IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

Orchestrated Clusteres using Kubernetes on Cloud Web Services

Mahesh G Prasad[1*], S. Vignesh[2]

[1] Department of Computer Science and IT, Jain (Deemed to-be University), Bangalore, India
[2] Department of Computer Science and IT, Jain (Deemed to-be University), Bangalore, India

[*]*Corresponding Author: maheshgprasad@gmail.com*

DOI: https://doi.org/10.26438/ijcse/v7i5.856860 | Available online at: www.ijcseonline.org

Accepted: 14/May/2019, Published: 31/May/2019

Abstract— In the rapidly developing world of technology where a new concept or a plausible implementation of a long-lost concept is born and is deployed on the cloud for various benefits that it offers. With the rapidly growing and ever-increasing dependency on cloud, it is of the most importance to ensure stability from a developer standpoint and it is also very important that it is dynamically scalable and fault-tolerant and in a hope to achieve the same, here is a project set to fulfill the necessities of the modern application which could benefit from the autoscaling resource optimized clusterization. This is set to be achieved by using the help of a popular containerization platform called Docker. Using a container translates to better isolated environments which are independent of their operational state and in-turn better provides robust security and generally contributes to the fault tolerant nature of an infrastructure adding Kubernetes in the mix not only makes the deployment highly robust and scalable it also makes deployment simpler for the developer mainly because of its declarative instruction advantage over Docker CLI or even KubeCTL's Imperative instruction type. Majorly optimizing the efforts of developer's deployment rather than the build from scratch approach that was previously used extensively. The implementation is designed with fluidity in mind and its main intention is to provide a seamless experience regardless the operations being carried out in the background. These background operations may include spawning of new nodes or pods, re-creation of deteriorated or erroneous containers (PODS) inside the nodes.

Keywords—Scalable, Fault-Tolerant, Containerization, Docker , Clusterization, Robust, Kubernetes, pods,CLI.

I. INTRODUCTION

In this modern day and age, almost all organizations and major corporations are making their move to the cloud. As it makes it possible for them to cut down on infrastructural costs and mainly because they are not required to fork-up the cost of investing on datacentre grade hardware, with prime focus now on cloud, almost all the companies which have had their cloud presence are seeking ways to improve their online presence more efficiently and securely. This being their key objective, some of them might prefer to use fail-over servers, some of them may prefer to have their cloud processes outsourced and some other companies go above and beyond to stay on the bleeding edge of technology. That is where docker and Kubernetes comes into play. Docker is a container-based platform where a little memory, storage, network is isolated to be then used by a server or a service such as apache (httpd2.0), or NGINX servers or may be a mail server like MailU. It goes above and beyond the humble virtualization to bring improvements such as isolation between the containers and a system wherein an application inside a container fails, it is limited to that particular container itself and does not affect other healthy and running containers. Taking matters one step ahead comes in Kubernetes, a container orchestration tool which was initially built for the use of Google applications by Google and then later released to be used by Dev-Ops engineers worldwide, this came in advantageous to anyone and everyone who started to implement Kubernetes in their infrastructural considerations. It being open source and with that came a large community of developers worldwide who contribute to the betterment of the entire system. And also support any fellow member of the community to obtain guidance if and when a technical difficulty was encountered. Hoping to learn and implement the rapidly pacing Kubernetes technology, I bring to you a paper based on Kubernetes which is just a scratch on the surface to what it is truly capable of.

II. RELATED WORK

The Adoption of Microservice based architecture is for the foreseeable future is ever expanding. In a way, this architecture with its tiny, and modular nature conceptualized, built deployed and scaled with dependency, however for a large multinational corporate to move to a Microservice based architecture remains a concern, with that being said, Kubernetes provides a platform that breaks the normal stereotype of how a microservice is thought after, it is an open source platform that defines a set of building blocks which provides a mechanism that can deploy, maintain, scale and heal the containerized microservices. Thus, hiding the complexity of the microservice orchestration while also managing their availability.

III. METHODOLOGY

The work was carried out by setting up infrastructure on the Google Cloud Platform (GCP). By using the Google Kubernetes Engine (GKE) for the project. The deployment process was carried out with the help of a Content Integration tool called as Travis CI. And the Application which was deployed to the cluster was a Fibonacci calculator which used various other services to function. Various findings are reported in the paper.

