetes that reflects the failure event was detected.

Repair Time: The time between the first reaction of Kubernetes and the repair of the failed pod.

Recovery Time: The time between the first reaction of Kubernetes and when the service is available again.

Outage Time: The duration in which the service was not available. It represents the sum of the reaction time and the recovery time.

4.4.4 Micro services

Micro services" - yet another new term on the crowded streets of software architecture. Although our natural inclination is to pass such things by with a contemptuous glance, this bit of terminology describes a style of software systems that we are finding more and more appealing. We've seen many projects use this style in the last few years, and results so far have been positive, so much so that for many of our colleagues this is becoming the default style for building enterprise applications. Sadly, however, there's not much information that outlines what the micro service style is and how to do it. In short, the micro service architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

4.4.5 Container Deployment

Container Deployment is the next step in the drive to create a more flexible and efficient model. Much like VMs, containers have individual memory, system files, and processing space. However, strict isolation is no longer a limiting factor. Multiple applications can now share the same underlying operating system. This feature makes containers much more efficient than full-blown VMs. They are portable across clouds, different devices, and almost any OS distribution.

4.4.6 Containers Orchestration

Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments. Software teams use container orchestration to control and automate many tasks:

Provisioning and deployment of containers.
Redundancy and availability of containers.
Scaling up or removing containers to spread application load evenly across host infrastructure.
Movement of containers from one host to another if there is a shortage of resources in a host, or if a host die.

V. CONCLUSION

Kubernetes enables healing through its failure recovery actions and they are often evaluated through internal operations. For these types of operations, Kubernetes reacts reasonably well in comparison with its reaction to failures resulting from external triggers. It is important to note that the default configuration of Kubernetes results in a significant service outage in case of externally triggered node failure. As our measurements show for these types of failures, the outage time is about 5 minutes, which is equivalent to the amount of downtime allowed in a one-year period for a highly available system. These differences indicate that high availability requirements are not satisfied automatically by deploying an application or a micro service with Kubernetes.

REFERENCES

[1] S. Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Inc., 2015.
[2] "Microservices," martinfowler.com. [Online]. Available: https://martinfowler. com/articles/microservices.html. [Acc.: 06-Feb-18].
[3] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow," in Present and Ulterior Software Engineering, Springer, Cham, 2017, pp. 195–216.
[4] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in SoutheastCon 2016, 2016, pp. 1–5
[5] Kanso, H. Huang, I. T. J. Watson, and A. Gherbi, "Can Linux Containers Clustering Solutions offer High Availability?," p. 6.
[6] "Docker - Build, Ship, and Run Any App, Anywhere." [Online]. Available: https://www.docker.com/. [Acc.: 02-Jan-18].
[7] "Kubernetes," Kubernetes. [Online]. Available: https://kubernetes.io/. [Acc.:
[8] 24-Jan-18].