EasyChair Preprint

№ 3231

Composable DevOps Architecture: The Need for Secure and Flexible Deployment

Parastou Moghaddam, Harkaran Singh, Karan Sharma,
Noha Elissawy and Jeong-Joo Park

April 22, 2020

Composable DevSecOps Architecture

The Need for Secure and Flexible Deployment

Parastou Moghaddam, Harkaran Singh, Karan Sharma, Noha Elissawy, and Jeong-joo Park
Volgenau School of Engineering - Department of Cyber Security Engineering
George Mason University - Fairfax, VA
Email: pmoghadd@gmu.edu, hsingh23@gmu.edu, ksharm11@gmu.edu, nelissaw@gmu.edu, jpark76@gmu.edu

Abstract*——By the end of 2020, 83% of enterprise IT operations will resort to cloud platforms [1]. Companies are choosing to transition to cloud environments due to many benefits associated with cloud services, including: affordability, security, scalability, resiliency, and agility. While cloud computing offers massive economies of scale, understanding cloud computing best practices in order to enhance security and avoid vendor lock-in has been an ongoing challenge for some companies. In this paper, the best approach in orchestrating a cloud-based DevSecOps pipeline is demonstrated.

Keywords——DevOps, CI/CD, GitHub, Kubernetes, Docker, Ansible, Prometheus, SonarQube, AWS.

I. INTRODUCTION

Companies seek the optimum approach to engineer Continuous Integration Continuous Development (CI/CD), and cloud-based solutions play a major role in facilitating the process. Cloud computing eliminates guesswork about the company's capacity needs as there is no need for upfront capital costs or predicting future work loads. Underutilized on-premise hardware resources can be replaced by on-demand resources provided by cloud service providers (CSPs). With only a few clicks, the process of scaling-in or scaling-out whether horizontally or vertically can be fully automated.

Beside negotiation and business-wise strategies that all companies need to follow, designing a composable architecture that incorporates applications that can be easily decoupled and transferred from one platform to another is a key factor in cloud migration. In other words, application compatibility plays a major role to reduce the risk of proprietary lock-in. In order to facilitate flexible deployment, open-source tools that can be easily decoupled from physical resources are chosen.

In this project a comprehensive analysis of creating an application by exclusively incorporating open-source tools to orchestrate a DevSecOps pipeline is illustrated. Continuous testing of the pipeline for security and functionality purposes is verified by performing smoke testing, security testing, and performance testing. In addition, the process of containerization and deployment of applications are explored to analyze scalability and maintain efficiency of resources. The goal is to highlight the benefits of using open-source tools to optimize cost, prevent vendor lock-in, and secure the DevOps architecture in cloud environments.

II. MISSION REQUIREMENTS

A. Stakeholder

Companies strive to migrate to cloud-based technologies but for many the risk of vendor lock-in is a barrier hard to overcome.This project aims to use emerging cloud technologies to develop and document a process for securely setting up a DevSecOps pipeline. In addition, supplementary research is provided on the pipeline across cloud platforms to benefit organizations that are considering to migrate to cloud by proposing a composable and secure architecture.

B. Concept of Operations

The concept of operations (CONOPS) as shown in figure 1 demonstrates components that work together to engineer this system. Mattermost, an online chat service, is selected as a repository to be replicated to function as the main input of the pipeline. Amazon Web Services (AWS) platform and open-source

applications are utilized as enablers to orchestrate the pipeline. User interaction is achieved by smart devices with Internet connection, and users need to obtain credentials to gain access. A few constraints are set on the implementation of the project such as deadline, fully automated testing, autonomous deployment, and application containerization. The result of all elements functioning together would be a robust DevSecOps pipeline.

