Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes

Artur Cepuc
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
arturcepuc@gmail.com

Robert Botez
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Robert.Botez@com.utcluj.ro

Ovidiu Craciun
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
ovidiu.craciun94@gmail.com

Iustin-Alexandru Ivanciu
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Iustin.Ivanciu@com.utcluj.ro

Virgil Dobrota
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Virgil.Dobrota@com.utcluj.ro

2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet) | 978-0-7381-1265-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ROEDUNET51892.2020.9324857

Abstract— **Nowadays, cloud computing has become the go to solution for most enterprises. This has led to the introduction of DevOps techniques in which developers work closely with network engineers in order to ensure fast and reliable deployment of their applications. This paper presents an entire automated pipeline, starting with detecting changes in the Java-based web application source code, creating new resources in the Kubernetes cluster to host this new version and finally deploying the containerized application in AWS. The solution follows DevOps best practices and relies on Jenkins for the Continuous Integration stage. The novelty herein is that we used Ansible for Continuous Deployment thus increasing the scalability and overall ease of use. The solution ensures zero downtime and proves fast, even though it combines six different technologies and requires very few computational resources.**

Keywords— Ansible; Amazon Web Services; Continuous Integration/ Continuous Deployment; DevOps; Docker; Jenkins; Kubernetes.

I. INTRODUCTION

Cloud computing has evolved into one of the most ubiquitous concepts in the IT industry. Its success lies in on-demand services rather than deploying a whole infrastracture which would bring additional costs like purchasing and maintaining the equipment, paying the employees and so on. The National Institue of Standards and Technology (NIST) classifies whether or not a service is a cloud service according to the following characteristics [1]: broad network access, rapid elasticity, measured service, on-demand self-service and resource pooling. The cloud services are available in three different models [2]:

- Infrastructure as a service (IaaS): the customer is provisioned with compute, storage and networking in order to manage operating systems (OS), middleware, runtime, data and applications.
- Platform as a service (Paas): in this model, the customer is additionally provisioned with OS, middleware and runtime for managing data and applications. This model is similar to the concepts of serverless computing and is widely used by developers to create customized software without worrying about the underlying system.
- Software as a service (Saas): compared to the previous models, SaaS delivers applications that are managed by a third-party provider directly to the customer.

These service models lie on top of several deployment models of the Cloud: Private Cloud, Public Cloud and Hybrid Cloud. For this paper we chose AWS, which is a public cloud solution, for its wide variety of cloud services.

In order to tackle the slow delivery of services to customers as in Traditional IT approaches, many organizations adopt DevOps and Continuous Integration and Continuous Deployment (CI/CD) techniques wich better meet customer needs. The gap between developers and operations teams could be filled by DevOps, which combines them into one unit, thanks to the high flexibility and scalability of the infrastructure in the Cloud [3]. Moreover, to facilitate the automation of the entire software delivery process, DevOps practices involve three steps: Continuous Integration, Continuous Delivery and Continuous Deployment.

In a nutshell, the CI/CD central approach is represented by automation. This eliminates the need for human checks but it does not come without security risks. According to [4], the application development phases include the following: coding, building, testing, integrating, infrastructure provisioning, troubleshooting, configuration management, setting up runtime environments, and also deploying application in different environments. There are several CI/CD tools other than Jenkins and Ansible (that are used in this paper); some of which will be presented below.

JetBrains TeamCity is a user-friendly and a powerful Continuous Integration and Continuous Deployment server that works out of the box [5]. It can run parallel builds simultaneously on different environments and platforms. Some of the main advantages of TeamCity are optimizing the code integration, reviewing on-the-fly test results reporting, using intelligent tests re-ordering, running code coverage, finding duplicates, customizing statistics on build duration, code quality, success rate and custom metrics. TeamCity contains an integrated builds artifactory repository. The resulting artifacts are stored either on an external storage or on the server-accessible file system. After the build is finished, the artifacts from the build checkout directory are found by TeamCity according to the artifact path. The files are then published to the TeamCity server, where they are available for download through the web UI.

