

# Building An Automated Ecosystem On AWS Implementing Robust Security Measures Using DCVS

MSc Research Project  
Cloud Computing

Ganesh Patil  
Student ID: x20193009

School of Computing  
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Ganesh Patil
<b>Student ID:</b>	X20193009
<b>Programme:</b>	Cloud Computing
<b>Year:</b>	2021
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Sean Heeney
<b>Submission Due Date:</b>	15/08/2021
<b>Project Title:</b>	Building An Automated Ecosystem On AWS Implementing Robust Security Measures Using DCVS
<b>Word Count:</b>	5500
<b>Page Count:</b>	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

<b>Signature:</b>	Ganesh Patil.
<b>Date:</b>	15th August 2022

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Building An Automated Ecosystem On AWS Implementing Robust Security Measures Using DCVS

Ganesh Patil  
x20193009

## Abstract

Development and Operations or DevOps is an approach that tries to create a working relationship between developer and the operations teams to deliver quickly and to automate the continuous delivery of new applications, shorten the SDLC or software development lifecycle to create high quality software. In the similar fashion, incorporating the security practices into the Devops methodology is called as DevSecOps. Software developed using this process guarantee the privacy, safety, and resilience of applications. DevSecOps is not a new concept, but it's adoption has been very late in the industry. In this research paper we will try to build an architecture which will be able to integrate security inside the DevOps methodology using the AWS Cloud. Integrating security in this context means using open-source vulnerability scanner into the architecture and making the whole system of static and dynamic scanning automated.

## 1 Introduction

Cloud Computing is a discipline which is a popular for enhancing efficiency and growing capacity of the existing infrastructure and it relies on the principle of how the application is being created, developed and run on the cloud environment. Cloud-Native applications has a major contribution in building the applications that are scalable and developed in modern environment. In order to build these loosely coupled systems technologies such as containers, microservices, orchestrations tools are taken help of to make the architecture more resilient, flexible and scalable Garces et al. (2020)

As we progressing with the technology, Cloud-native seems to have established itself as the defacto norm for the business now a days. With these modern practices comes the threat of being vulnerable to attacks and being exposed to external attacks which can be disastrous especially when these are critical applications and peoples life depends on it for day to day use, such as banking applications.

What this essentially means that, while developing the applications, sufficient care should be taken such that the application which is developed should follow all the DevSecOps principles. Akbar et al. (2022) state that DevSecOps is nothing but the integration of Development, Security and Operation which essentially translates to developing the applications using Devops methodology and while integrating the security methods. DevSecOps is a software development method where security is integrated into application development process to assure application confidentiality, integrity, and availability.

Regardless of the fact that security has grown to be businesses' top issue when it comes to digital transformations, But the new paradigm such as DevSecOps has not be able to be implemented by operations team because of the technical challenges and cultural hurdles. According to the Tomas et al. (2019) DevSecOps aims to bring together the previously established IT Team silos of the operations, security and app development team .

Applications developed by using microservices based architecture may help businesses' archive continuous delivery of software as a part of their methodology, but they have also made it more susceptible to abrasive cyber-attacks like the well known Log4j exploit. This could be avoided by introducing the application level security which can run side by side of the app development process.

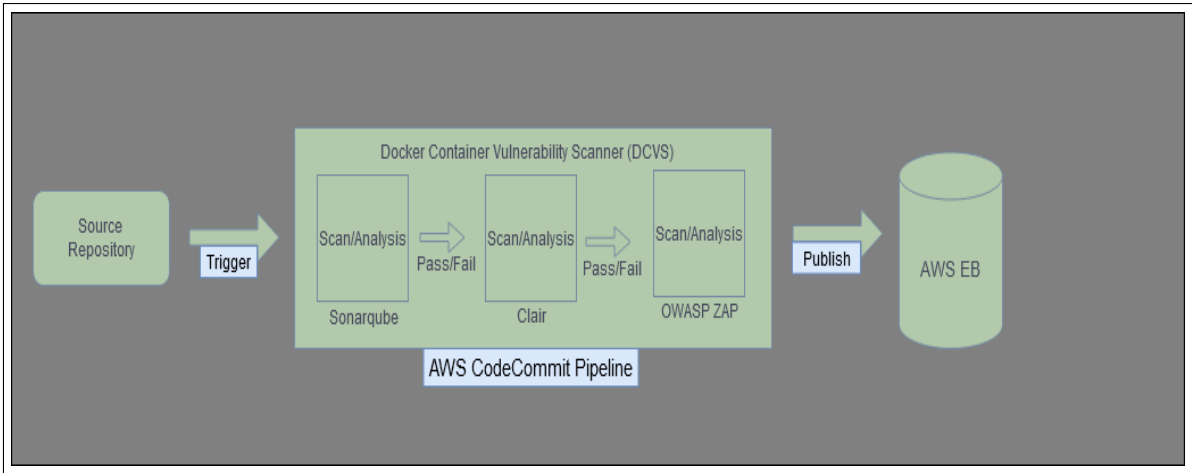


Figure 1: Basic Architecture

## 1.1 Motivation

In today's era combining multiple microservices to form one big application is generally the way in which the software industry works. This kind of architecture is loosely coupled and provides much more flexibility in developing as well as deploying the applications on cloud(Suram et al.; 2018). Such kind of architecture is helpful in troubleshooting the issue and finding bugs in the application, which is not the case if considered monolithic architecture. Especially when we are trying to follow the greenfield approach this kind of methodology helps a lot in making the whole DevOps process simpler.

With the increase in adoption of this modular approach and making the developer and operations team work together is not enough. Subashini and Kavitha (2011) suggests rather than just focusing on the delivering the capabilities, building it quickly and delivering the needs is just making the applications more and more susceptible to exploits, because quicker delivery means less time to conduct all tests and thus less testing of the application will result in making it prone to attacks.

Even if the issue of getting enough time to conduct tests is fixed, but there still remains the issue of types of tests that can be performed as in most of the companies the only type of testing is done is static. They use tool such as sonarqube to conduct the static testing, making it vulnerable to dynamic and runtime vulnerabilities.

Also, everytime the developer makes a commit, there is dependency on security team to see if the code is complaint and not exposed or any sensitive information is not being leaked out. The main goal of the DevOps mythology was to automate the repetitive and manual task and considering the dependencies if the security team's head is by chance not available and the substitute engineer is also not available to perform the checks and changes made in repository. This situation will create a bottleneck and hamper the delivery estimation leading to revenue loss.

Considering all these facts, I tried to automate the manual and repetitive tasks of scanning the repository for not only static vulnerabilities, but also checking for the dynamic and runtime ones. Thus emphasizing on the DevSecOps principles.

## **1.2 Research Question**

Can the reliability of cloud-native applications be improved by automating the static and dynamic vulnerability scanning using a novel Docker Container Vulnerability Scanner (DCVS) by leveraging it on AWS platform?

DCVS is an amalgamation of open-source security vulnerability scanners which are integrated with the AWS platform which is able to detect the static and dynamic vulnerability scanning. Static vulnerabilities are those anomalies which scan the code before the application is deployed. it usually check the source code and byte codes and scans for any suspicious code smell or security weakness which could be found. While the dynamic vulnerabilities are those type of anomalies which are detected after the application is already deployed. According to DuPaul (2015) what it basically does is, it scans for the application inside out and it simulates a kind of attack on the application and tries to manipulate it in the running state.