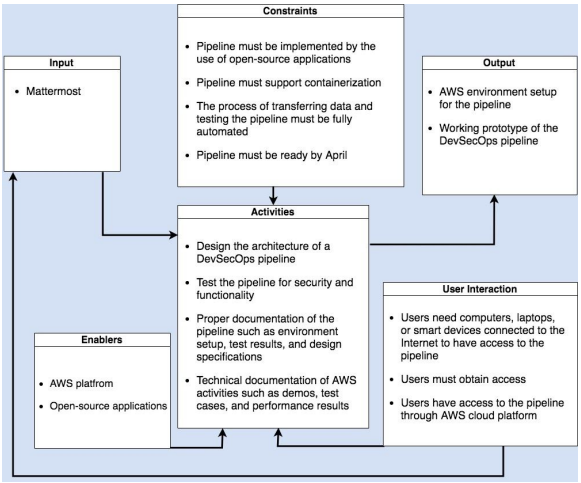

Figure 1: Concept of Operations

III. DESIGN REQUIREMENTS

Based on the specific constraints set for the pipeline, the architecture is designed. CI/CD is established by consistent and automated ways to build, package, test, and deploy applications. As shown in figure 2, the fabricated pipeline integrates the most popular open-source applications which are: GitHub, Jenkins, SonarQube, DockerHub, Kubernetes, Ansible, and Prometheus.

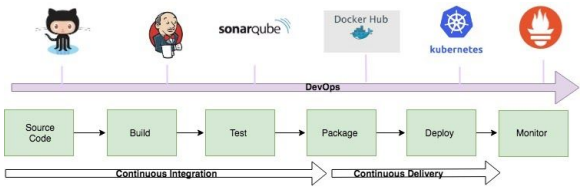

Figure 2: DevOps Project Architecture

A. *GitHub*

For source code management, the pipeline draws from GitHub services, which functions as a version control system. Any changes, new versions of code, or updates on commits can be viewed and managed from the GitHub dashboard, from there Github manages the source code, which in this case is Mattermost's open-source Web Server. Mattermost's open-source repository is cloned to utilize the source code in the pipeline. Any revisions to the source code shows up in the new Mattermost repository, and this allows revisions from anywhere and with ease, as GitHub changelogs shows commits and changes in code. GitHub is linked to a customized Jenkins CI server, which is set up to draw the code repository from GitHub and go through continuous integration.

B. *Jenkins*

Continuous integration is essential in building a DevSecOps pipeline, as it allows teams to integrate multiple stages. The DevSecOps pipeline is based on a Jenkins CI build server. The Build stage automates the DevSecOps process, by integrating various tools at the development, security, and operations stage. Jenkins enables testing and building onto the Mattermost source by utilizing different tools.

Continuous integration is achieved with the use of plugins, and every tool that is needed to integrate in order to complete the project depends on a plugin. The customized Jenkins Build has a tool for various stages, such as: Build, Version Control, Continuous Monitoring, Continuous Deployment, Configuration Management, and Continuous Testing. The Jenkins server configuration process allows a simple installation for plugins out of the box, and all of the tools selected for the pipeline come into touch with the Jenkins build server.

C. *SonarQube*

Performing Continuous Testing (CT) is crucial to ensure the quality of the code deployment and the security of the pipeline. SonarQube is a continuous code quality management tool that can perform static analysis in more than 25 programming languages. It incorporates pattern matching approaches by integrating OWASP, CWE, WASC, SANS and CERT security standards to detect vulnerabilities, refactoring opportunities, and code smells [2]. Different jobs in Jenkins are declared to perform tests on these source codes, and each of these jobs are customized based on the languages used in the source codes. As the figure 2 shows, on the SonarQube server different projects such as appTest, server, and

app are created to receive data from the Jenkins server.

Mattermost repositories in GitHub are fetched by Jenkins and transferred to SonarQube server to be scanned for potential code flaws. Source codes in Mattermost are mainly written in JSON, Go, and Ruby. As illustrated in figure 3, different unit tests are created, and a few code smells and minor vulnerabilities are detected. One common warning among these codes is the detection of unused function parameters, but all unit tests have passed the security testing.