IV. PROBLEMS EXISTING IN THE CURRENT ENVIRONMENT

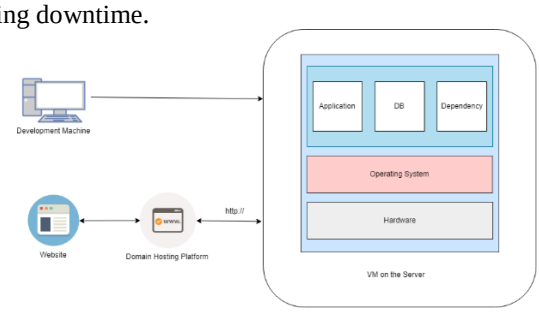
The Architecture was built around just a virtual machine on the web or a local server taking care of all the duties of serving the front-end application, the database and other dependencies however, this type of architecture introduced a point of failure where, in case a failure occurred probably on the front-end side where large volumes of influx requests stalled the system, the entire server would be non-operational and there would be no sure way of telling what in the system which has failed. Simply put it was easily prone to malfunction, this could totally mean downtime and losing out on customers and in turn income.

V. PROBLEMS THAT ARE BEING ADDRESSED BY THE MODERN IMPLEMENTATION.

Primarily the switch to containerization which paved the way to better isolation of environments i.e. the front-end from the database etc. Also, from each other which loosely translated to being much fault tolerant and them failing on their own environments and not pulling down other active services along. Containers could be spun out extremely fast and all that is necessary are the Docker file, Docker-Compose file. Deployment file or automation scripts which are responsible for defining the architecture and the environments within. With the help of containerization, the applications have an underlying environment often just to cater the needs of that particular application and only it. The Container images built using the several aforementioned techniques and are pushed to docker hub repository for smooth deployment in the later stages of the build. These images which are built are then deployed on top of a cluster is managed by Kubernetes where the containers reside as PODS. These PODS are managed entirely by Kubernetes as in kills and re-generates the environment when the running environment gets updated or becomes unresponsive.

VI. DRAWBACKS OF EXISTING SYSTEM

The existing infrastructure consists of a local development machine on which the deployable content is created and then it is hosted on a web server either on premise or on a public cloud such as Amazon Web Services or Azure. This architecture is compact and is capable of handling lower scale workloads such as for hosting a website for a start-up or a small-scale industry which provides tools or drivers for some of its services. However, this infrastructure is non fault tolerant as in even when a small dependency such as an update to the internal infrastructure would have the capability to take it down the entire infrastructure without a hitch causing downtime.

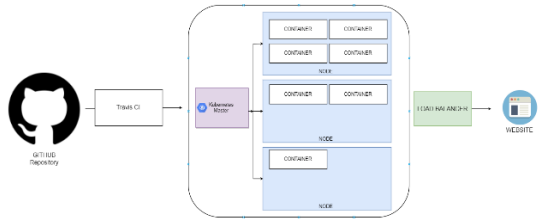


1. Existing Application/Environment

VII. PROPOSED SYSTEM

The Proposed System consists of containerization using the most widely adopted docker platform on top of which the whole architecture is implemented, Docker platform was opted for the architectural design as it has significantly low specification requirements for operation and the container images used in this project are built using a bare minimum version of Linux named as Alpine Linux which has a maximum size of about 5MB. Along with other features such as container level isolation, minimum overhead, security (as containers cannot communicate amongst each other or be aware of each other's existence unless mapped by the developer.), failure containment (in case of a failure fails in its own container and does not affect any other service or containers) and finally fault tolerance. To top it off, Kubernetes the orchestration tool is designed and developed by Google and is now backed by a large community ever since it was made open source and available to everyone. Kubernetes is used to orchestrate these containers built using the images to enhance the replication as a part of fault tolerance and load handling, pod (container) health monitoring and appropriate actions taken care of by

Kubernetes to ensure that a service running inside of a pod is completely healthy and if not take necessary actions such as killing the failed pod and creating a new one. All Kubernetes actions take place almost instantaneously so much so that the end user would not notice any disturbance because of all the background processes taking place all thanks to the agile and fluent nature of Kubernetes.