## **1.3 Structure Of The Paper**

The research paper has the following sections. Section 2 conveys about the related work done in the past regarding this topic. Section3 tells us about the procedure followed to obtain the required output for this research. Section 4 gives a brief overview of the architecture used to achieve the results. Section 5 provides the description about steps taken to implement this project. Section 6 and Section 7 explains the results obtained and tells us about the futher improvements that can be made in the research respectively.

## 2 Related Work

DevSecOps has been gaining traction in the last few years and there is no denying that it has gained a lot of popularity. It has become necessary to integrate good security practices into the traditional DevOps practices to be ahead of the competition in the world.

### 2.1 Background of Microservices

It used to be a laborious procedure for software developers to offer instructions on how to execute the program along with a list of prerequisites when they had to send apps to another team or deploy them over the cloud. This led to the creation of an idea called container. A container is somewhat similar to a virtual machine but typically just on the OS level. The OS level virtualization is what it uses to isolate its resources, houses a fully virtualized guest OS running on the host OS. Many big tech companies such as Google and Amazon are using Docker as a platform as a service for running containers according to Brady et al. (2020). With the help of this technology the software developer can package all the code, dependencies and configuration files together into one single package known as container.

### 2.2 Types of Vulnerabilities

The taxonomy below discusses the vulnerabilities or the security flaws which are usually encountered by the containers before or during deployment. Aside from the aforementioned flaws, security vulnerabilities can be categorized into two types and those are static and dynamic.

- **Static:** Static analysis is the type of analysis in which the evaluation of the code takes place beforehand and is scanned for any vulnerabilities such as infinite loops, memory leaks or code smells and there are some predefined actions to take against such vulnerabilities. The Common Vulnerabilities and Exposures is a database which stores the latest records of such flaws and any static vulnerability scanner will scan for exposures or any other security flaw and compare it with the CVE database before the container is being deployed.
- **Dynamic:** This sort of scanner is based on the principle that it would start its vulnerability assessment after the application is installed. The application is scanned and monitored for the consumed resources and it monitors the ports which it is using. The CVE database also contains the list of such vulnerabilities and usually applications performing the dynamic scanning scan the apps first and then check the run-time usage and also detect if the app is facing a DDOS attack or any such type of attacks.

### 2.3 Existing Methodologies

Lopes et al. (2020) have proposed the method of container hardening which essentially means that if the attack surface of the container which is running the application is reduced then it will prevent it from being vulnerable to attacks. For this a sec-comp

profile needs to be created to know what kind of image is used and according to this a different profile will be created.

Yang et al. (2021) in their paper have stated that if there are two containers sharing the same abstract resource by using the OS-level virtualization, then the attacker can easily gain the access of the shared kernel using the abstract resources. The proposed a very unique and innovate way of approach and i.e we should control the or confine the abstract resources which are being used by the containers. Once we control the resources the attacker will get very less opportunity to attack.

Tunde-Onadele et al. (2019) provided a brief study about the container vulnerability exploit detection which provides a detailed overview of the static and the dynamic vulnerabilities. It has also stated that the use of unsupervised machine learning technique can be used to detect any vulnerabilities in the system.

In the paper written by R.G.K.P.Kulathunga (2020) it provides a new approach in the system of intrusion detection. It stated that if the memory usage of the model should be decreased and so as to limit the usage of resources so that if in case an external DDoS attack takes place then we will be able to control the mitigate the attack sooner.

Sarkale et al. (2017) propose in the research paper that a privileged based application control system should be used. In the paper they have made a detailed overview of the latest methods available for the analysis of the security model and they have found that privileged based access control is the best out all the current approaches.

Sultan et al. (2019) in their paper have given a detailed summary on the security of the container and have also proposed various ways to increase the proposed security of the container. The first approach is related to container hardening, second one being to role based access control, third being the limiting the resources and last one being the reducing the surface of attack for the container.

Abhishek and Rajeswara Rao (2021) This is the base paper of the research and it considers the factor of static analysis and it is done using the third party vulnerability scanners which scan for the static vulnerabilities and notifies the user.

## 2.4 Comparative analysis of previous methodologies

Reference	Title	Proposed Approach	Advantages	Limitations
Abhishek and Rajeswara Rao (2021) Abhishek et al.	<b>Framework to Secure Docker Containers</b>	Use Sonarqube for detecting static vulnerabilities	Bugs and vulnerabilities which are static in nature are detected before pushing app to staging environment	Doesn't consider dynamic vulnerability scanning. Also, malware and Trojan scanning not included.
Garg and Garg (2019) Somya et al.	Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security	The approach provides an brief overview of automation of the security practices	Docker best practices are provided with detailed overview	Do not support CPU and memory bound workloads.
Brady et al. (2020) Kelly et al.	Docker Container Security in Cloud Computing	Scanned the container images with antivirus for removing malicious code.	Virus scans are effective against docker container	false positive cases might be there sometimes
Javed and Toor (2021) Omar et al.	Understanding the Quality of Container Security Vulnerability Detection Tools	Detection hit ratio is an approach which is used for the scanning of the docker containers	OS as well as Non-OS vulnerabilities are identified	Doesn't consider run-time vulnerabilities.

Table 2.2: Literature Review Summary.

### 3 Methodology

This section provides a detailed methodology that has been followed to achieve this research. The flow followed by the process will be explained in 3.1 and 3.2 will provide a brief overview of the services and tools used during this research.

The research adds value to the existing methods of testing the applications for security vulnerabilities by automating the process and also not just testing for static vulnerabilities, but also scanning them for dynamic and runtime vulnerabilities. To get the task done without worrying about buying expensive tools licenses and dealing with its compatibility, this research has used open-source tools to conduct these experiments.

To achieve the expected results for this research, the application code will be pulled from the repository everytime a commit is made by the user, which later on will go through a pipeline of rigorous checks. The pipeline consists of DCVS or Docker Container Vulnerability Scanner which is an amalgamation of various open-source tools put together. All the open source tools used are mentioned in 3.2 which would be contributing in making the application more secure. The code will be scanned for not only static, but also dynamic and runtime vulnerabilities. When the initial static scanning is done an image is created which is then pushed to repository and then further dynamic and runtime scanning is done. If the build is successful then the container is sent to Amazon EKS staging environment and later to production. If the build fails at any stage then a lambda function is invoked which in turn generates a cloud watch event and notifies the user about the result and the log files are stored into Amazon S3 bucket. The beauty of this process is it is all automated and leads very less manual interference making it avoid bottlenecks in the process.

#### 3.1 Process flow of research

To perform the research we have utilized the platform of Amazon WebServices and with the help of some third party open source tools an architecture is built to scale according to the user's needs. This architecture is named as DCVS or Docker Container Vulnerability Service which will enable in automating the security testing of the cloud-native application. A simple HTML web-app will be scanned for secrets and static vulnerabilities and later converted into image and stored in a repository which further will be converted into a container for deployment into the staging environment where dynamic and runtime scans will be performed to evaluate for sending it into production environment.

In the architecture the static scanning will be done by the SonarQube, Dynamic scanning will be done by the OWASP ZAP. More details about the tools used are explained in the 3.2 section.

The following figure 2 shows the process flow of the research.