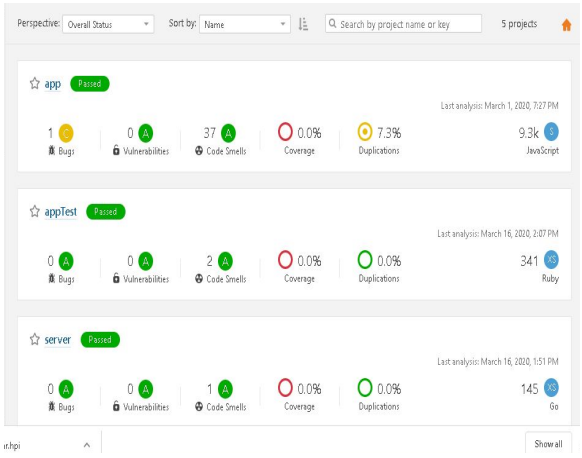

Figure 3: SonarQube Scan Result

D. Docker

Containerization is the process of distributing and deploying applications in a modular fashion. Docker is a software containerization platform that standardizes the application deployment in enterprise environments (qa, dev, staging, production). Docker accomplishes this by packaging source codes and its dependencies into standardized, isolated, and lightweight process environments called containers. Mattermost application is used to test the prototype of the pipeline through containerization using Docker.

Docker is the core component of the pipeline in distributing container deployments that provide scalability and management. Images are typically built from a Dockerfile, a text file that specifies the variables and components that are included in the build steps of a container. These images are stored in a registry, most notably Docker Hub, from where they can be downloaded and run on various instances. For the Mattermost-server application, there are three Docker images: web, app, and db. Each image serves one aspect of the application. For instance, the 'web' image is responsible for handling HTTP(s) requests from incoming users, while sending and retrieving data to and from the database container.

E. Kubernetes

Management of multiple Docker containers grow more important as the complexity of applications continues to grow. Kubernetes is a central component to the DevSecOps pipeline as it's responsible for the management and deployment of "dockerized" applications. The Mattermost-server application is stored in Docker containers within a Kubernetes pod. A pod is the smallest resource in Kubernetes, and it represents a group of one or more application containers and volumes.

For this pipeline, the Mattermost application is deployed using a Kubernetes deployment, which is responsible for creating, updating, and deleting pods. Kubernetes deployment is implemented because of its self-healing mechanism. If a node hosting an instance shuts down or is deleted, the Deployment controller in the Kubernetes cluster replaces the instance with an instance on another node in the cluster. This mechanism provides continuous uptime and management.

F. Ansible

The DevSecOps pipeline relies on Ansible for application deployment. Ansible is an automation platform that makes applications and systems easier to deploy. Ansible configuration management starts with the concept of an inventory. An inventory is simply a list of hosts to run configurations on. In this case, the target host is the Kubernetes API server that is accessed by all users and components in the cluster. Hosts can be grouped into a list of hosts, which is specified in an Ansible playbook.

A playbook is a list of tasks or configurations to run on a subset of hosts. Tasks can range from the execution of bash commands to starting/stopping a specific daemon or service. For the continuous deployment of the Mattermost application, Ansible is tasked to modify the image directive of the Kubernetes resource files, while subsequently applying the file changes to the Kubernetes cluster to run the latest application version.

G. *Prometheus*

As developers seek to move applications to an enterprise-level, monitorization is of utmost importance. It's critical to know whether an instance can handle web traffic or whether a specific service or daemon is running. Prometheus is a monitoring solution that gathers time series based numerical data. One of the key features of Prometheus is that it uses a time series database. Users can fine-tune the definition of metrics and generate more accurate and precise reports because it provides a powerful query language. The pipeline utilizes Prometheus as means to gather application metrics from the Kubernetes pods that are hosting the Docker containers needed to run the Mattermost-server application.

Metrics such as network input/output and filesystem usage are continuously monitored. In order to better interpret application metrics pulled by Prometheus, Grafana is used in the pipeline to visualize the data pulled by the Prometheus pod. Grafana is a leading graph and dashboard builder visualizing time series infrastructure and application metrics. It allows the creation of alerts, notifications, and filters for various data sources. Grafana portrays the current status of the Kubernetes cluster by displaying various cluster-level metrics such as network input/output, filesystem usage, CPU usage, etc as shown in figures 4 and 5.

