Proceedings of the Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)
IEEE Xplore Part Number:CFP20OSV-ART; ISBN: 978-1-7281-5464-0

Comparison of zero downtime based deployment techniques in public cloud infrastructure

Chaitanya K. Rudrabhatla
Executive Director- Solutions Architect
Sony Pictures Entertainment
Los Angeles, USA
Chaitanya2409@gmail.com

***Abstract*—The ever-metamorphosing nature of modern businesses demand changes to be rapidly deployed to the production environments without causing any major outages. Cloud computing in conjunction with containerization and orchestration technologies has undoubtedly contributed to the development and proliferation of quicker and light weight deployment strategies. Of late, Micro service architecture (MSA) has become the de-facto standard for building such lightweight containers, which made the deployments even quicker. Though public cloud technologies and MSA based containers helped in rapid deployments, continuous deployment with zero downtime is not something which comes out of the box. It needs the implementation of Blue Green (BG) deployment techniques to achieve the zero downtime with no disruption to end users while applying the patches. But there are multiple ways to implement the BG deployment pattern. This research paper aims to address three things - (i) Explore Blue Green, Canary and Rolling deployment techniques possible in the cloud platforms (ii) Perform a research project, to simulate the BG deployment strategies using DNS routing swap versus Load Balancer swap versus newer image switch techniques(iii) Draw the comparisons and elucidate which deployment strategy is suitable for which scenario**.

Keywords—component; Cloud computing, Containerization, Orchestration, Microservices, Blue Green strategy, Canary strategy, Zero downtime, Rapid deployments.

I. INTRODUCTION

Before the advent of public cloud platforms, majority of the web applications followed a design pattern where in the backend and front end was compiled and packed into a single monolithic unit called web archive (war) or an enterprise archive (ear) file. These files were usually deployed on the enterprise middle ware servers like WebLogic, WebSphere or JBoss to name a few, which run on the virtual machines in the local data centers. These virtual machines run on top of a bare metal server which has a host operating system and multiple guest partitions done using a software like hypervisor [11]. A typical monolithic deployment architecture can be represented as in Fig 1. Since the setup with virtual machines and hypervisor software is heavy, the time required to bootup the server is high. Also, with the heaviness of binaries and libraries involved in the enterprise application and web servers involved in the design, the deployment time was significantly high. In this traditional approach, a deployment would mean replacing the older version with the newer version of the application on the same infrastructure. This not only causes outages during the deployments [4], but also makes the rollback difficult to implement. To counter these shortfalls and solve the heaviness in the design, micro service architecture (MSA) was brough into place. This involved breaking the heavy monolith into multiple smaller, light weight and fully decoupled units. This design brought many advantages like service isolation, separation of concerns, leverage to use heterogeneous technologies, and faster defect identification and build cycles to name a few. The concept of containerization was long existent in the Unix world. Many technologies like Docker, Rocket, Odin etc. have gained a lot of traction with the advent of public cloud platforms like AWS, Azure and GCP to name a few. The primary reason for that being, containers are super lightweight compared to virtual machines. The underpinning reason for the lightness comes from the fact that a virtual machine is an atomic unit which contains all the heavy libraries and binaries in itself whereas a container shares the common files with others, thus making it a lighter unit. These container technologies are found to be highly suitable for the micro services which are fundamentally isolated from each other [7]. Public cloud platforms added a great support for the container deployments due to their ability to spin up the compute resources in a very short span of time. To handle the deployment and configurations of the containers in the cloud, newer orchestration technologies like Docker swarm and Kubernetes came in to light [17]. This combination of micro service architecture, containerization, orchestration and public cloud compute resources have significantly reduced the deployment timelines [5] [9]. However, the modern businesses are extremely dynamic and fluid in nature. They demand changes in the web applications continuously and rapidly [1]. Though the containerized microservices running on the public cloud infrastructure are light weight and quick to deploy, they still cause an outage when a new patch is deployed [12]. A zero-downtime deployment would ideally not cause any outage to the end users. The old version continues to run till the new version is ready. That way the traffic is routed seamlessly to the newer version without any disruption. To handle the deployment of newer changes without causing an outage [10], Blue Green (BG) deployment strategies need to be applied. There are multiple ways to achieve the BG deployment in a cloud platform. Identifying the right strategy based on the need is critical for attaining the optimal performance during the deployment [13]. Rest of this paper is designed as follows – In section II we explain the Blue Green deployment methodology in the cloud environment. In section III we walk through the research project performed to simulate various BG deployment approaches. In section IV we elucidate the pros and cons of each of those strategies and present the conclusions.