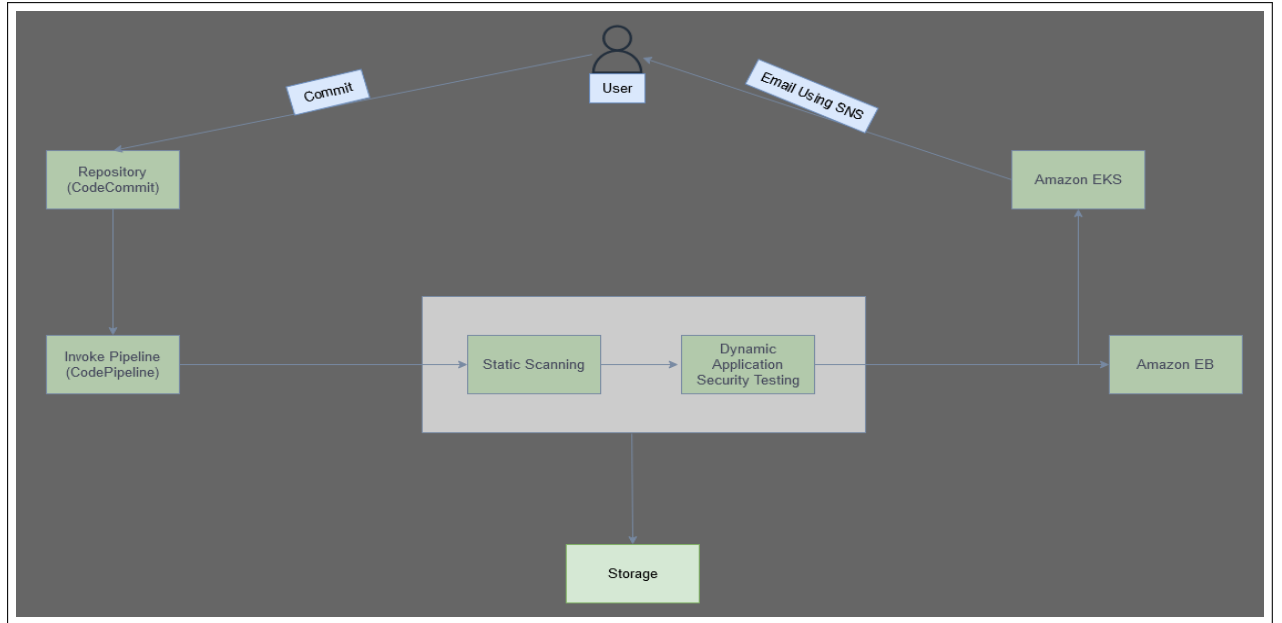


Figure 2: Process Flow Of The Research

### 3.2 Tools and Technologies Used In Research

To perform this research and implement the proposed architecture a public cloud cloud called AWS or Amazon Web Service has been used. The resources used as clubbed with open source tools to take care of security testing of applications and technologies based on open source tools have been used to provision the instances and spin up the clusters. To keep it simple a sample HTML application will be used as an example to demonstrate the research.

Following tools and services have been used to conduct the research:

- **CodeBuild:** It is a totally autonomous continuous integration service which can compile the code and generate software packages that can be deployed easily into the cloud. CodeBuild takes care of all the provisioning, managing and scaling of servers for you.
- **CodeCommit:** Codecommit is a fully managed version control service that is similar to GitHub and is able to host private repositories such as those present in Git. One of the best feature it has is that it can scale very easily and is secure.
- **CodeDeploy:** It is an AWS service which takes care of software deployment for you. It is fully automated and can be managed easily. Using CodeDeploy, the deployments can be automated to various compute services such as EC2 and Fargate.
- **CodePipeline:** CodePipeline is a continuous delivery tool which can help developers to automate release cycles for quicker and dependable software and infrastructure changes. Every-time a change is made in the code by the user it automatically builds, tests deploys parts of the release workflow.

- **Lambda:** It is a service which allows users to run code without the need to manage servers or provision them. We can pay for the only time we used the compute resources and don't need to pay any extra costs.
- **S3:** It is an online storage which can be accessed with the help of internet and the data stored in it can be retrieved at any point of time with the help of internet.
- **ECR:** ECR or also known as Amazon Elastic Container Registry is a highly scalable and secure container registry which developers can rely upon to store private repositories which can be accessed only by IAM permissions.
- **Cloudwatch Logs and Events:** The Cloudwatch events provides a very accurate and real-time stream of the events happening in the system which describes the updates that took place in the AWS services. While the AWS Cloudwatch Logs closely monitors, stores and retrieves the log files from various sources such as EC2 instance, Route53 etc.
- **IAM:** Allows you to securely control authorization to AWS services and resources. IAM enables you to create and handle AWS users and groups, as well as utilize permissions to grant and deny them access to AWS resources.
- **SNS:** Simple Notification Service is a service which provides services of messaging either from application to application or application to person.
- **Clair:** Clair is an open source tool used by ECR to scan for the repositories. It scans for the OS type vulnerabilities.
- **OWASP ZAP:** It is responsible for providing dynamic application security as it assists in automatically detecting the security flaws in the web-apps while building and testing them.

### 3.3 Assessment Carried Out In This Research

In total three experiments would be conducted to evaluate the results obtained from the vulnerability scans. We will also check if for conditions such as if the pipeline invokes when the commit is made and ensure that changes made in the repository are reflected in the production environment and at the end we will check if the user receives updates via email about the success of the build.

The three experiments will be based on the fact where will expose the architecture with a clean application image, an image with medium vulnerabilities and the last image would be of highest number of vulnerabilities. Based on these three cases we will build up an analysis on how the architecture reacts and perform results.

The observations are showcased in the 6 where a detailed analysis of the results are evaluated and discussed.

## 4 Design Specification

The preferred configuration used for crafting the research project is mentioned below. Section 4.1 describes about the tools and the operating system used while 4.2 provides an overview of the architecture which was proposed earlier. Section 4.3 describes about the architecture of DCVS

## 4.1 Proposed Specification Of The System

Below are the proposed specifications used for the system. As the account we created was a free tier account and the project requirement was not very computing intensive. Hence it was tried to use the configuration mostly in free tier only with minimum requirements. The Platform used was AWS cloud and an in-house code editor called Cloud9 and for configuring the resources remotely through CLI AWS version 2 was used which is the latest.

Configuration Used		
<b>Instance</b>	AWS	Config=t2.micro
<b>Operating System</b>	Windows	Version = 10
<b>Orchestration tool</b>	AWS CLI	Version = 2.7.19
<b>Container engine</b>	ECR	Version = 1.61.3

Table 2: Preferred configuration for the architecture.

## 4.2 Proposed Architecture Of The System

To access the infrastructure of the AWS through command line interface we must use the AWS CLI client. To conduct the study kubect1 verison 1.21 has been installed and the installation of kubeadm package is done. In total we would be pushing the repository from a code repository to container service All the services used in this system as scalable and highly available, So managing the traffic and scaling the infrastructure won't be a problem.

The build stage consist of four stages, each stage has been divided into four stages with each responsible for a certain task. If at any point the task fails the lambda function is invoked which updates the security hub with the details and post the detailed reports of the vulnerabilities in the S3 bucket. The system is constantly monitored by the Cloudwatch events and an event is triggered every time an critical update is present.

The nodes are integrated with tools to sense anomalies in the system and the pipeline is designed in such a way that if the number of vulnerabilities get across the specified limit, it would fail the build and notify the user. Every time a commit is made to the repository the cycle starts and it follows the process from the start. The figure 3 provides an high level overview of the proposed system.

