International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.512|

|| Volume 9, Issue 7, July 2020 ||

Generalized Open Blue Green Deployment with All Types of Cloud

Ganesh Kudale[1], Dr. Shyam Gupta[2]

Department of Computer Engineering, Siddhant College of Engineering, Sudumbre, Pune, India[1, 2]

ABSTRACT: Generalized open blue green deployment strategies for any cloud native application is a release model that gradually transfers user traffic from a previous version of an app or microservice to a nearly identical new release—both of which are running in production at enterprise scale.
The old version can be called the blue environment while the new version can be known as the green environment. Once production traffic is fully transferred from blue to green, blue can standby in case of rollback or pulled from production and updated to become the template upon which the next update is made.
There are downsides to this continuous deployment model. Not all environments have the same uptime requirements or the resources to properly perform CI/CD processes like blue green. But many apps evolve to support such continuous delivery as the enterprises supporting them digitally transform.

KEYWORDS: Application Release Strategies, Blue Green Deployments in Cloud, Hybrid Cloud Deployments, Containerized Application Release Models in Cloud.

I. INTRODUCTION

A Generalized Open Blue/green deployments provide near zero-downtime release and rollback capabilities with open source technologies with zero software cost for a large scale enterprise environment. The fundamental idea behind blue/green deployment is to shift traffic between two identical environments that are running different versions of your application. The blue environment represents the current application version serving production traffic. In parallel, the green environment is staged running a different version of your application. After the green environment is ready and tested, production traffic is redirected from blue to green. An application is developed and deployed to any public/private/hybrid cloud environment, having two separate, but identical, environments—blue and green—increases availability and reduces risk. In this Quick Start architecture, the blue environment is the production environment that normally handles live traffic. The CI/CD pipeline architecture creates a clone (green) of the live Application environment (blue). The pipeline then swaps the URLs between the two environments. environment in service is awareness about the circumstance. Then it can be easily adjust to the dynamic service.

A pod is a higher level of abstraction grouping containerized components. Any application which is cloud native and stateless can be containerized into POD. A pod consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources. The basic scheduling unit in Kubernetes is a pod.

Each pod in Kubernetes is assigned a unique Pod IP address within the cluster, which allows applications to use ports without the risk of conflict. Within the pod, all containers can reference each other on localhost, but a container within one pod has no way of directly addressing another container within another pod; for that, it has to use the Pod IP Address. An application developer should never use the Pod IP Address though, to reference / invoke a capability in another pod, as Pod IP addresses are ephemeral - the specific pod that they are referencing may be assigned to another Pod IP address on restart. Instead, they should use a reference to a Service, which holds a reference to the target pod at the specific Pod IP Address.

A pod can define a volume, such as a local disk directory or a network disk, and expose it to the containers in the pod.[26] Pods can be managed manually through the KubernetesAPI, or their management can be delegated to a controller.[23] Such volumes are also the basis for the Kubernetes features of ConfigMaps (to provide access to configuration through the filesystem visible to the container) and Secrets (to provide access to credentials needed to access remote resources securely, by providing those credentials on the filesystem visible only to authorized containers).

Once Application is containerized into pod then creating an deployment and services can be easier with kubernetes and making those deployment be manageable with automation tools and configuration management can help you to manage the lifecycle of a container

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.512|

|| Volume 9, Issue 7, July 2020 ||

II. LITERATURE SURVEY

Approach introduced by Google for an application releasesvia kubernetes with easy ways. Understanding how people use or want to use Kubernetes can help us shape everything from what we build to how we do it. To help us understand how application developers, application operators, and ecosystem tool developers are using and want to use Kubernetes, the Application Definition Working Group recently performed a survey. The survey focused on these types of user roles and the features and sub-projects owned by the Kubernetes organization. That included kubectl, Dashboard, Minikube, Helm, the Workloads API, etc.

The results are in and the raw data is now available for everyone.

There is too much data to summarize in a single blog post and we hope people will be able to find useful information by pouring over the data. Here are some of the highlights that caught our attention. Introducing Jenkins with kubernetes will help to manage open cloud scale application deployments easily.

Jenkins evolved from being a pure Continuous Integration Platform to a Continuous Delivery one, embracing the new design tendency where not only the build but also the release and the delivery process of the product is automated. In this scenario Jenkins becomes the orchestrator tool for all the teams/roles involved in the software lifecycle, thanks to which Development, Quality & Assurance and Operations teams can work closely together. Goal of this paper is not only to position Jenkins as a hub for CD, but also introduce the challenges that still need to be solved in order to strengthen Jenkins' tracking capabilities.

