Securing the End Points of Microservices using Gitlab Client-Based Authentication

MSc Industrial Internship
Cyber Security

Rohit Anand Ahuja
Student ID: 21168296

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rohit Anand Ahuja		
Student ID:	21168296		
Programme:	MSc in CyberSecurity	**Year:**	2022-23
Module:	MSc Industrial Internship		
Supervisor:	Vikas Sahni		
Submission Due Date:	06/01/2023		
Project Title:	Securing the End Points of Microservices using Gitlab Client-Based Authentication		
Word Count:	5927	**Page Count**	21

Signature: Rohit Anand Ahuja

Date: 06/01/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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):	

Securing the End Points of Microservices using Gitlab Client-Based Authentication

Rohit Anand Ahuja
21168296

Abstract

The integration of GitLab and Kubernetes has become increasingly important as organizations seek to streamline their application development and deployment processes. However, ensuring the connection between these two systems is secure is a key concern, as any vulnerabilities could potentially expose sensitive information or compromise the stability and security of the system. This research proposes a comprehensive study on how to securely integrate GitLab with an AWS Elastic Kubernetes Service (EKS) cluster for the purpose of managing cluster applications built on GitLab. The GitLab Kubernetes agent is proposed as a solution and can be configured using standard Kubernetes Role-Based Access Control (RBAC) rules to ensure secure access to the cluster. Additionally, the research aims to provide a comprehensive solution for implementing appropriate Kubernetes security controls and GitLab security to maintain a safe environment for the cluster. According to the research proposed and analysis, implementing certain design and security controls can decrease the number of potential attack vectors and reduce the risk exposure of deployed microservices using Kubernetes and GitLab.

1 Introduction

Cloud computing has become a dominant force in the way applications are designed, developed, and deployed. According to a recent Gartner Hype Cycle report (STAMFORD, 2022), cloud computing has reached the stage of Slope of Enlightenment, as the demands of the market have changed the landscape for software companies due to emerging technology and various factors such as business competitiveness, digital disruptions, and digital reinventions.

One technology that is widely used in cloud computing for the automated deployment of containerized applications is Kubernetes, a container orchestration platform. Kubernetes allows users to launch, scale, and manage containerized applications with ease. Modern development practices also rely on Continuous Integration and Continuous Delivery (CI/CD) tools like GitLab, which focus on creating and running tests for every commit to keep the test environment updated. Thus, the integration of GitLab and Kubernetes has become increasingly important as organizations seek to streamline their application development and deployment processes. Ensuring that the connection between these two systems is secure is a key concern, as any vulnerabilities could potentially expose sensitive information or compromise the stability and security of the system. In the past, certificate-based integration was used to connect Kubernetes clusters to GitHub for the deployment of containerized applications, but this method exposed direct access to the Kubernetes API to the outside world. According to a recent report (Montalbano, 2022) and (*Shodan Facet Analysis*, 2022),

around 380,000 Kubernetes APIs were exposed to the public internet. It is crucial to properly secure the Kubernetes API server to prevent unauthorized access and protect against malicious actors modifying the state of the cluster. If the Kubernetes API server is exposed to the public internet without proper protection, it could potentially be accessed by anyone with an internet connection, allowing attackers to gain access to sensitive information, perform unauthorized actions, or compromise the stability and security of the cluster. Additionally, the certificate-based integration with Kubernetes adds additional attack surfaces.
In this research, a comprehensive study on how to securely integrate GitLab with an AWS Elastic Kubernetes Service (EKS) cluster for the purpose of managing cluster applications built on GitLab has been proposed. To achieve this, the GitLab Kubernetes agent can be used. It can be configured using standard Kubernetes Role-Based Access Control (RBAC) rules to ensure secure access to the cluster. Additionally, it is important to implement appropriate Kubernetes security controls to maintain a safe environment for the cluster. The research aims to provide a comprehensive solution for securely integrating GitLab with EKS clusters, ensuring the stability and security of the cluster and the applications it hosts.

1.1 Research Question

Taking into consideration the crucial aspect investigated in the research problems, the primary research question for the study would be:

- ***How can an application based on microservices be securely deployed in a cloud-native environment using Kubernetes and Gitlab*?**

This question aims to explore the use of Kubernetes and GitLab for deploying microservices-based applications in a cloud-native environment in a secure manner. The research will explore tools and approaches that can be used to facilitate the integration of GitLab and Kubernetes, and it will examine the benefits and challenges of using these two systems together. The goal is to provide a comprehensive solution for securely deploying microservices-based applications in a cloud-native environment using Kubernetes and GitLab.