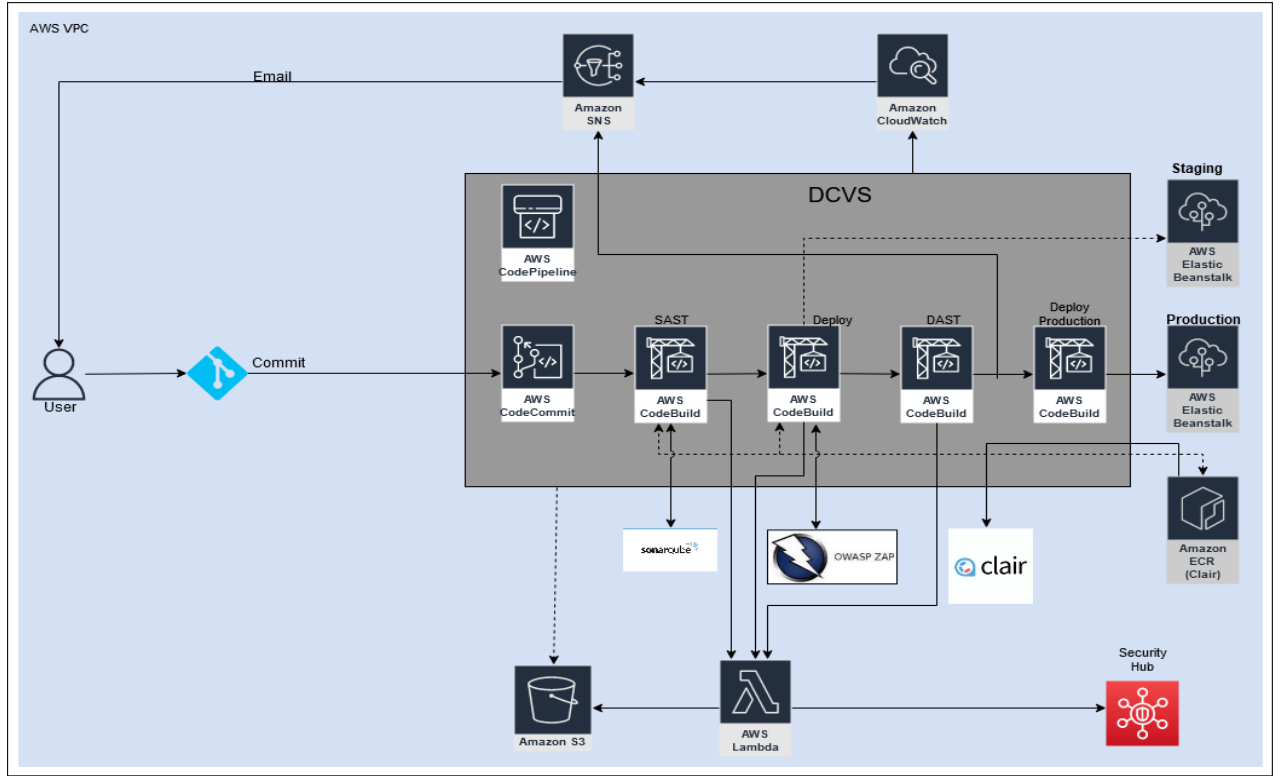


Figure 3: Architecture Diagram of DCVS

### 4.3 Proposed Architecture of DCVS

While conducting the research on this topic, an attempt is tried to design a system which would be automated and can perform security testing on the cloud native applications without getting stuck for any dependencies on other team. The DCVS has been designed in such a way that each code is first packaged as an image tested for its vulnerabilities and later sent into production to run inside a container. It not only takes care of the security before running the container, but also after it is being deployed into production. These tools have been used in the DCVS, Hence there is no need to worry about any tool being deprecated or any vulnerability not being able to be detected.

The architecture shown in 3 provides a high level overview of proposed system. This proposed architecture is based on the fact of automation and it follows the security principles of not only scanning for static vulnerabilities, but also scan for the dynamic. As stated in the proposal, the static vulnerabilities are been taken care of by the sonarqube engine and in combination with clair it will scan for the OS and non-OS type vulnerabilities. For handling the vulnerabilities such as dyanmic ones we have OWASP ZAP which has the latest database of CVE(Common Vulnerabilities and Exposures) which will work as agent to remove these types of vulnerabilities Considering all these factors we can say that this is an overall review of the vulnerabilities which can be thought of affecting the lifecycle of the microservice.

The security of the DCVS is protected by the IAM roles which restricts to access to certain resources and people only S3 bucket policies have also be configured to manage efficient use of resources in pipeline. Using AWS architecture helps in not only encrypting

the data in pipeline, but also provides SSL grade transport security. A parameter store helps in storing the sensitive info such as passwords and tokens.

## 5 Implementation

The general implementation of the whole architecture has been discussed in 5.1. The detailed steps are provided as as below:

### 5.1 Implementation of DCVS

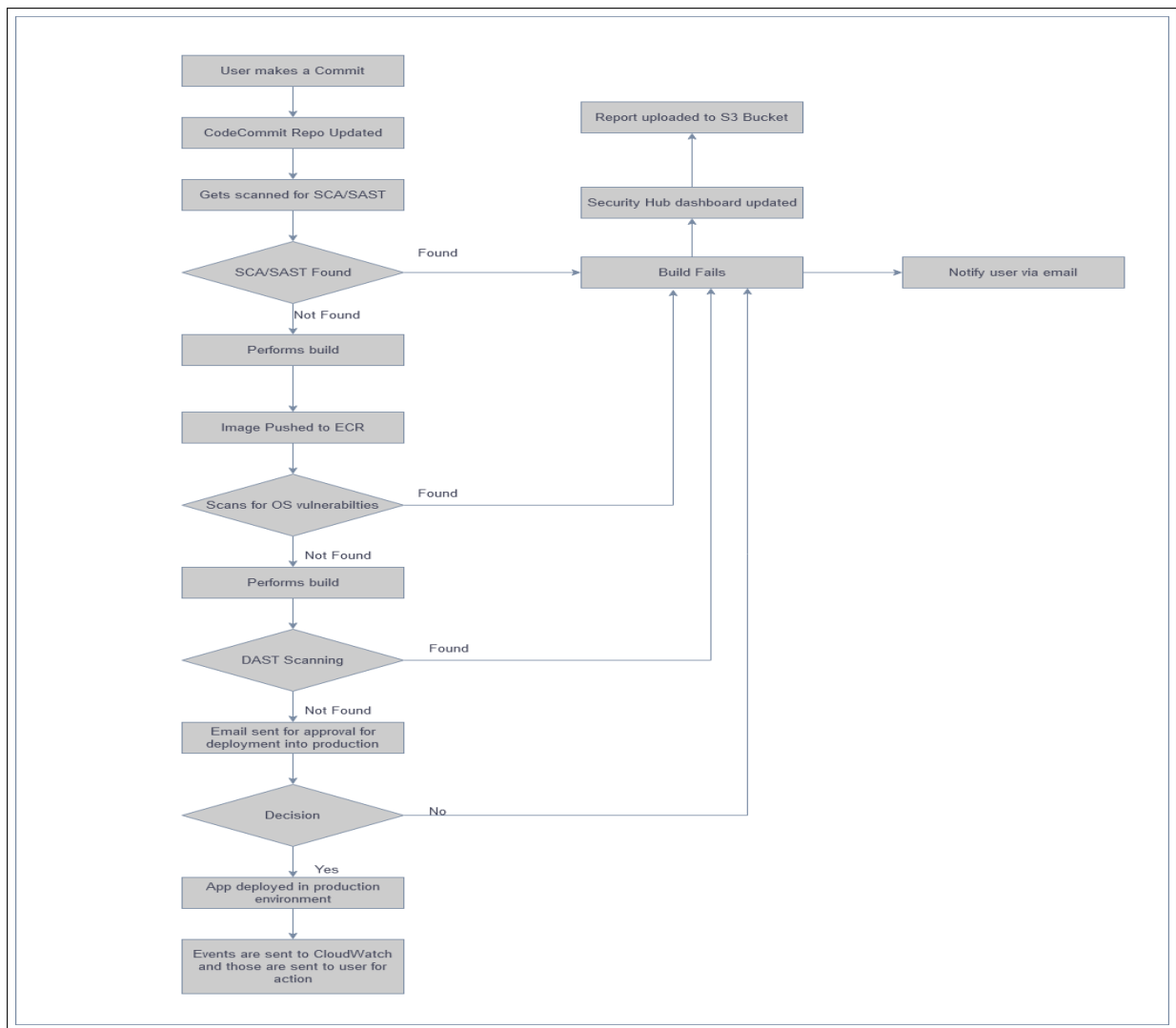


Figure 4: Flow diagram of DCVS

- When the CodeCommmit repo is updated by the developer,an event is generated of Cloudwatch which will notify the codepipeline for further orchestration of the resources.

- As the pipeline starts the CodeBuild starts scanning for static vulnerabilities using the tool called Sonarqube which scans for vulnerability such as code smells, unreachable code etc. Sonarqube is the tool which is responsible for performing the static scanning. If the scanner encounters any security vulnerability as mentioned earlier then the lambda function is invoked and build gets failed and the report is generated which gets uploaded to the S3 bucket.
- This event is tracked by the AWS Config which notices that vulnerabilities are found and it will notify AWS Cloudwatch to generate an event which will send an Alarm notification to user via email and post the remaining details to security hub.
- If no vulnerabilities are found then the CodeBuild moves to the next stage. Now as the code seems to be free of SCA/SAST vulnerabilities, so now CodeBuild generates an image so that it can be further pushed to ECR for scanning for the OS vulnerabilities by using Clair. CodeBuild is based on the approach which believes customization, so by default it doesn't have its own tool and it depends on user what type of tool to integrate with.
  - Clair scans the code and if any vulnerabilities are found then a Lambda function is invoked which notifies the security hub. The security hub helps to find all the security evaluations at one place. The results of the scans are uploaded to an S3 bucket.
  - If the scanner doesn't find any vulnerabilities then the image is further sent for Dynamic scanning.
- As soon as the image is ready and sent by CodeBuild it is sent to the staging environment.
- When the staging environment is successfully set up then DAST scanning begins with the help of OWASP ZAP. The ZAP spider scans for the whole application first and then draws a map. This monitors the ports being and if there is any memory leaks. If the scans find any vulnerability then same procedure with uploading the results to S3 by Lambda function occurs.
- At the end if no vulnerabilities are found, then a email gets triggered which is delivered by the help of SNS for the approval to deploy the application to production environment to the user. Once the user approves that the application is good to go. The manual approval stage passes and the application is successfully deployed into the production.
- The user is notified of the critical events with the help of SNS using the email notifications such as failed builds or unexpected errors.

## 6 Evaluation

The evaluation section will consider three cases which has assessed the application based on the factor of severity of the vulnerabilities. The severity of vulnerabilities essentially means that the application will be tested by introducing vulnerabilities inside the application to look if the architecture is successfully able to detect the vulnerability and the

build fails accordingly. The architecture is able to detect for the static and dynamic type vulnerabilities. Hence, we will try to introduce vulnerabilities which are of three types i.e Critical, Medium, Low.

## 6.1 Experiment 1

The first experiment is assessing the critical type of vulnerabilities in the application. To exploit this vulnerability a critical loops exit has been commented out. This essentially means that resource on which it is provisioned will be stuck in infinite loop and won't be able to come out of it. This is a bad place to be in, If such type of mistake happens in the production environment.

```
46     if [ $high_risk_dependency = "yes" ]; then
47         echo "there are high or medium alerts.. failing the build";
48         aws lambda invoke --function-name ImportVulToSecurityHub \\  
49             --payload file://payload.json dependency-check-report.json && echo "LAMBDA_SUCCEEDED" || echo "LAMBDA_FAILED";
50         # exit 1;
51     fi
```

Figure 5: Critical vulnerability introduced

The result of introduction of the vulnerability can be seen in the image 6. Also the pipeline has failed can be seen in the image 7.

## 6.2 Experiment 2

The second experiment involves assessing the vulnerabilities which comes under the range of being not too vulnerable but enough to expose the application and thus comes under the category of medium vulnerable. In this experiment the EC2 has been configured with an extra security group which is exposed to the outside requests and has access to all the application which is not an ideal case. The filtering should be done by security group to isolate the resources and hence these have been categorized as medium severity vulnerabilities. In the image 10 we can observe the pipeline failed because of the same reason.

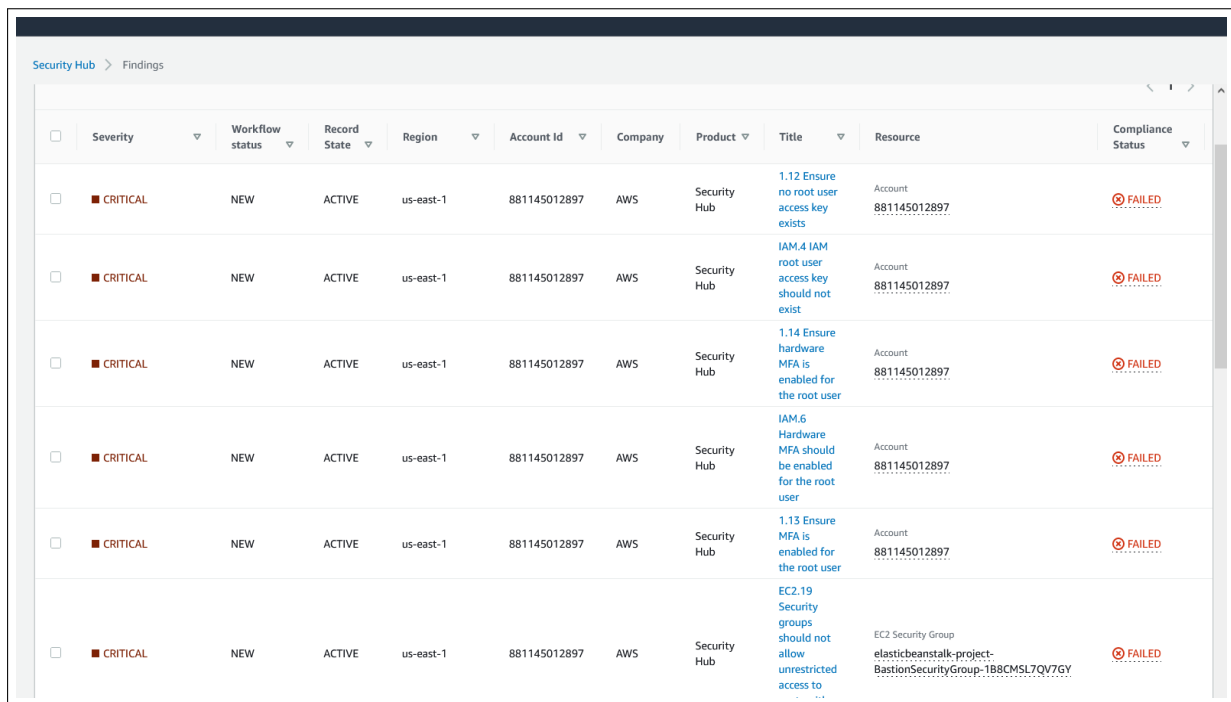
## 6.3 Experiment 3

Third experiment is based on the low severity vulnerabilities which is often related to not so critical but important vulnerabilities. These kind of vulnerabilities might not affect the application working or affecting the resource usage directly, but might be needed to be considered sometime. Hence, the application can be easily deployed onto the environment. In experiment 3, the vulnerabilities of IAM and Cloudformation have been detected as can be seen in image 11 and as they might not cause problem while the application is running the environment the application is successfully deployed. The image of successful deployment is shown in 12

## 6.4 Discussion

After conducting the three experiments by introducing the vulnerabilities in the system, the architecture was able to detect and categorize the type of vulnerability and provide a go or no-go to the pipeline for deployment. It was observed that the system can successfully identify the static and dynamic vulnerabilities thus making it much more secure and reliable.