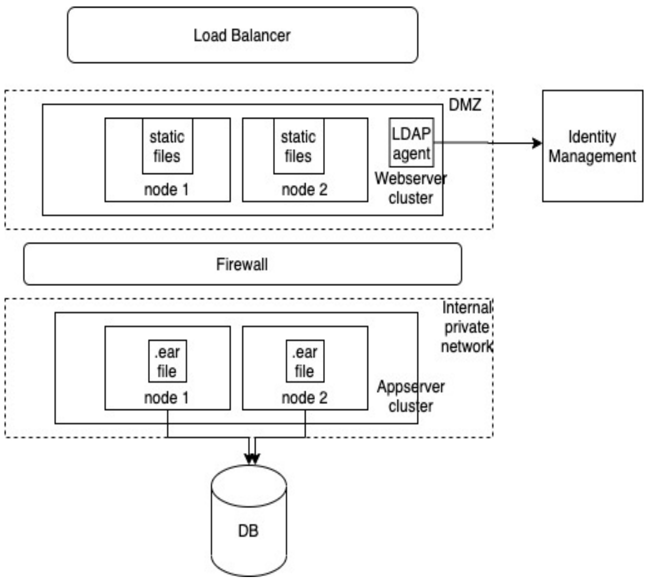

Fig. 1. Deployment architecture for a monolithic application in the local datacenter.

II. ZERO DOWNTIME BASED DEPLOYMENT STRATEGIES

There are 3 major ways by which a zero-downtime deployment can be achieved namely (1) Blue Green Deployment (2) Canary Deployment (3) Rolling deployment. There are various pros and cons for each of these techniques. Sections below explain various techniques in detail.

A. Blue Green Deployment

Blue green deployment methodology involves replacing the container with an older image with that of a newer image without causing any downtime [2]. This can be achieved by various design techniques in which the micro service running in a container with older image (Blue image) is replaced with the container having the newer image (Green image). This can be easily visualized from the Fig 2 where the traffic from the DNS router is routed to Blue image initially while the Green image is being deployed [6]. Upon the successful deployment the traffic is routed to Green instead of Blue. This strategy is useful in case of major functionality rollouts in the cloud where all the user base can be routed from old application version to the newer version. This is well suited for cloud infrastructures with container-based deployments and less suitable for data center-based deployments.

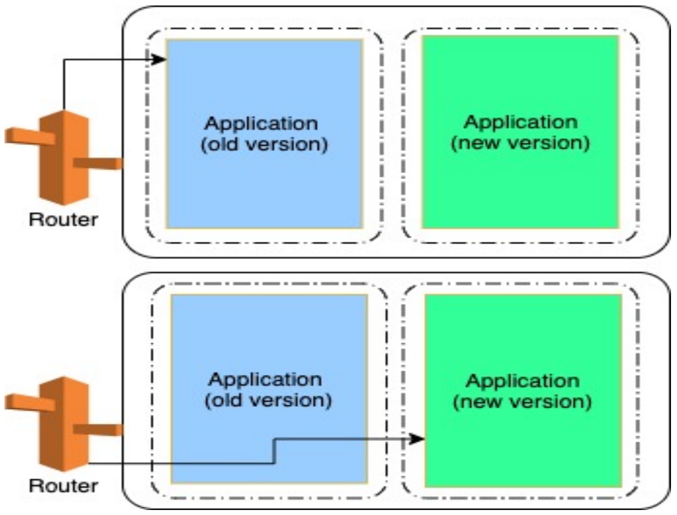
Application (old version)
Application (new version)
Router
Application (old version)
Application (new version)
Router