1.2 Paper Structure

The remaining structure of this research paper is as follows:
In Section 2, related work on the topics of microservices and Kubernetes is discussed. Section 3 outlines the methodology for implementing the proposed approach. Section 4 presents the design specification of the given approach. Section 5, provide a detailed description of the implementation process. Finally, in Section 6 and 7, the evaluation results and conclusions based on findings is discussed.

2 Related Work

In recent years, there has been a growing interest in using GitLab and Kubernetes together to manage and deploy containerized applications. In this section, the related work on Kubernetes, GitLab, Kubernetes security, and microservices will be reviewed.

(Sultan et al., 2019), the authors have discussed how much container security is important and what are the issues currently faced during the container security and how the issues can be remediated. The research helped to understand the container security requirements and get a better understanding of potential attacks and vulnerabilities. (Poniszewska-Marańda and Czechowska, 2021), the authors have discussed methods for deploying application to the Kubernetes clusters in cloud for automating software production environment. The paper has focused on fulfilling the needs of the production environment, also comparing two methods of deploying the Kubernetes cluster in the cloud.

(Bose, Rahman and Shamim, 2021) authors discussed the under reported security defects in Kubernetes which are very important to be addressed otherwise such security flaws could inspire attackers, which could have detrimental effects. (Дарвеш, Хаммуд and Воробьева, 2022), the authors have discussed based on practitioner reports, there are 10 security best practices such as RBAC, namespace separation in the Kubernetes cluster that users should adhere to in order to safeguard the infrastructure. (Giangiulio and Malmberg, 2022), the author have discussed various attacks which can be carried out on the Kubernetes cluster running containerized application, notably on an AKS platform. and how these attacks can be saved by providing recommended solutions. (Agrawal and Abhijeet, 2020), the authors have discussed the different security vulnerabilities that might occur related to Kubernetes-based container deployments and outlined the best practices to adopt when setting up the infrastructure for deployment using a set of guidelines to allow businesses to run cloud-based systems without being concerned about cyber security attacks.

Another study that purposed a solution for securing a Kubernetes cluster on google cloud platform (Autio, 2021), the authors have discussed the steps for securing the Kubernetes cluster, also how the databases can be secured and how the cost can be reduced.(Dakic, Redzepagic and Basic, 2022), the authors have focused on the security of CI/CD which is important when developing a cloud infrastructure. (Sinde *et al.*, 2022) proposed how the application can be deployed in the Kubernetes cluster using GitHub actions as a continuous integration and delivery solution but the authors have focused less on the security purpose.(Gupta *et al.*, 2022), the authors have discussed the benefits of implementing GitOps in the Kubernetes environment and how to implement Kubernetes on AWS also how Git's adoption standardizes the Git-centric workflow, boosts output by enabling continuous delivery and deployment, improves dependability due to the rollback features, and promotes visibility.

(Ramadoni, Utami and Fatta, 2021), the authors have used Argo CD for pull based deployments and Kubernetes as a platform for deploying container-based applications. The authors have focused more on the deployment than the security of the container-based applications. (Felix, Garg and Dikaleh, 2019), the authors have focused on the security of the kubernetes clusters, a thorough knowledge of the interrelationships between Role Based Access Control (RBAC), resource quotas, pod security rules, namespaces, network policies, and image generation and scanning policies to provide a safer environment for operating shared services. (Throner *et al.*, 2021) have discussed the steps to setup the DevOps environment and have focused on the CI/CD pipeline and the security of the environment against different attack vectors, however there is less focus on secure connection between CI\CD pipeline and DevOps environment.

3 Research Methodology

In this section, the method used for conducting the research is described. The steps followed in the research process are outlined in section 3.1.

In this study, a technique for establishing a secure connection between a Kubernetes cluster comprising microservices using Gitlab client authentication is presented, specifically the Gitlab Kubernetes Agent. The primary focus is on securing the connection between Gitlab and the Kubernetes cluster, with a secondary focus on the separating the application build and application environment projects. This research proposes a solution to the problem of exposing the Kubernetes Control API to the network in order to connect with GitLab for push-based approaches. The proposed method also enhances the performance and security of the application by separating the application build project and application environment project into different GitLab repositories, and conducting security testing and container testing to identify vulnerabilities in the application.