Bamboo Atlassian is a CI tool which is suitable for multiple programming languages and other popular technologies like Docker, Amazon S3 and AWS CodeDeploy. The user can choose from a big variety of tasks for both builds and deployment projects. Bamboo represents a CI server which can be connected with other Atlassian products and can be used to automate the release for a software application. According to [6], Bamboo uses the concept of a plan with tasks and jobs in order to configure the actions in the workflow.

TABLE I. Comparison of Integration Tools

	TeamCity	Bamboo	Jenkins
OpenSource	No	No	Yes
Ease of use	5/5	4/5	3/5
Built-In Features	5/5	4/5	2/5
Integrations	338	221	1447
Support	5/5	5/5	4/5
Cloud Support	Yes	Yes	Yes
Pricing	From $299	From $888	Free

Puppet is an open source management tool which relies on a client-server architecture: clients, called Puppet Agents, make periodical requests to the server, called Puppet Master, asking for the desired state and respond with status reports [7]. It allows the quick upgrade, creation and deletion of nodes which are organized in clusters. Each cluster may have multiple masters. The main advantages of using Puppet refer to the compliance across numerous environments and the dedicated web interface and reporting tools. However, Puppet may prove difficult to learn for new users and does not scale very well.

Chef is another automation tool for both system and Cloud infrastructures [7]. It allows the automated installation of applications on bare metal, virtual machines and container-based cloud. Chef introduces new concepts such as cookbooks and recipes in order to provide functionality. The configurations are deployed from the Chef Server to the Chef Nodes by means of the Chef Workstation around an infrastructure-as-a-code (IaaC) model. Unlike Puppet, Chef can scale up to tens of thousands of hosts and the cookbooks specifying the desired state of the node always generate the same result, regardless of the number of times they were executed. While there are many predefined cookbooks available online, the documentation is huge and at times difficult to follow.

The motivation of this paper was fueled by our experience with the CI/CD tools and DevOps practices described in the previous paragraphs. While most enterprises are considering the switch from traditional IT techniques to DevOps, the process will be lengthy and costly. Moreoever, since the main goal of such companies is to deliver a stable and reliable product, there is not much room for experimenting with new architectures and deployments. However, we decided to stray away from the proverbial well-trodden path and try out some novel combinations of software tools for both CI and CD. The remainder of this paper is organized as follows. Section 2 presents an overview regarding the working principle of Jenkins and the Ansible architecture. The implementation of the solution is discussed in Section 3, followed by experimental results in Section 4. The paper ends with conclusions and future work.

II. Overview of Jenkins and Ansible

Jenkins is an open-source automation tool written in Java with different plugins developed for the purpose of continuous integration [8]. It is widely used for building and testing software projects continuously, facilitating the integration of changes in the project and obtaining a new build by developers. Jenkins integrates with a large number of testing and deployment technologies (such as Selenium, Kubernetes, etc.) offering an easy way to configure an environment of continuous integration or delivery (CI/CD) for almost any combination of programming languages or repository hosting services using pipelines. It can work in master-slave architecture and in this type of architecture a master node can execute jobs on multiple slaves running on different platforms. Although Jenkins does not eliminate the need of creation of scripts for individual steps, it brings a faster and more robust way to integrate the entire chain of tools for building, testing and deployment. An example of continuous integration with a Jenkins server is presented in Fig. 1. The developers commit changes to the source code to a source code management (SCM) tool such as GitHub. The changes are verified and validated (or not) by the Jenkins server and the builds with the test results are fed back to the developers.