Fig. 2. A typical Blue Green deployment diagram

B. Canary Deployment

Blue green deployment methodology can be performed where all the traffic is shifted from Blue version to Green version at once or it can be done even in a phased approach. This model where the traffic is routed in a phased model is called Canary style of deployment. The phased approach can be customized depending on the application needs. It can be implemented by switching the Blue version to Green to certain user population based on the type of the user or the privilege they have (or) it can be based on the geo-location of the user. For instance, the newer version may be made available to users from certain territory or region rather than rolling out the change to entire global audience. Once the green version is approved by the limited users, it is then rolled off for all the users. This strategy can be visualized from the Fig 3 below.

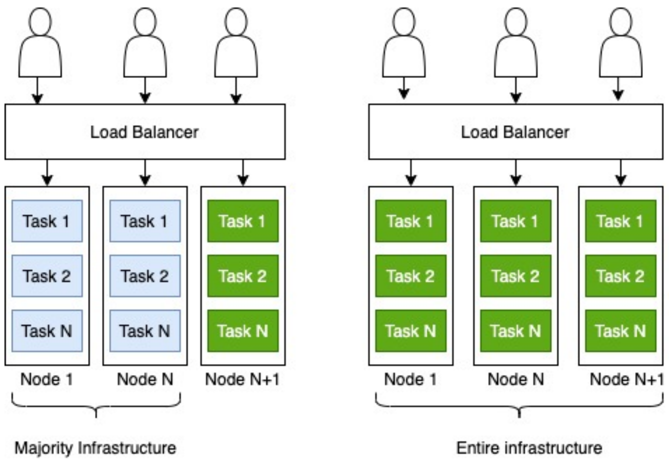

Fig. 3. Blue Green deployment in Canary Style

C. *Rolling Deployment*

Rolling deployment is more suitable for achieving the zero downtime in case of data center-based applications. This involves the manual intervention from the system administrators where they stagger the changes so that the patches are applied to certain server nodes while other nodes run the older version. Rolling deployments work well only when the organizations have enough spare capacity to roll out the newer versions without impacting the performance [15]. It may not be suitable where a major patch needs to be delivered to all the users at once.

Table1 given below summarizes the 3 zero downtime deployment strategies.

TABLE I. SUMMARY OF ZERO DOWNTIME DEPLOYMENT TECHNIQUES

Rolling Deployment	***Blue-Green Deployment***	***Canary Deployment***
Suitable for data centers.	Suitable for containers running in public cloud infrastructure	Suitable for both
Can be used for smaller patches	Can be used for major patches	Used for minor patches
Can be used for limited user routing at once	Can be used for entire user base routing	Can be used for limited user routing at once
Simple webapps are suitable	Complex enterprise and high traffic scenarios can be supported [14]	Suitable for minor changes for simple as well as complex apps.

III. RELATED WORK

Blue Green deployment can be achieved in multiple ways. Depending on the application needs and use case in hand, right strategy needs to be picked up. To explain the different strategies involved and compare the pros and cons of each, we have used Java based spring boot micro services. We have used AWS as the public cloud platform to perform various deployment strategies [8] [16]. We have used Terraform as the infrastructure as code (IAC) layer. We have used Mongo DB as the backend database needed for this research work.

D. *DNS Swap Technique for BG Deployment*

In this technique the DNS routing is changed to point to the Green version when the infrastructure rather than the older Blue version. For this the entire backend infrastructure and the load balancer need to be recreated for the green version. When the traffic needs to be swapped, the DNS router configuration would be changed to the newer version of the infrastructure, so that the newer version of services are delivered to the users. This technique can be visualized as shown in Fig 4 below. In here AWS Route 53 DNS is reconfigured to route the traffic to newer ALB.