2. Proposed Application/Environment

VIII. KUBERNETES

Kubernetes is a container orchestration tool for docker and It is open source which loosely translates to having a lot of support from the developers and the developer community. It allows the user to schedule containers on a cluster of computers or nodes as they are called, provides the user with the option to deploy multiple containers on one machine. Provides an opportunity to run long running services (e.g. Websites). Kubernetes will manage the state of the deployed containers, which means that it can start containers on specific nodes, restart a particular container or containers when they get killed either by accident, because of errors and finally has the ability to move a container from one node to the other without disrupting the overall work flow that depends on the systems to be operational. One more advantage of Kubernetes is that it can be run on-premise, on a public cloud such as AWS or Google Cloud and also be used to orchestrate hybrid compute architecture. Other advantages of Kubernetes include Modularity, its open source approach to availability and usage, great community support from online forums such as Github and majorly packs the greatest advantage of them all which is being backed by the global technology giant Google.

Kubernetes object: They are persistent entitles in the Kubernetes system, Kubernetes uses these entitles to represent the state of the cluster, such as: the specific containerized application is running and on which nodes, the resources available to those applications, policies around how these applications behave, such as restart policies, upgrades and also fault tolerance.
Kubernetes API: When we specify the API version, that scopes / limits the types of objects that we can specify that we want to create within a given configuration file. This essentially opens up the ability to access a pre-defined set of different object types. Which would give us access to Component Status, Configuration Mapping, End Points, Event, Namespace and Pod. Unlike Docker which has containers to house all the images and execute them, Kubernetes makes use of a concept called the objects which may be of the type deployment, service, volumes, Pods etc.

POD: Is the smallest deployment option to deploy a single container, which also has the ability to run one or more closely related containers within itself.

Services: Deals with a major portion of the networking aspect for a Kubernetes cluster. These services are classified into four major sub types which are:

a. **Cluster IP**: Assigns an IP address to the running cluster so that we could access the services running atop of the cluster.
b. **Node Ports**: Exposes a container to the outside world (this type however is only used for developmental purposes as exposing the actual node ports would introduce a vulnerability).
c. **Load Balancer**: Handles the incoming and outgoing network traffic.
d. **Ingress**: Exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Deployment: In order to deploy a Kubernetes cluster a set of instructions should be issued to the Kubernetes master stating the specific requirements and also the mapping within the deployment file. This deployment file is written in YAML and is either deployed to the Kubernetes cluster using the dashboard or the command line interface. This deployment file would be responsible for the deployment of either a POD or service. On the master, there is a variety of different programs that control the whole Kubernetes cluster. We make use of the "kube-apiserver", the kube api server is wholly responsible for monitoring the current status of all the different nodes which are inside the cluster and making sure that they are all essentially doing the correct thing. The master when given with a deployment file, makes a note of all the responsibilities that are to accompany the following deployment. So that is all the tasks, roles and responsibilities that we assign to the master to handle / take care of by itself. Consider this, when the master is handed over the task to fetch deploy and monitor the master, it will make sure that the service and pod which are configured are fault tolerant and is operational even in case of failures which may or may not be intentional. Any node created by the master is an autonomous body and that they do not communicate with other nodes but report directly to the master node and any connection which needs to be routed is done by the master itself.

The Deployment file: A deployment file is responsible for a pod or a service, the deployment file is always fed to the Kubernetes master using the KubeCTL command line interface to Kubernetes master.

On the master, there are a variety of different programs that control the whole Kubernetes cluster. Of which, we make use of "kube-apiserver". This server is a 100% responsible for monitoring the current status of all the different nodes inside the cluster and making sure that they are in complete operational status.

The master when given with a deployment file, makes a note of all the responsibilities defined inside of the deployment file, so that all the tasks, the rules and the responsibilities that we assign to the master to handle / take care of by itself. Consider this, when the master is handed over the task to fetch, deploy and monitor. It will follow the directions provided to it and then make sure that a service which is configured to be fault-tolerant is up and operational in case of any failures (be it natural occurrence and simulated/ created). Any node created by the master is an autonomous body un the sense they do not communicate with other nodes but do report to the master node and any connection which needs to be established is taken care of by the master. finally, to deploy something, we update the desired state of the master with a configuration file, to which the master continually works to meet the desired state. When we create a deployment object, it is going to have attached to something called the pod template. The pod template is essentially a block of configuration file which defines what any pod that is created by this deployment should look like.

Any modifications done to the deployment object will reflect the same changes on the running pods either by modification or by killing the pod and creating a new one up to spec.