Image 6 shows the critical vulnerabilities found out in the application. These vulnerabilities reflect the behaviour of the system and have been accurately captured by the architecture. Once the vulnerabilities have been detected the lambda function gets invoked which informs the security hub about the details and to send reports to S3 bucket as well as security hub. Further the cloudwatch is monitoring in the background gets to know about the security flaws and thus stops the execution of the pipeline. As the pipeline stops the SNS triggers an email which notifies the user about the status of the pipeline 8.



The screenshot displays the AWS Security Hub Findings interface. The table lists six findings, all categorized as 'CRITICAL'. Each finding is associated with the 'us-east-1' region, account ID '881145012897', and company 'AWS'. The findings are related to IAM policies and EC2 security groups. The 'Compliance Status' for all findings is 'FAILED'.

	Severity	Workflow status	Record State	Region	Account Id	Company	Product	Title	Resource	Compliance Status
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.12 Ensure no root user access key exists	Account: 881145012897	FAILED
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	IAM.4 IAM root user access key should not exist	Account: 881145012897	FAILED
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.14 Ensure hardware MFA is enabled for the root user	Account: 881145012897	FAILED
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	IAM.6 Hardware MFA should be enabled for the root user	Account: 881145012897	FAILED
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.13 Ensure MFA is enabled for the root user	Account: 881145012897	FAILED
<input type="checkbox"/>	CRITICAL	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.19 Security groups should not allow unrestricted access to	EC2 Security Group: elasticbeanstalk-project-BastionSecurityGroup-1B8CML7QV7GY	FAILED

Figure 6: Experiment 1 Summary of Critical Vulnerabilities

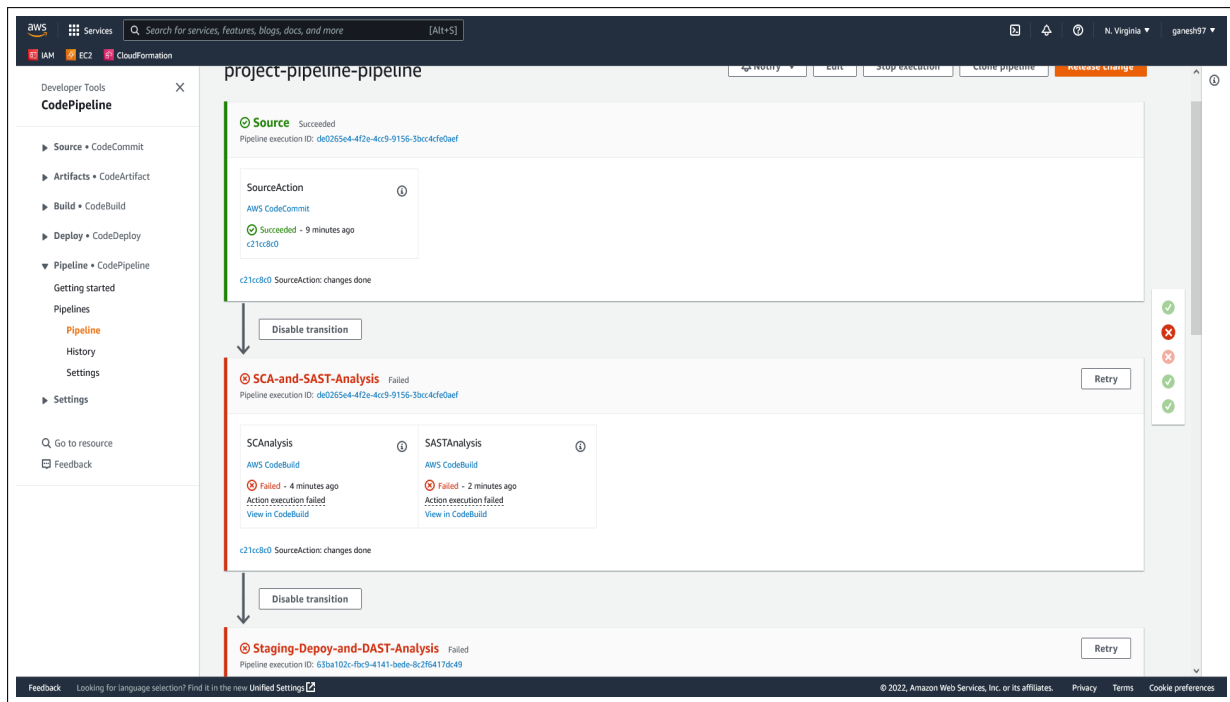


Figure 7: Experiment 1 Pipeline failed as the critical vulnerability is detected

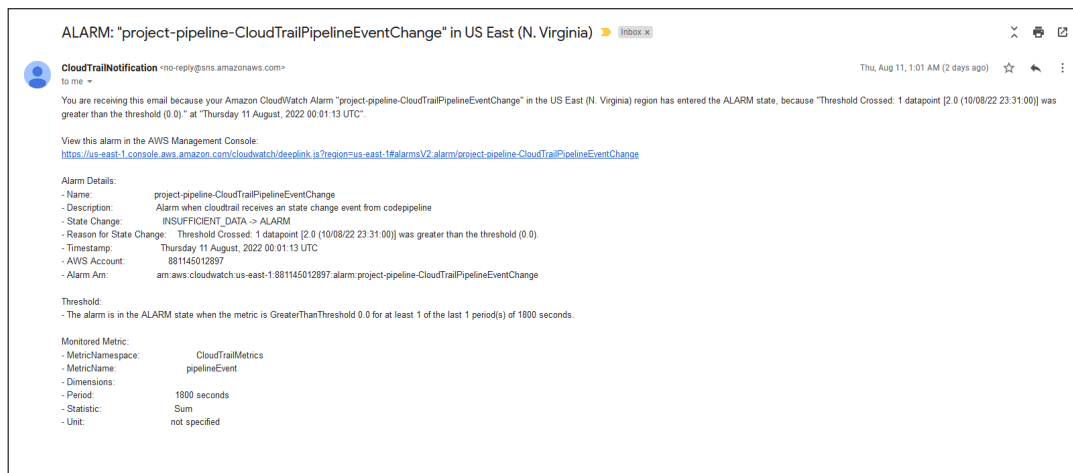


Figure 8: Experiment 1 SNS notification received by the user

In the experiment two, the vulnerabilities are captured and can be seen in the dashboard of securityhub 9. As intended in experiment 2, the security flaws have been detected successfully and it can be seen that these flaws have resulted in the pipeline to fail. A similar event of SNS is generated as explained above and the user receives an email about the failed pipeline as shown in the image 8.