To simulate this pattern, we have used 3 spring boot micro service MS1, MS2 and MS3. These micro services were dockerized and the images were uploaded to Docker hub as I11, I21 and I31. These are the initial version of the services which can be treated as the Blue versions. We used AWS Fargate based ECS (Elastic Container Service) cluster C1 as the hosting mechanism to run the docker containers. We used AWS application load balancer ALB with listeners for the service context paths configured to the target groups which point to the ports exposed for the ECS services.

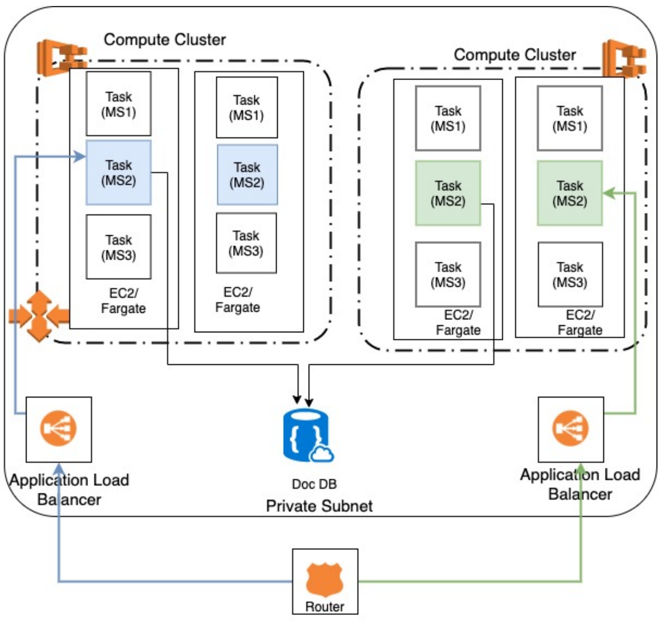

Fig. 4. DNS Swap Technique for Blue Green deployment

We used Terraform for standing up the AWS infrastructure. Once the ECS cluster was up and running the tasks, we introduced a newer version of docker images I12, I22, I32. These are newer versions which can be considered as the Green versions. Now in order to switch from Blue version to Green versions, we duplicated the ECS cluster C2 which hosts the newer version of micro services MS1, MS2 and MS3 using the newer images- I12, I22, I32 from the Docker hub along with the new load balancer ALB2. During this time, we had both versions of the services running. We then switched the DNS to point to newer ALB which connects to the newer ECS cluster C2 via the new load balancer ALB2. We observed that the traffic switched from Blue version to Green version after some time as the DNS takes time to swap the ALB. The swap time (SWT) is the sum of startup time for the new service (STT) and the DNS time to live (TTL) and the ALB health check time interval to start routing the traffic (HCT).

$$SWT = STT + TTL + HCT \quad (1)$$

We repeated the experiment for 3 runs and noted the total swap times in each run. We found the swap time to be as 243 seconds for a TTL of 60 seconds when each step is performed sequentially. When we repeated the experiment by working on duplicate ECS and ALB setup first and then the DNS swap, we observed that we could gain approximately 18 seconds which was the time needed for ALB health check. Hence total swap time in this case ranges from –

$$SWT : [(STT + TTL), (STT + TTL + HCT)] \quad (2)$$

E. Load Balancer Swap Technique for BG Deployment

In this technique the DNS routing is kept constant, but the load balancer is updated to point to a newer version [3] of services (green version). This can be visualized using the Fig 5 given below.