3.1 Research Flow

For the purpose of this research, a Kubernetes cluster was deployed on AWS Elastic Kubernetes Service (EKS), GitLab agent was deployed on Kubernetes cluster and three GitLab repositories were utilized: one for Kubernetes cluster management, one for application build, and one for application environment. The research flow is depicted in the below figure:

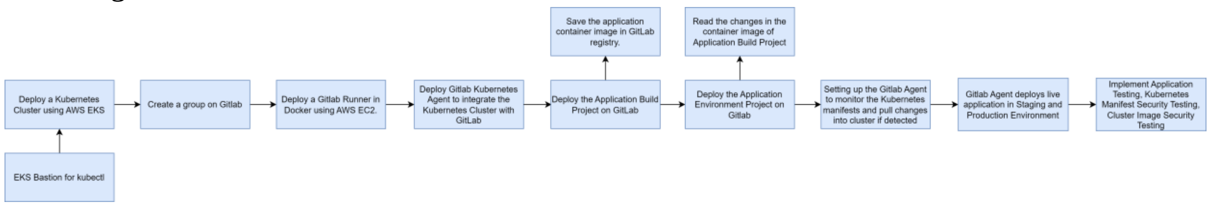

Save the application container image in GitLab registry.
Read the changes in the container image of Application Build Project
Deploy a Kubernetes Cluster using AWS EKS
Create a group on Gitlab
Deploy a Gitlab Runner in Docker using AWS EC2.
Deploy Gitlab Kubernetes Agent to integrate the Kubernetes Cluster with GitLab
Deploy the Application Build Project on GitLab
Deploy the Application Environment Project on Gitlab
Setting up the Gitlab Agent to monitor the Kubernetes manifests and pull changes into cluster if detected
Gitlab Agent deploys live application in Staging and Production Environment
Implement Application Testing, Kubernetes Manifest Security Testing, Cluster Image Security Testing
EKS Bastion for kubectl

Figure 1: Research Methodology

4 Design Specification

This section covers the design specifications for the research. It includes a discussion of the system requirements for running the implementation (in Subsection 4.1) and a description of the proposed system architecture (in Subsection 4.2).

4.1 System Specifications

In this section, the outline for required specifications of the system that is being developed is described. These specifications will ensure that the system is able to function effectively and meet the needs of its intended users. It is important to carefully consider and define these requirements upfront, as they will serve as a guide for the design and development of the system.

Kubernetes Cluster on AWS EKS	
EC2 type	t3.micro
No. of nodes	2
Operating System	Amazon Linux 2
Kubernetes Version	v1.21
Memory	10 GB
vCPU	2
RAM	1.0 GB

Docker on AWS	
EC2 type	m5.large
Operating System	Amazon Linux 2
Docker Version	20.10.17
Memory	10 GB
vCPU	2
RAM	8.0 GB

4.2 Proposed System Architecture

The proposed system architecture identifies the technology stack for installing and configuring the GitLab agent in a Kubernetes cluster. This design includes four key components: AWS EC2- Bastion Host, AWS Elastic Kubernetes Service, a Kubernetes cluster, and the GitLab agent itself.

The EC2 Bastion host was deployed to access and manage the Kubernetes cluster remotely securely, such as a laptop or desktop. This is particularly useful when the Kubernetes cluster runs in a private subnet and is not directly accessible from the internet. Second, the AWS Elastic Kubernetes Service (EKS) offers a fully managed solution for deploying, scaling and managing Kubernetes clusters in the cloud. EKS simplifies running a Kubernetes cluster by automating many underlying tasks and providing built-in monitoring and logging capabilities. Kubernetes clusters have been used to create a group of virtual machines that are used to run containerized applications. Kubernetes clusters is consisting of at least one master node, which is responsible for overseeing the cluster, and 4 worker nodes that host and run containerized applications. This structure is used for the efficient deployment, scaling, and management of containerized applications within the cluster.

The application build project and application environment project were kept in different GitLab repositories so that it is easy to test and deploy, it improves scalability and enhances security. For this research, GitLab agent was deployed in one of the Kubernetes Pod situated in deployment namespace which is responsible for continuously monitoring the git repository and if any changes are made in git repository, the agent will do a git pull and deploy the application in the containerized application securely without exposing the Kubernetes API and maintaining RBAC rules.