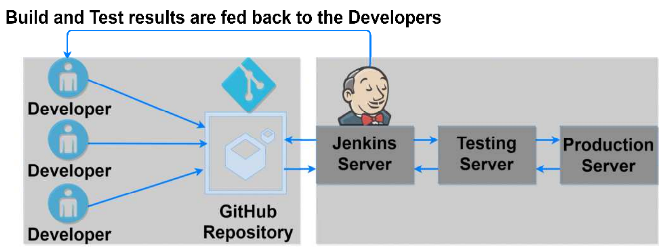


Fig. 1. Jenkins Architecture

Ansible is an automation tool developed by RedHat that offers a simple yet powerful cross-platform computer support [9]. This is mainly intended for deploying applications, workstation and server upgrades, cloud provisioning, configuration management and in-service orchestration. Ansible can be used to model the entire IT infrastructure because it focuses on the way different systems interact rather than managing them one by one. The deployment is easy and straightforward with no agents that need to be installed. The configurations are represented as playbooks which can be deployed on one or several machines at the same time. Probably the most important advantage of using Ansible is that the playbooks use YAML to describe the automation jobs as naturally as possible without having to worry much about syntax. Ansible connects to the desired nodes and executes small pieces of code called modules on them, using SSH. When finished, these modules are removed. The modules can be stored on any machine and this means that the architecture is very simple, with no special server, daemon or database needed. Hosts managed by Ansible are grouped in inventories which can be user defined or drawn from different cloud environments. While there are many predefined plugins, users can further extend the functionality of Ansible by adding new ones using the dedicated APIs. An overview of the components described in this paragraph is shown in Fig. 2.

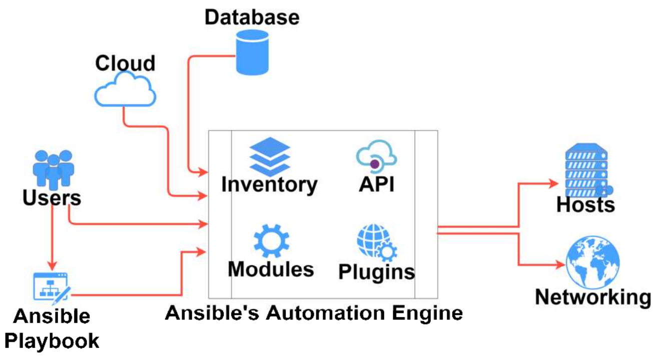


Fig. 2. Ansible Architecture

III. Implementation

A. Solution Architecture

This paper presents an automated pipeline for the CI/CD of a web application in AWS. The proposed architecture, illustrated in Fig. 3, consists of two VPCs, one for the Ansible, Jenkins and Kubernetes Management Servers, and the second for hosting the Kubernetes Cluster.

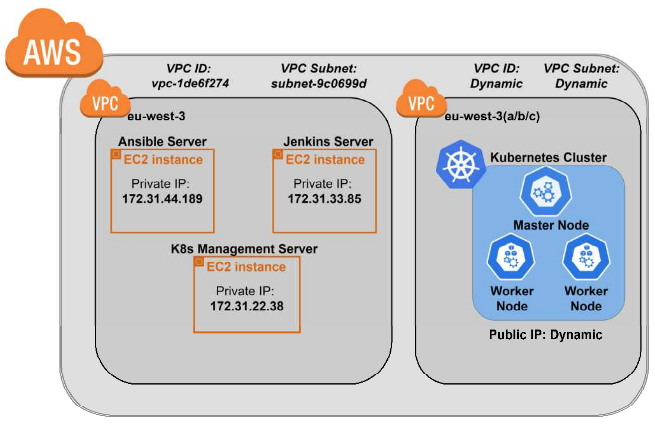


Fig. 3. The proposed architecture

The web application was developed using Java and an Apache Maven archetype which helps describe the software project, its dependencies and other external components in a Project Object Model (POM) .xml file. Once created, the source code of the application was loaded in GitHub.

In order to perform CI, a Jenkins server was installed and configured on an EC2 AWS instance. By default, Jenkins uses TCP port 8080, therefor it is mandatory to allow incoming connections on this port. This was achieved by creating a new security group which allows us to define different security rules based on protocol, port range and IP source address. One last step in configuring Jenkins is the installation of the GitHub and Apache Maven plugins to ensure the interoperability of all three servers.

Next, the official Tomcat server image was downloaded from the DockerHub. This image was used as a primary layer in Dockerfile and was customized by adding several resources, including the Web Application Resource (WAR) file of the application. The WAR file is used by Maven WAR Plugin for collecting all dependencies, classes and resources of the application. The new customized Docker image, called *tomcat8*, was pushed to DockerHub to be used from now on.