To simulate this pattern, we have used the same 3 spring boot micro service MS1, MS2 and MS3 and the same setup as described in the earlier section. When the Blue version of services I11, I21, I31 are up and running, a newer version of services I22, I22, I32 are stood up on a newer version of ECS cluster C2. But in this case, rather than the DNS swap, ALB configuration is changed to route to the newer Green version of the services. This technique needs configurations to adjust the scale up and scale down rules at the ALB level such that the Blue tasks are scaled down and Green tasks are scaled up gradually. The total swap time (SWT_2) in this case is a sum of sum of startup time for the new service (STT_2) and the ALB health check time interval to start routing the traffic (HCT_2).

$$SWT_2 = STT_2 + HCT_2 \quad (3)$$

We repeated the experiment for 3 runs and noted the total swap times in each run. We found the average to be as 186 seconds. This technique seems to be more effective than the DNS swap technique in terms of performance. But it calls for extra time needed to tune the scaling configurations of ALB.

F. Application Service Swap Technique for BG Deployment

In this technique the DNS routing is kept constant, load balancer routing configurations are kept constant but the docker image is updated to a newer version in the task definition. This can be visualized from Fig 5 given below.

To simulate this pattern, we have used the same 3 spring boot micro service MS1, MS2 and MS3 and the same setup as described in earlier section. When the Blue version of services I11, I21, I31 are up and running, a newer version of services I22, I22, I32 are uploaded to Docker hub. The ECS task definition is updated to pick up the newer version of docker images. This technique has the least configuration changes. The total swap time (SWT3) in this case is a sum of sum of ECS response time needed to react to the new service definition supplied (ERT3) and startup time for the new service (STT3) and the ALB health check time interval to start routing the traffic (HCT3).