III. PROPOSED METHODOLOGY

We should first identify application components which can be cloud native. Containerized all the independent components of an application and create Pods out of it. Tag an image of each component and attach a version of each to it. Publish it to the local personal registry.

Attached a development code repository and version registry of an image to automation engines like Jenkins/Ansible/Chef to it which will be open source and can be generalized to it as per application requirement and it's configuration management part.

Switching to microservices an application simplified the components within application and its operations but increases the interaction complexities within services of an application. In the cloud hundreds to thousands microservices run together to form an application as a single unit. Like Google, AWS are the infrastructure provision applications, but they are made up of thousands of microservice together to form a cloud. To maintain the compatibility and health of an application each microservice must have a valid and stateful response for other microservice even though it's adding new features to it. Each microservice of an application raises its version to higher with backward compatibilities between other microservices. In the cloud, you will always find multiple versions of a microservice will be running in the cloud.

Create CI/CT/CD cycle or pipelines from the automation tools via multi region cluster of kubernetes

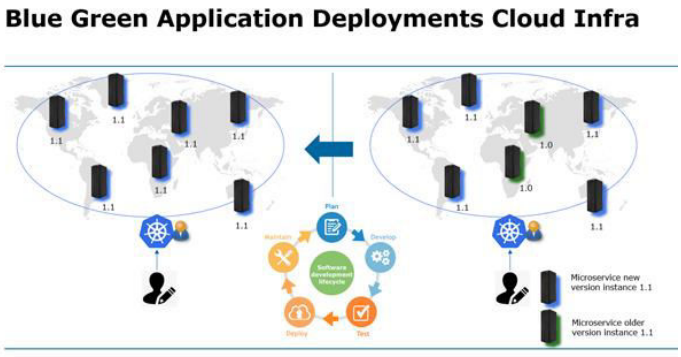

Fig 1: Proposed Generalized Blue Green System Strategy

Figure 1 shows the architecture of the proposedopen generalized strategy for application release via blue green lifecycle.

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.512|

|| Volume 9, Issue 7, July 2020 ||

IV. RESULT ANALYSIS AND DISCUSSION

A. Results

Fig 2: Cloud Pricing Comparison with CI/CT/CD

Cost Saving Analysis: In analysis, the author applies the open source tool pipelines for any application after containerization which saves the application release cost per cloud. A generalized open source cloud independent pipeline for application release pipeline will help huge enterprise to choose an independent cloud based on the required platform which they can choose based on cloud costing model from cloud providers

B. System Requirements

Software Requirement

- ·Google Cloud/AWS/Azure/Kubernetes
- ·Any programming platformwith automation engines
- ·Docker - Application Containerization

Hardware Requirement:

- ·Cloud Scale Infra
- ·Processor: All supported cloud processors
- ·Computation: All processing computational
- ·Storage: Cloud Storage

V. CONCLUSION

User's dynamic decision to cloud deployment model will leverage an enterprise to save huge costs with a generalized open blue green deployment pipeline with lowest downtime.

REFERENCES

[1] Marhubi, Kamal (2015-09-26). "Kubernetes from the ground up: API server". kamalmarhubi.com. Archived from the original on 2015-10-29. Retrieved 2015-11-02.

[2]LaToza, Thomas (2019) "Deployment" (PDF). Archived from the original (PDF) on 2020-01-14. Retrieved 2020-01-14..

[3] Fowler, Martin (2010-03-01). "Blue Green Deployment". Archived from the original on 2020-01-10. Retrieved 2020-01-14.

[4] Posta, Christian (2015-08-03). "Blue-green Deployments, A/B Testing, and Canary Releases". Archived from the original on 2018-03-30. Retrieved 2020-01-14.

[5] AbhishekVerma; Luis Pedrosa; Madhukar R. Korupolu; David Oppenheimer; Eric Tune; John Wilkes (April 21–24, 2015). "Large-scale cluster management at Google with Borg". Proceedings of the European Conference on Computer Systems (EuroSys). Archived from the original on 2017-07-27.

[6] AWS Whitepapers, © 2016, Amazon Web Services Y. G. Jiang and J. J. Wang, "Blue Green Methodologies", IEEE Transaction Cloud, Jan/Mar 2016.