For further automation of the CI/CD pipeline, Ansible was added in another EC2 instance. The installation and configuration steps are similar to those used for Jenkins. In our approach, Ansible performs CD: the devised playbooks help create a new Docker image every time the web application changes and then upload it to DockerHub.

A Kubernetes cluster was created for hosting the containerized application in the second VPC, in Fig. 3. The cluster is composed of one master node and two worker nodes.

The final step was to monitor the entire CI/CD pipeline: CloudWatch for the AWS resources, dedicated processes for the web application on the Tomcat server and the Email Extension Plugin for Jenkins notifications.

B. CI/CD Process

Fig. 4 presents the entire automated pipeline, the implementation of which was described in the previous section. CI/CD concepts were followed using DevOps best practices and tools. The Jenkins server is responsible for the CI process, while the CD process was orchestrated by Ansible. While most pipelines use Jenkins for both CI and CD, for this implementation we decided to integrate Ansible due to its ability to execute commands directly on the target hosts, in this case, the Kubernetes cluster. Using Ansible also ensures scalability as a single playbook can be executed on any number of hosts. Jenkins on the other hand needs a separate SSH connection and specialized plugins for each target. Also, Ansible offers the possibility to revert to the last working configuration in case of failure.

Any change of the project source code in the GitHub repository triggers the CI job. This was done by selecting the Poll SCM setting in Jenkins which uses a Java plugin based on Timestamp. Next, the Jenkins job clones the GitHub repository in the local workspace. The code from the local workspace is then compiled and packaged using the Maven plugin in a .war file. This .war file is securely transferred from Jenkins to Ansible with SSH. Ansible now executes the first playbook which pulls the custom Tomcat image from DockerHub, adds the .war file in the specific folder, then builds the new image and pushes it back to DockerHub.

Once this CI process is completed successfully, Jenkins automatically triggers the CD part of the pipeline which in turn executes two other Ansible playbooks: the first one creates the Kubernetes specific deployments, while the second one creates Kubernetes services. The configuration files for the Kubernetes cluster are written in YAML and stored in another GitHub repository. In order to prevent downtime, a rolling-update strategy was employed.