$$SWT_3 = ERT_3 + HCT_3 \quad (4)$$

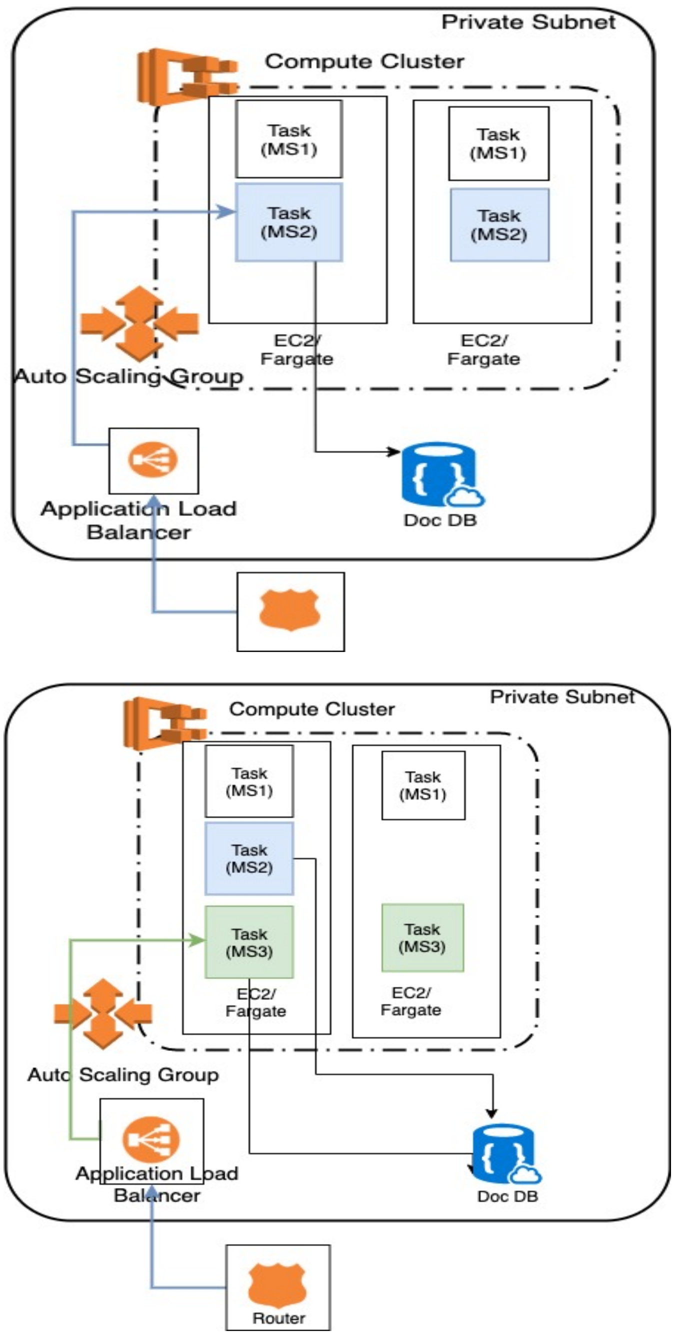

Fig. 5. Load Balancer Swap Technique for Blue Green deployment

We repeated the experiment for 3 runs and noted the total swap times in each run. We found the average to be as 201 seconds. This technique seems to be the least complicated approach, but we observed Canary style of deployment is not possible in this design.

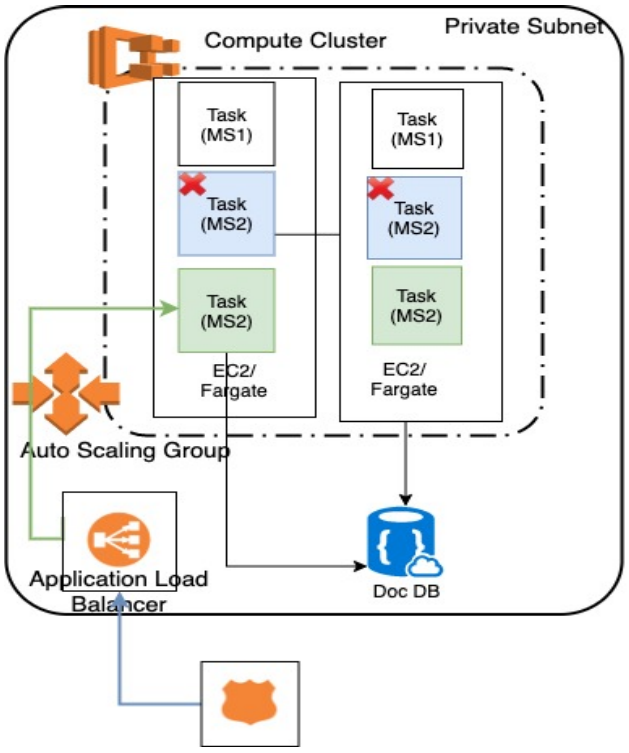

Fig. 6. Application service Swap Technique for Blue Green deployment

IV. Conclusion

Based on the research work above we observed that there are multiple ways of achieving the zero downtime-based deployments in the cloud infrastructure. We found that DNS swap technique is most simple way of performing the BG deployments. But we found that it is not cost effective as it needs entire infrastructure to be duplicated. Also, the swap time and roll back time is higher due to the additional DNS time to live (TTL) parameter. On the other hand, the load balancer swap technique was found to be better performing in terms of swap time than the previous approach. However, it needed constant adjustment to ALB configuration to scale down the Blue version and scale up the Green version. Finally, the application swap service was found to be better in terms of swap time when compared to DNS swap technique but lesser when compared to ALB swap technique. However, this technique has benefit of lesser scaling adjustments compared to ALB swap technique. It was also noted that this technique has a shortfall of not being able to support the canary style of deployment. It can be concluded that a right deployment strategy needs to be employed in the public cloud platform based on the application functionality being supported. Future scope of research includes working on scenarios involving Blue Green deployment strategies for the microservice based containers in the public cloud platform where the patch being applied has a dependency on the database layer as well. This would involve research around the containers designed in a way such that the new code is compatible with old DB schema or vice versa where old image is supportive of new DB schema changes.