Findings											
A finding is a security issue or a failed security check.											
<input type="text" value="Severity label is MEDIUM"/> <input type="text" value="Region: us-east-1"/> <input type="text" value="Workflow status is NEW"/> <input type="text" value="Workflow status is NOTIFIED"/> <input type="text" value="Record state is ACTIVE"/> <input type="button" value="Add filters"/> <input type="button" value="X"/>											
<input type="button" value="X"/> <input type="button" value="1"/> <input type="button" value="..."/> <input type="button" value="X"/>											
<input type="checkbox"/>	Severity	Workflow status	Record State	Region	Account Id	Company	Product	Title	Resource	Compliance Status	Updated at
<input type="checkbox"/>	MEDIUM	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.15 EC2 subnets should not automatically assign public IP addresses	AwsEC2Subnet subnet-0746edbb676aa60aa	FAILED	3 minutes ago
<input type="checkbox"/>	MEDIUM	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.15 EC2 subnets should not automatically assign public IP addresses	AwsEC2Subnet subnet-026ce06a414775f3a	FAILED	3 minutes ago
<input type="checkbox"/>	MEDIUM	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.15 EC2 subnets should not automatically assign public IP addresses	AwsEC2Subnet subnet-0a3afb0253c8ab1f	FAILED	5 minutes ago
<input type="checkbox"/>	MEDIUM	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.22 Unused EC2 security groups should be removed	EC2 Security Group elasticbeanstalk-project-BastionSecurityGroup-1B8CML7QV7GY	FAILED	2 hours ago
<input type="checkbox"/>	MEDIUM	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	EC2.22 Unused EC2 security groups should be removed	EC2 Security Group ssh-ingress-sg	FAILED	2 hours ago

Figure 9: Experiment 2 Medium severity vulnerability detected

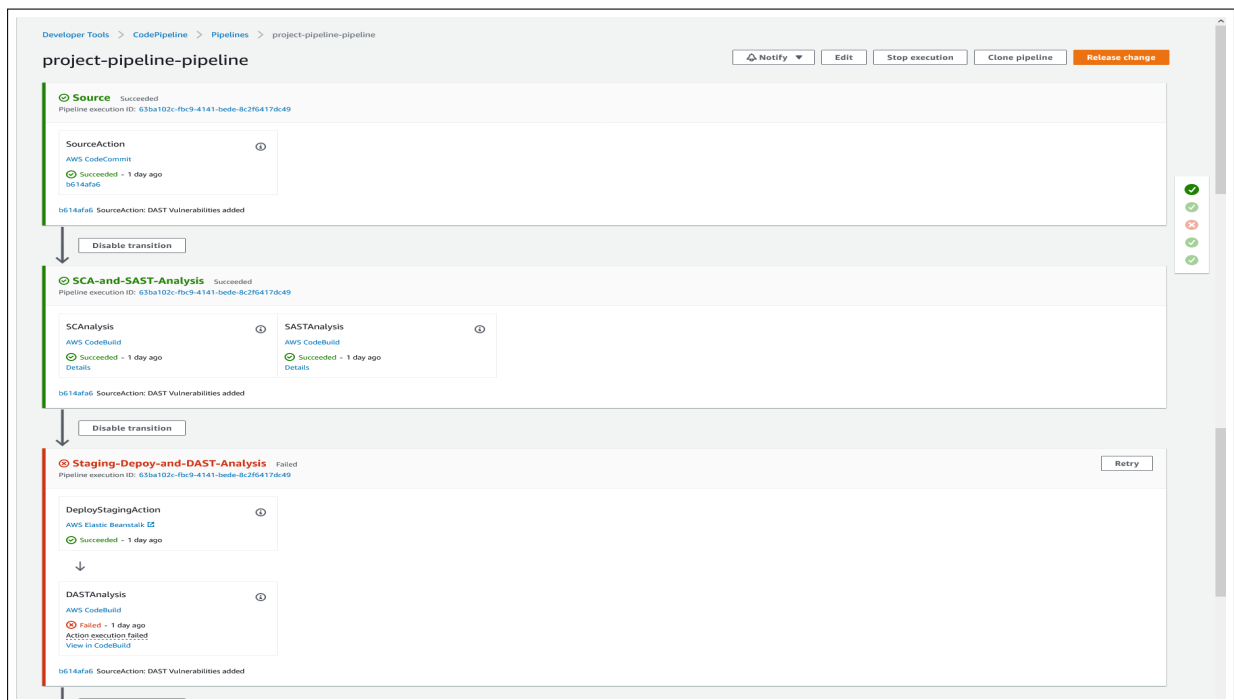


Figure 10: Experiment 2 Build failed because of detection of medium severity vulnerability

Experiment three was able to detect all the minor security flaws which was expected to happen and can be seen in the image 11. As these security vulnerabilities take the

very least priority security check. Hence, the application is deployed successfully and can be seen in image 12.

Security Hub > Findings

Findings

A finding is a security issue or a failed security check.

Severity label is LOW X Region: us-east-1 X Workflow status is NEW X Workflow status is NOTIFIED X Record state is ACTIVE X Add filters

<input type="checkbox"/>	Severity	Workflow status	Record State	Region	Account Id	Company	Product	Title	Resource	Compliance Status	Updated
<input type="checkbox"/>	LOW	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.11 Ensure IAM password policy expires passwords within 90 days or less	Account: 881145012897	FAILED	2 hours ago
<input type="checkbox"/>	LOW	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.10 Ensure IAM password policy prevents password reuse	Account: 881145012897	FAILED	2 hours ago
<input type="checkbox"/>	LOW	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	1.20 Ensure a support role has been created to manage incidents with AWS Support	Account: 881145012897	FAILED	2 hours ago
<input type="checkbox"/>	LOW	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	CloudFormation.1 CloudFormation stacks should be integrated with Simple Notification Service (SNS)	AwsCloudFormationStack: d69ba740-180d-11ed-8ad0-121f1bbe5741	FAILED	5 hours ago
<input type="checkbox"/>	LOW	NEW	ACTIVE	us-east-1	881145012897	AWS	Security Hub	CloudFormation.1 CloudFormation stacks should be integrated with Simple Notification Service (SNS)	AwsCloudFormationStack: d69ba740-180d-11ed-8ad0-121f1bbe5741	FAILED	5 hours ago