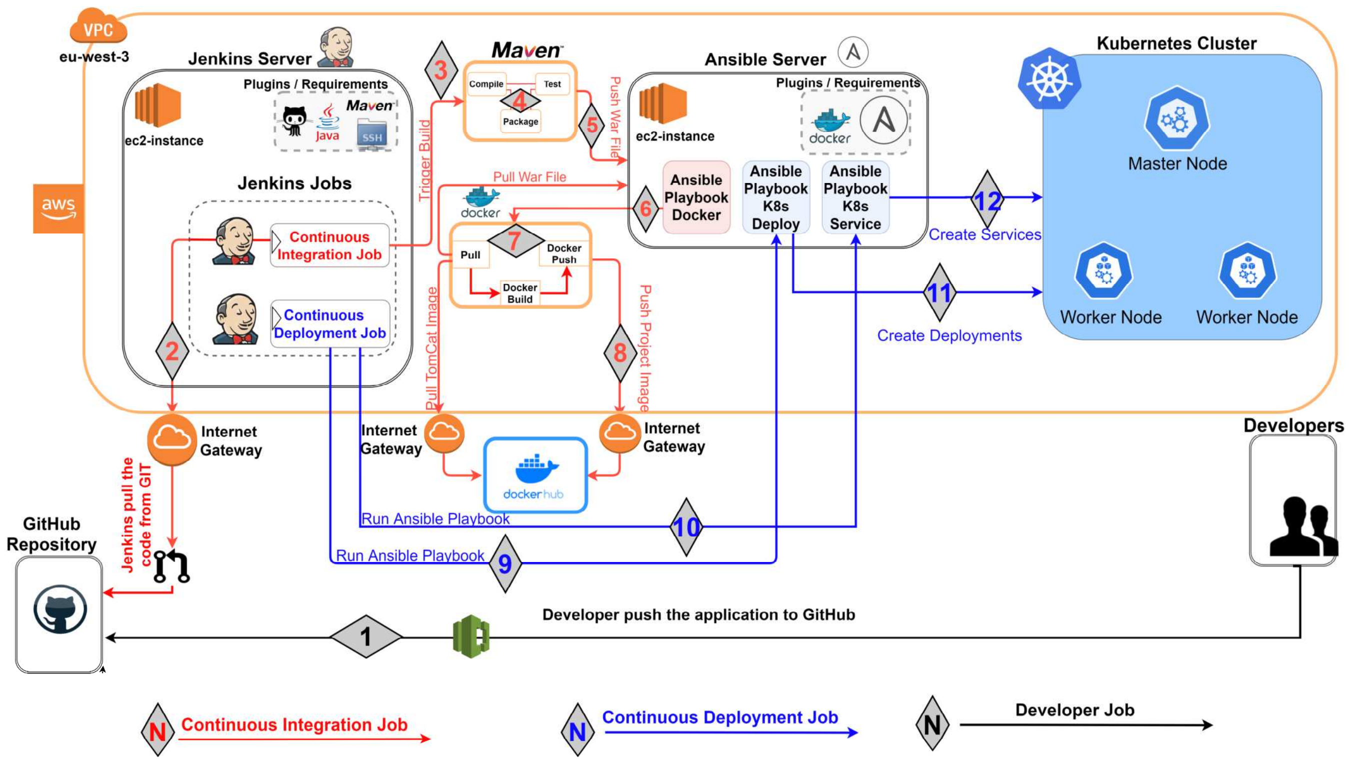


Fig. 4. The entire automated pipeline

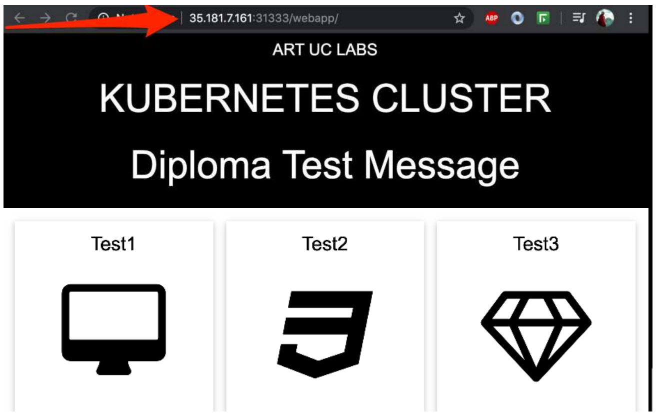
IV. Experimental Results

This section presents the results obtained in every step of the pipeline. First, a Kubernetes cluster was created in AWS, having one master and two worker nodes. All three nodes are of type t2.micro and are located in the eu-west-3a region. This process took about 10 minutes, as illustrated in Fig. 5.

```
INSTANCE GROUPS
NAME     ROLE MACHINETYPE MIN MAX SUBNETS
master-eu-west-3a Master t2.micro 1 1      eu-west-3a
nodes Node t2.micro 2 2                    eu-west-3a
NODE STATUS
NAME ROLE READY
ip-172-20-47-224.eu-west-3.compute.internal node True
ip-172-20-48-24.eu-west-3.compute.internal node True
ip-172-20-50-239.eu-west-3.compute.internal master True
Your cluster k8s.arthur1cp.eu is ready
real 10m9.641s
user 0m5.312s
sys 0m0.665s
```

Fig. 5. Kubernetes cluster

The containers of the web application are hosted on two pods and are accessible via a Kubernetes resource, namely a LoadBalancer service. The first version of the web application is thus exposed to the Internet, as can be observed in Fig. 6.