References

[1] Chen L. Continuous delivery: Huge benefits, but challenges too. IEEE Software. 2015 Mar;32(2):50-4.

[2] Martin Fowler, "BlueGreenDeployment", 2010, Internet URL: https://martinfowler.com/bliki/BlueGreenDeployment.html

[3] B. Yang, A. Sailer, S. Jain, A. E. Tomala-Reyes, M. Singh and A. Ramnath, "Service Discovery Based Blue-Green Deployment Technique in Cloud Native Environments," 2018 IEEE International Conference on Services Computing (SCC), San Francisco, CA, 2018, pp. 185-192, doi: 10.1109/SCC.2018.00031.

[4] Lu, Hsin-Ke & Lin, Peng-Chun & Huang, Pin-Chia & Yuan, An. (2017). Deployment and Evaluation of a Continues Integration Process in Agile Development. Journal of Advances in Information Technology. 8. 203-209. 10.12720/jait.8.4.203-209.

[5] Lai, Sen-Tarng & Leu, Fang-Yie. (2015). Applying Continuous Integration for Reducing Web Applications Development Risks. 386-391. 10.1109/BWCCA.2015.54.

[6] Buzachis, Alina & Galletta, Antonino & Celesti, Antonio & Carnevale, Lorenzo & Villari, Massimo. (2019). Towards Osmotic Computing: a Blue-Green Strategy for the Fast Re-Deployment of Microservices. 1-6. 10.1109/ISCC47284.2019.8969621.

[7] Celesti, Antonio & Mulfari, Davide & Galletta, Antonino & Fazio, Maria & Carnevale, Lorenzo & Villari, Massimo. (2019). A study on container virtualization for guarantee quality of service in Cloud-of-Things. Future Generation Computer Systems. 10.1016/j.future.2019.03.055.

[8] Yang, Bo & Sailer, Anca & Mohindra, Ajay. (2020). Survey and Evaluation of Blue-Green Deployment Techniques in Cloud Native Environments. 10.1007/978-3-030-45989-5_6.

[9] Chaitanya K Rudrabhatla. (2018) A Systematic Study of Micro Service Architecture Evolution and their Deployment Patterns. International Journal of Computer Applications 182(29):18-24, November. doi: 10.5120/ijca2018918153

[10] Malik, Sahil. (2019). Continuous Deployment: Surveying the Options. 10.1007/978-1-4842-4409-8_5.

[11] Shirinbab, Sogand & Lundberg, Lars & Casalicchio, Emiliano. (2020). Performance evaluation of containers and virtual machines when running Cassandra workload concurrently. Concurrency and Computation: Practice and Experience. 10.1002/cpe.5693.

[12] V, Dr. (2020). Constraints Mitigation in Cognitive Radio Networks Using Cloud Computing. Journal of Trends in Computer Science and Smart Technology. 2. 1-14. 10.36548/jtcsst.2020.1.001.

[13] Ochei, L., Petrovski, A. & Bass, J. Optimal deployment of components of cloud-hosted application for guaranteeing multitenancy isolation. J Cloud Comp 8, 1 (2019). https://doi.org/10.1186/s13677-018-0124-5

[14] Giannakopoulos, I., Konstantinou, I., Tsoumakos, D. *et al.* Cloud application deployment with transient failure recovery. *J Cloud Comp* **7,** 11 (2018). https://doi.org/10.1186/s13677-018-0112-9

[15] Soni, Mitesh. (2015). End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery. 85-89. 10.1109/CCEM.2015.29.

[16] Tavbulatova, Z & Zhigalov, K & Kuznetsova, S & Patrusova, A. (2020). Types of cloud deployment. Journal of Physics: Conference Series. 1582. 012085. 10.1088/1742-6596/1582/1/012085.

[17] Garbarino, Ernesto. (2019). Beginning Kubernetes on the Google Cloud Platform: A Guide to Automating Application Deployment, Scaling, and Management. 10.1007/978-1-4842-5491-2.