The deployment object will constantly watch all of the different pods that It maintains, it is going to be watching their state and making sure they are always in a correct state. Which would be Active State out of all the states such as Inactive, Non-Existent, killed, crashed, terminated. In the deployment file metadata about the pods help identify them better.

IX. DDATA STORAGE DESIGN

Starting with deployment, the build automation tool Travis CI stores variables such as usernames and passwords in order to access the docker images from docker hub as well as the Cloud Web Services here the depicted environment is of AWS. Using this Environmental variables, the application could be successfully deployed. On Kubernetes cluster, for the application to write data into the database, credentials are required and so in comes the Kubernetes in-cluster environmental variable setting where the password to access the Postgres database is sourced from the environmental variable instead of user input or hard coded plain text in the application. Postgres is a database hosted inside of a container which is encapsulated inside of a Kubernetes pod. The above table describes the field and its type of value which is to be ingested by the database.

Redis is an in-memory datastore and is hosted inside a different container encapsulated in a different pod and it has the ability to store data as key- value pair and hence stores the index and the calculated value obtained after the application performs Fibonacci operation.

X. PPHASED APPROACH

The Front End and the back-end code is written on a local machine which is then moved to a containerized environment and are then tested locally on Docker Desktop for operational health. The Containers are then linked together by docker networking locally using docker-compose and then tested for its working. Once all the local testing is done, a repository is created on Github and all the code for creation of containers, the source code for the application itself and everything related to is then committed and pushed to the previously created repository. The deployment part of this process takes place in phases, namely phase 1: where all the environment on the cloud is set up. Phase 2: Set up a deployment mechanism that would deliver the docker images. Phase 3 would be to set up a domain to the ip address exposed by the cluster ip service.

In Phase 1, Cloud Instances and Its correspondent environmental variables are set-up on TRAVIS CI and GKE.

In Phase 2. The script takes care of the rest of the automated deployment and setting up process. However, there are a certain aspect of the networking configuration which are to be done by the developer.

In Phase 3, the remaining process is to associate a domain name with the exposed ip that is given out by the internal load balancer of the architecture and also the external load balancer assigned to the cluster by the cloud provider.

XI. PPSEUDO CODE FOR KUBERNETES CONTAINER CONFIGURATION

spec => Replica:3
matchLabels: => Component: Web
Template => Containers: => Image:
Image:<docker id>/frontend
Ports: 80

XII. CCONCLUSION

To conclude the paper, it can be said that the key objectives which was to containerize an application and make use of Kubernetes to orchestrate the entire infrastructure that was built on a cloud platform was set to be achieved was completed successfully. However, it is believed that this paper barely scratches the surface of Kubernetes and its potential which could be explored further by extensive study and research.

REFERENCES

[1] Jonathan Baier, "Getting Started with Kubernetes", PACKT Publishing, 2015
[2] Publishing Docker images, https://www.howtoforge.com/tutorial/building-and-publishing-custom-docker-images. (as on April - 2019).
[3] Kubernetes Concepts and Fundamentals, https://kubernetes.io/docs/concepts/ (as on April - 2019).
[4] Docker- Stories, Accelerate digital transformation with docker, https://www.hub.docker.com
[5] Multi Author, "YAML", Tutorials Point, 2018
[6] Docker INC, "Introduction to Docker" , Docker Fundamentals 2cb8348, 2014
[7] Amazon Web Services Route 53 Documentation, https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-domain-registration.html
(as on April - 2019).
[8] Kubernetes Encryption before REST, https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/ (as on April - 2019).
[9] Leila Vayghan, Mohamed Saied, Maria Toeroe, Ferhat Khendek, "Kubernetes as an Availability Manager for Microservice Applications", Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson , October 2018.

Authors Profile

Mr. Mahesh G Prasad pursed Bachelor of Computer Applications (BCA) from Dayananda Sagar Institutions affiliated to Bangalore University, Bangalore in 2016 and will complete Master of Computer Application from Jain University in year 2019.

Mr S Vignesh pursed Bachelor of Technology from Sri Vidya College of Engineering and Technology affiliated to Anna University in year 2009 and Master of Information technology from MIT Anna university in year 2015 .He is currently working as Assistant Professor in Department of IT, Jain University. He has published research papers in reputed international journals available online. His main research work focuses on Cryptography Algorithms, Network Security, Cloud Security and Privacy, Big Data Analytics, Data Mining, IoT and Computational Intelligence based education.