Figure 4: Grafana dashboard demonstrating Pods CPU usage

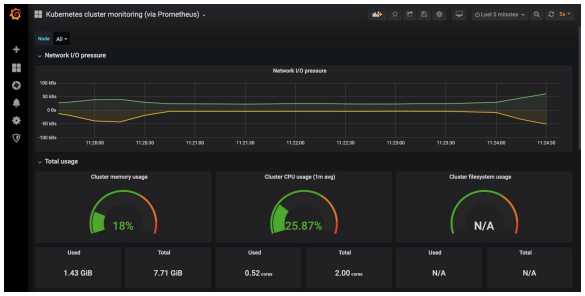

Figure 5: Grafana monitoring Kubernetes clusters

IV. IMPLEMENTATION

There are three prominent cloud platforms in the market: AWS, Microsoft Azure, and Google Cloud Platform (GCP). Since AWS has the advantage of a seven year head start and is the lead in the CSP market share, it is selected as the cloud environment for the DevSecOps pipeline.

A. *AWS*

The architecture of the pipeline is implemented on the AWS platform. Five Elastic Compute Cloud (EC2) instances are launched as follows: 4 t2.medium and 1 t3a.large. The t2.medium has 2 vCPUs and 4 GiB memory with an operating cost of $ 0.0464/hour while the t3a.large instance has 2 vCPUs and 8 Gib memory with an operating cost of $ 0.0752/hour. Public IPs, referred to as Elastic IPs in AWS, are incorporated into the architecture, and they are attached to instances that require Internet access. Security groups are properly configured at instance level to act as firewalls, and all instances permit SSH traffic since access to server instances is possible through SSH connection.

Public Key Infrastructure (PKI) is the technology used to authenticate to the server instances based on the RSA cryptosystem, and each instance has its own unique RSA key. Open-source terminal emulators such as PuTTy and Mobaxterm are used to connect to the servers using SSH protocol. AWS generates PEM (Privacy Enhanced Mail) keys, a base64 container format for encoding keys. Mobaxterm accepts PEM keys, but PuTTy requires PPK(Putty Private Key); as a result, PEM keys must be converted to PPK, and conversion is possible through PuTTY features.

AWS assigns public IPV4 addresses, private addresses for internal traffic flow, and public DNS addresses to each EC2 instance. Server instances that need to communicate to each other would use the URL address to initiate a conversation in order to exchange data. For EC2 instances that have Elastic IPs attached to them, this address remains the same each time the machine is stopped/restarted. However, for the virtual machines that are configured without Elastic IPs, every time an instance is restarted the URL address changes.

B. *Alternative Approach*

Microsoft Azure is another suitable platform to orchestrate the DevSeOps pipeline, and it has many services that range from computing services to storage and data management. According to Microsoft, more than 95 percent of Fortune 500 companies use Azure [3]. Azure services can also be utilized to integrate with open-source applications. Research reveals these differences between AWS to Azure:

1. PRICING: The price of Azure compute services is slightly cheaper than AWS for smaller instances. The biggest difference between their On-Demand per-hour compute service prices is around 20 cents for a Windows 16GB, 4CPU, General Purpose instance. Also, Azure follows a pay-by-minute model while AWS follows a pay-by-hour model.
2. COMPLIANCE: Both CSPs are offering services to handle classified information for the public sector. Azure Government cloud and AWS GovCloud are infrastructures designed by these companies to serve government entities. AWS provides a high-availability cloud platform sporting the security and reliability needed by government organizations across all classification levels, including Unclassified, Sensitive, Secret and Top Secret. Azure, meanwhile, offers a flexible and hybrid environment with security and compliance [3].
3. FLEXIBILITY: AWS offers more support and flexibility with third party vendors while Azure services are not fully compatible with certain vendors.
4. HYBRID FLEXIBILITY: Azure offers more support and flexibility with customers who wish to keep using their on-premise servers.

Both CSPs allow customers to reserve virtual machines from 1 or 3 years allowing savings from 40 to 62 percent. However, AWS allows convertible payment attributes allowing to change instance families, operating system, tenancy, and payment options [4]. Since all the instances in the proposed DevSecOps architecture require at least a medium size machine to run applications on, the cost estimate for both platforms are similar.

V. BUSINESS PLAN

This pipeline is designed to investigate the functionality of open-source tools in cloud environments, so the scope of the design is tailored for experimental purposes. In addition, the project is aimed to estimate the cost of AWS services on demand and without any long-term or short-term contracts. Duration of the project is approximated for six months, utilizing 5 EC2 instances for the chosen applications.