35.181.7.161:31333/webapp/

ART UC LABS

KUBERNETES CLUSTER

Diploma Test Message

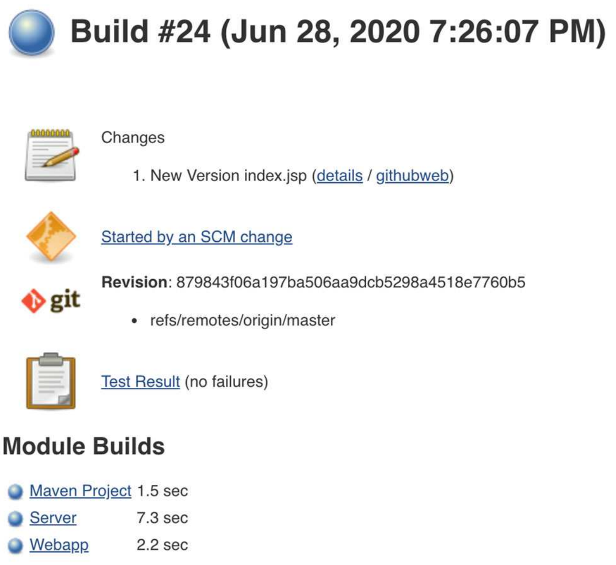
Test1

Test2

Test3

Fig. 6. The first version of the web application

The next step in the pipeline was to make a change in the application and push the new version to GitHub. This change was automatically detected by Jenkins which starts deploying Kubernetes resources for hosting the new application (see Fig. 7):



Build #24 (Jun 28, 2020 7:26:07 PM)

Changes

1. New Version index.jsp (details / githubweb)

Started by an SCM change

Revision: 879843f06a197ba506aa9dcb5298a4518e7760b5

- refs/remotes/origin/master

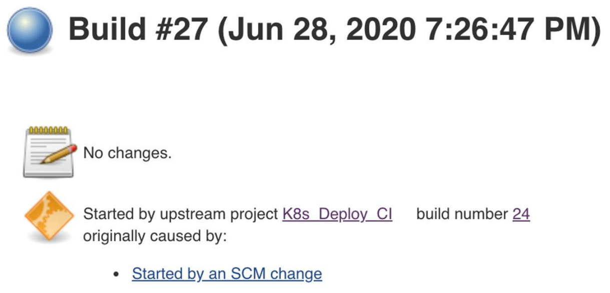
Test Result (no failures)

Module Builds

Maven Project 1.5 sec
Server 7.3 sec
Webapp 2.2 sec

Fig. 7. Detection by Jenkins of the changes in the source code

The successful completion of this process means the end of the CI part of the pipeline. Next, another Jenkins job starts implementing the CD part, as in Fig. 8.



Build #27 (Jun 28, 2020 7:26:47 PM)

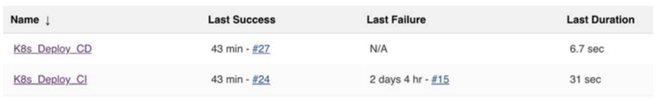
No changes.

Started by upstream project K8s_Deploy_CI build number 24
originally caused by:

- Started by an SCM change

Fig. 8. Jenkins job responsible with the implementation of CD part

Fig. 9 shows that the two processes take 37.6 seconds and integrate a total of six different technologies running on AWS, DockerHub and GitHub.



Name ↓	Last Success	Last Failure	Last Duration
K8s_Deploy_CD	43 min - #27	N/A	6.7 sec
K8s_Deploy_CI	43 min - #24	2 days 4 hr - #15	31 sec

Fig. 9. The total time spent by processes

The implemented rolling update strategy means that the old pods are replaced one by one in order to reduce the downtime of the application. This process is shown in Fig. 10.

```
root@ip-172-20-63-148:/home/admin#   kubectl   get
pods
NAME READY STATUS RESTARTS AGE
art-deployment-5cdc86b88d-27162              0/1
ContainerCreating 0 1s
art-deployment-5cdc86b88d-8m8k5              0/1
ContainerCreating 0 1s
art-deployment-7d84d9b986-27d7z  1/1  Terminating  0
20m
art-deployment-7d84d9b986-2x7d4 1/1 Running 0 20m
```

Fig.10. Output of the `kubectl get pods` command.

While the application is still accessible at the same IP address and port number as before, the changes are easily visible (as in Fig. 11).