Figure 11: Experiment 3 Vulnerabilities with low severity

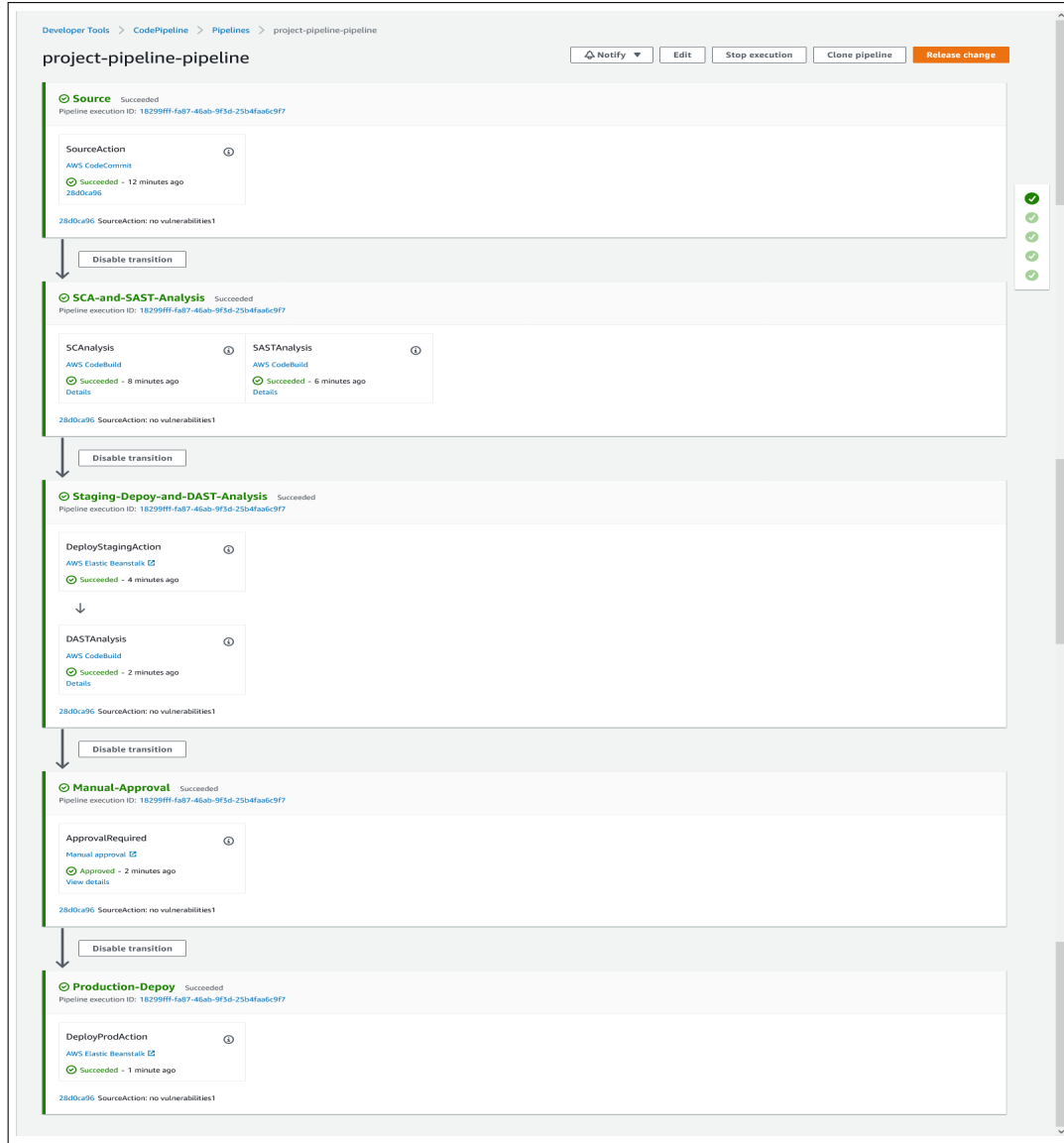


Figure 12: Experiment 3 Successful deployment of application for low vulnerabilities

## 7 Conclusion and Future Work

A manual and repetitive process can be automated with the help of technology and applying this context to DevOps while integrating security in it, this research paper tries to automate the manual task and abiding by the DevSecOps principles has achieved the desired result of detecting the static and dynamic vulnerabilities. Considering the traditional practices, adding the dynamic scanning tool and automating the system is the goal. The research paper originally based on the IEEE paper which talked about eliminating the vulnerabilities using the static scanning using sonarqube as the tool, has managed to reach the goal of dynamic scanning. In near future, I would like to include the malware and Trojan scanner to be integrated to be integrated with this system architecture as it would ensure complete obliteration of all the vulnerabilities.

## References

- Abhishek, M. K. and Rajeswara Rao, D. (2021). Framework to secure docker containers, *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pp. 152–156. Core Rank = not found.  
**URL:** <https://doi.org/10.1109/WorldS451998.2021.9514041>
- Akbar, M. A., Smolander, K., Mahmood, S. and Alsanad, A. (2022). Toward successful devsecops in software development organizations: A decision-making framework, *Information and Software Technology* **147**: 106894.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0950584922000568>
- Brady, K., Moon, S., Nguyen, T. and Coffman, J. (2020). Docker container security in cloud computing, *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0975–0980. Core Rank = not found.  
**URL:** <https://doi.org/10.1109/CCWC47524.2020.9031195>
- DuPaul, N. (2015). Static testing vs. dynamic testing.  
**URL:** <https://www.veracode.com/blog/secure-development/static-testing-vs-dynamic-testing>
- Garces, L., Martinez-Fernandez, S., Graciano Neto, V. V. and Nakagawa, E. Y. (2020). Architectural solutions for self-adaptive systems, *Computer* **53**(12): 47–59. JCR Impact Factor: 4.419.  
**URL:** <https://doi.org/10.1109/MC.2020.3017574>
- Garg, S. and Garg, S. (2019). Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security, *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, IEEE, New York, NY, USA, pp. 467–470. Core Rank = not found.  
**URL:** <https://doi.org/10.1109/MIPR.2019.00094>
- Javed, O. and Toor, S. (2021). Understanding the quality of container security vulnerability detection tools.  
**URL:** <https://doi.org/10.48550/arXiv.2101.03844>
- Lopes, N., Martins, R., Correia, M. E., Serrano, S. and Nunes, F. (2020). Container hardening through automated seccomp profiling, *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds*, WOC’20, Association for Computing Machinery, New York, NY, USA, pp. 31–36. Core Rank = not found.  
**URL:** <https://doi.org/10.1145/3429885.3429966>
- R.G.K.P.Kulathunga (2020). *Dynamic security model for container orchestration platform*, PhD thesis.  
**URL:** <https://dl.ucsc.cmb.ac.lk/jspui/handle/123456789/4533>
- Sarkale, V. V., Rad, P. and Lee, W. (2017). Secure cloud container: Runtime behavior monitoring using most privileged container (mpc), *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, IEEE, pp. 351–356. Core Rank = not found.  
**URL:** <https://doi.org/10.1109/CSCloud.2017.68>

- Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing, *Journal of Network and Computer Applications* **34**(1): 1–11.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S1084804510001281>
- Sultan, S., Ahmad, I. and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead, *IEEE Access* **7**: 52976–52996. JCR Impact Factor:4.48.  
**URL:** <https://doi.org/10.1109/ACCESS.2019.2911732>
- Suram, S., MacCarty, N. A. and Bryden, K. M. (2018). Engineering design analysis utilizing a cloud platform, *Advances in Engineering Software* **115**: 374–385. JCR Impact Factor: 3.884.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0965997817303733>
- Tomas, N., Li, J. and Huang, H. (2019). An empirical study on culture, automation, measurement, and sharing of devsecops, *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–8.
- Tunde-Onadele, O., He, J., Dai, T. and Gu, X. (2019). A study on container vulnerability exploit detection, *2019 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, pp. 121–127. Core Rank = not found.  
**URL:** <https://doi.org/10.1109/IC2E.2019.00026>
- Yang, N., Shen, W., Li, J., Yang, Y., Lu, K., Xiao, J., Zhou, T., Qin, C., Yu, W., Ma, J. and Ren, K. (2021). Demons in the shared kernel: Abstract resource attacks against os-level virtualization, *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21*, Association for Computing Machinery, New York, NY, USA, pp. 764–778. Core Rank = A\*.  
**URL:** <https://doi.org/10.1145/3460120.3484744>