In order to optimize cost, unutilized EC2 instances are stopped after accomplishing tasks. The only cost incurring for open-source tools is the cost of the server that application is running on it. Other costs are associated with the use of the platform. There is no cost for transferring data within AWS, but cost would incur for data transferring out. Depending on the service and duration of usage daily cost varies, but the pipeline incurs a flat cost of $2.95. Based on the type and on-demand usage of services that are required for a period of six months, a budget of $500 is approximated for this project.

Return on Investment(RoI) depends on the business model that aspires to engineer the DevSecOps pipeline in the cloud platform. If incorporating the DevSecOps pipeline increases releases twice as fast, it is possible to expect the annual revenue to be doubled. For enterprise businesses, it is proven that a 20% annual increase in annual revenue is feasible by incorporating DevOps [5].

VI. RESULTS

Once every component of the pipeline is placed and configured according to the architecture, the entire pipeline is checked to confirm system meets requirements and specifications, and that it fulfills its intended purpose

A. *Verification*

Pipeline is gone through smoke testing also known as “verification testing”. The main purpose of this test is to ensure that most important instances are functioning properly. The result pinpoints a single

point of failure caused by the Jenkins server. When this server is overloaded with requests, it tends to crash frequently. In order to have a stable and fault-tolerant system, auto-scaling is configured to vertically scale in/out the instance on demand.

Applications and instances are frequently patched to mitigate vulnerabilities. Every time, instances are updated or upgraded, regression testing is performed to ensure changes do not cause malfunctions. During the last regression testing, SonarQube server stopped running. After troubleshooting, it becomes evident that SonarQube relies on Java or OpenJDK 11 to run, and upgrading to OpenJDK 13 causes the server to crash. After the system is set back to JDK 11, the failure is resolved. Once it is verified that every component of the pipeline works in a harmonious way, the project is concluded.

B. Validation

Validation of DevSecOps pipeline depends on different factors such as product reliability, security, expense reduction, increase in revenue and efficiency. Since servers are running on demand, the cost is significantly reduced. The reliability and security of the pipeline in the platform is guaranteed by AWS Shield, a managed Distributed Denial of Service (DDoS) protection service. AWS Shield provides always-on detection and automatic inline for the pipeline. The functionality and reliability of the pipeline is verified by monitoring Grafana dashboard as it shows a smooth performance of the Mattermost prototype within the DevSecOps pipeline.

VII. CONCLUSION

Open-source applications optimize cost and eliminate the issue of vendor lock-in; consequently, they empower businesses to have the flexibility of deployment at ease. Automation, containerization, and testing can all be achieved by opens-source tools. Investigation shows that both AWS and Azure are great vendors for flexible deployment. However, the choice between these two CSPs comes down to the business model. If a customer is planning on employing more open-source applications, then AWS is the excellent choice as it offers several integrations. On the other hand, if a customer desires a flexible hybrid cloud system to connect the on-premises servers to the cloud, then Microsoft Azure is recommended. Indeed, a meticulous architecture along with a proper cloud service provider could facilitate deployment, prevent vendor lock-in, and bring any DevSecOps projects into reality.

REFERENCE

[1] Columbus, L. (2018). 83% of enterprise workloads will be in the cloud by 2020. *Forbes,* https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/#39f45006261a

[2] D. Marcilio, R. Bonifácio, E. Monteiro, E. Canedo, W. Luz and G. Pinto, "Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube," *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, Montreal, QC, Canada, 2019, pp. 209-219.

[3] D, Ramel, AWS vs. Azure Heats Up the Market, *Washington Technology,* September 14, 2018. Accessed on: March 20, 2020. [online]. Available: https://washingtontechnology.com/articles/2018/09/14/aws-vs-azure-public-sector.aspx

[4] Azure vs. AWS. *Microsoft Azure.* Accessed on: March 20, 2020. [online]. Available: https://azure.microsoft.com/en-us/overview/azure-vs-aws/

[5] A. Manday, The ROI of Enterprise DevOps, *DevOps.com*, March 9, 2019. Accessed on: March 20, 2020. [online]. Available: https://devops.com/the-roi-of-enterprise-devops/https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/#39f45006261a