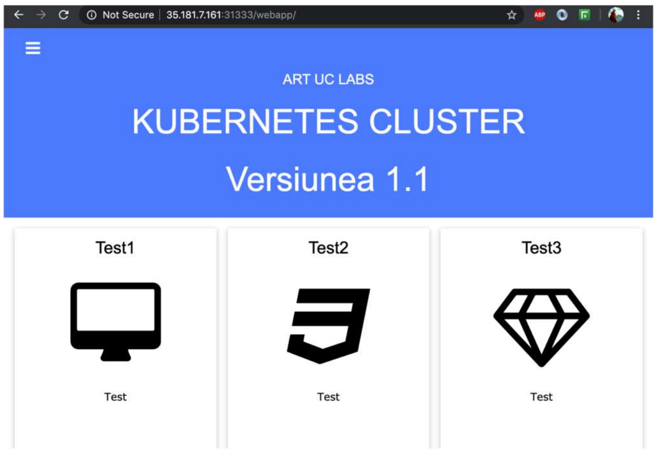


Fig. 11. The new version of the application

The Kubernetes deployment helps keep the same number of active pods. To test this, we stopped one of the worker nodes and then checked the availability of the application and the status of the pods. The age of the second pod (21 seconds) shows that it was recently created due to node failure (see Fig.12).

```
root@ip-172-20-63-148:/home/admin#   kubectl   get
pods
NAME READY STATUS RESTARTS AGE
art-deployment-5cdc86b88d-27162 1/1 Running 0 65m
art-deployment-5cdc86b88d-8m8k5 1/1 Running 0 65m
root@ip-172-20-63-148:/home/admin#   kubectl   get
pods
NAME READY STATUS RESTARTS AGE
art-deployment-5cdc86b88d-27162 1/1 Running 0 65m
art-deployment-5cdc86b88d-cx2dt 1/1 Running 0 21s
```

Fig.12. Output of the kubectl get pods command after a new pod was created

Monitoring the application was performed using CloudWatch. Fig. 13 presents the CPU utilization in real-time for each of the hosts running processes in the pipeline. All the values are below 10%, even when using instances with the least amount of resources available in AWS. The rise in CPU utilization of the Jenkins server corresponds to the launch of an automated deployment process. In case of exceeding the pre-allocated amount of CPU, the deployment resource can be configured to restart the overloaded pods.

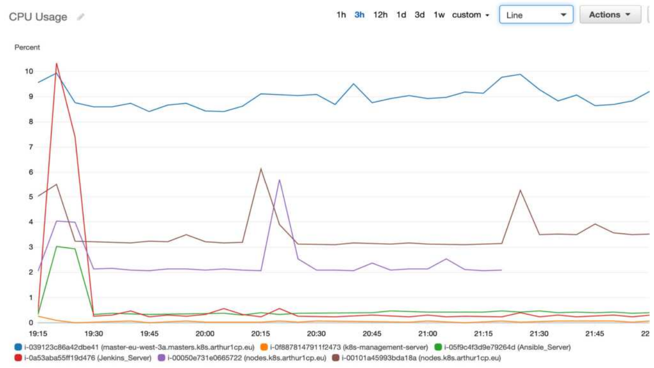


Fig. 13. CPU utilization of the pipeline components

Other statistics available in CloudWatch refer to incoming and outgoing network traffic for the instances hosting the Kubernetes cluster (Fig. 14). The lack of metrics for the worker node depicted in blue (output traffic) and green (input traffic) is due to the resiliency test described previously.

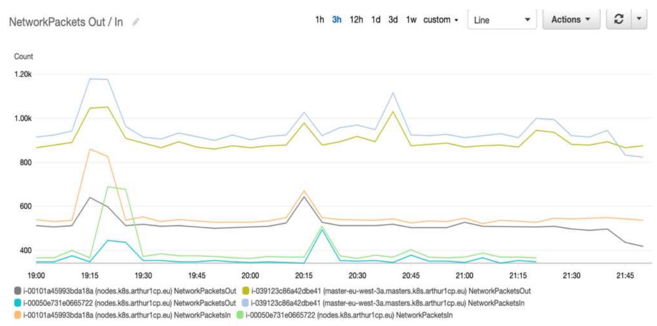
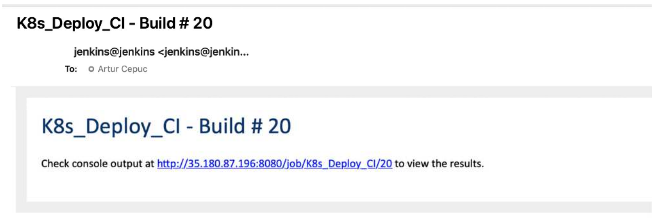


Fig. 14. Network traffic for Kubernetes cluster

The last component of the monitoring system is responsible for detecting a Jenkins job failure. In this case, an alert email is automatically sent by the Jenkins server (see Fig. 15).



K8s_Deploy_CI - Build # 20

jenkins@jenkins <jenkins@jenkin...

To: Artur Cepuc

K8s_Deploy_CI - Build # 20

Check console output at http://35.180.87.196:8080/job/K8s_Deploy_CI/20 to view the results.

Fig. 15. E-mail alert in case of Jenkins job failure

V. Conclusions and Future Work

This paper presents an automated CI/CD pipeline for deploying a Java-based web application in AWS. While the DevOps best practices were followed, we introduced an element of novelty by employing Ansible in the CD process. This allowed us to send commands directly to the Kubernetes cluster and to ensure better overall scalability. Experimental results showed that the proposed solution is reliable, having 0 seconds downtime, easily scalable and fast. Thus, any change in the source code of the application is automatically detected and triggers an entire chain of events, composed of six different technologies, in just 37.6 seconds. Any failure of a Jenkins job when deploying a new version of the application is detected and the system rolls back the latest stable version. CloudWatch was used to monitor resource consumption in AWS and highlighted the efficiency of the proposed solution, which works perfectly even on the least powerful machines offered by the Cloud. While especially tailored for a Java-based web application, the techniques and processes described herein are a good starting point for hosting 5G network core functions in Docker containers and managed by Kubernetes. This, in turn, enables the creation of network slices which help guarantee the best service for different types of applications: autonomous driving, telemedicine, smart cities, augmented reality and others.

Future work refers to implementing the same solution on several public and private clouds and devising an algorithm which selects the best cloud to deploy the web application on in terms of latency.

References

[1] "The NIST Definition of Cloud Computing", Computer Security Resource Center, NIST 2020, [Online], Available: https://csrc.nist.gov/publications/detail/sp/800-145/final.

[2] S. Watts, M. Raza, "SaaS vs PaaS vs IaaS: What's The Difference & How to Choose", BMC Software, 2019, [Online], Available: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/.

[3] C. Jackson, J. Gooley, A. Iliesiu and A. Malegaonkar, "Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide", Cisco Press, 2020.

[4] C. Geiger, D. Przytarski and S. Thullner, "Performance testing in continuous integration environments", Institute of Software Technology, University of Stuttgart, 2014.

[5] "Continuous integration with TeamCity", TeamCity 2020, [Online], Available: https://www.jetbrains.com/help/teamcity/continuous-integration-with-teamcity.html#ContinuousIntegrationwithTeamCity-BasicTeamCityconcepts.

[6] R. Varga, "Changing Dashboard build system to Bamboo", CERN Summer Student Program, No. CERN-STUDENTS-Note-2013-135, 2013, [Online], Available: https://cds.cern.ch/record/1596224.

[7] E. Luchian, C. Filip, A. B. Rus, I. Ivanciu and V. Dobrota, "Automation of the infrastructure and services for an openstack deployment using chef tool," 2016 15th RoEduNet Conference: Networking in Education and Research, Bucharest, 2016, pp. 1-5, doi: 10.1109/RoEduNet.2016.7753200.

[8] N. Seth and R. Khare, "ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development," 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), Chandigarh, 2015, pp. 1-6, doi: 10.1109/RAECS.2015.7453279.

[9] "Ansible Documentation', Ansible Project 2020, [Online], Available: https://docs.ansible.com.