The Gitlab agent has been deployed considering some potential points:

- **Integration with GitLab:** The agent is deployed in such way that it is able to connect to a GitLab instance and communicate with it in order to trigger deployments and other actions based on events in the GitLab workflow.
- **Management of applications:** The agent is able to deploy and manage applications in a Kubernetes cluster as it is been deployed in one of the deployment namespaces pods.
- **Monitoring and logging:** The agent provide monitoring and logging capabilities to help track the status and performance of deployed applications, as well as troubleshoot any issues that may arise.
- **Security:** The agent is installed with security in mind, including measures to protect against unauthorized access and ensure the confidentiality and integrity of data transmitted between the agent and GitLab or other systems.

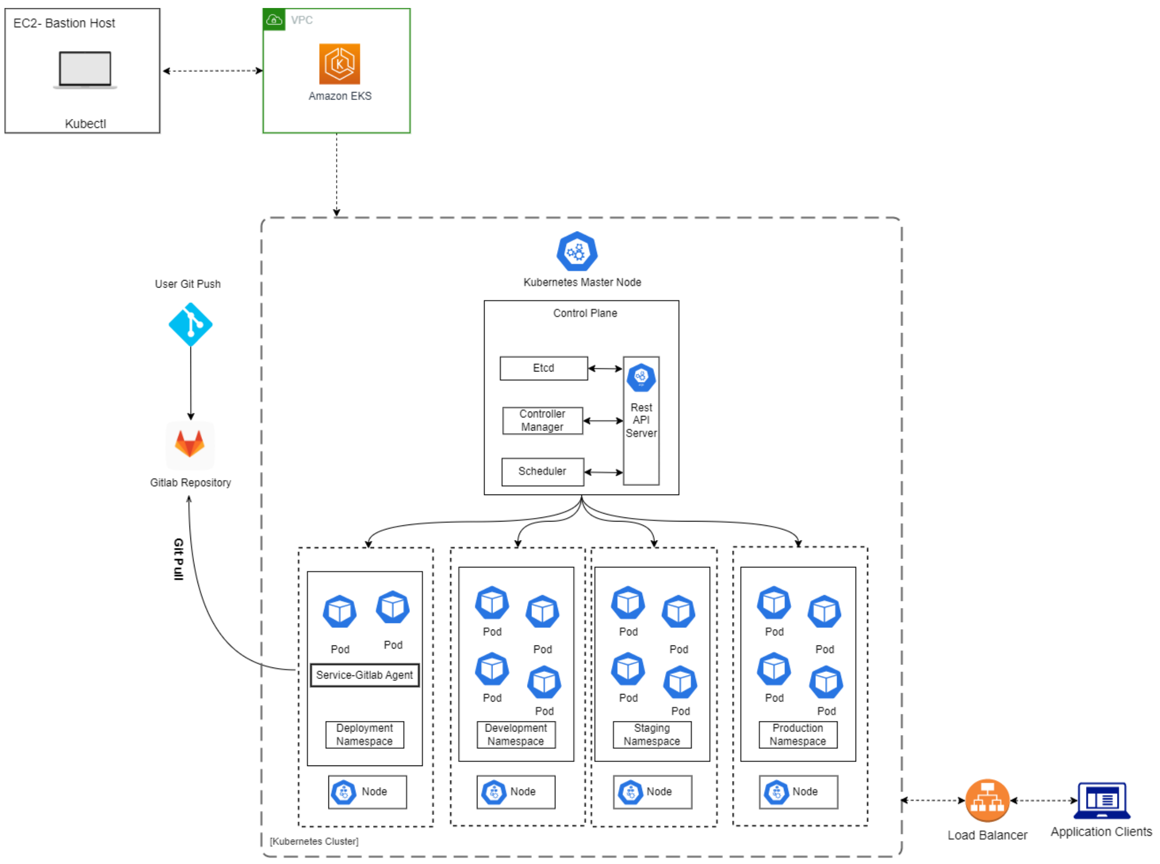


Figure 2: Proposed System Architecture

5 Implementation

In this section, the implementation and the steps taken to carry out the study and achieve the research objections is discussed. The proposed system architecture was completed with the implementation figure depicted below, taking into account security aspects. This means that the system architecture was designed with security in mind and includes measures to protect the system from potential security threats or vulnerabilities. The implementation figure shows how the various components of the system are integrated and how they work together to provide the desired functionality while also maintaining a high level of security. This figure includes details such as the software used, the network architecture, and the roles and responsibilities of different system components. By considering security aspects in the design of the system architecture, the proposed system aims to provide a secure and reliable solution that meets the needs and requirements of the users or an organization for deploying secure microservices.

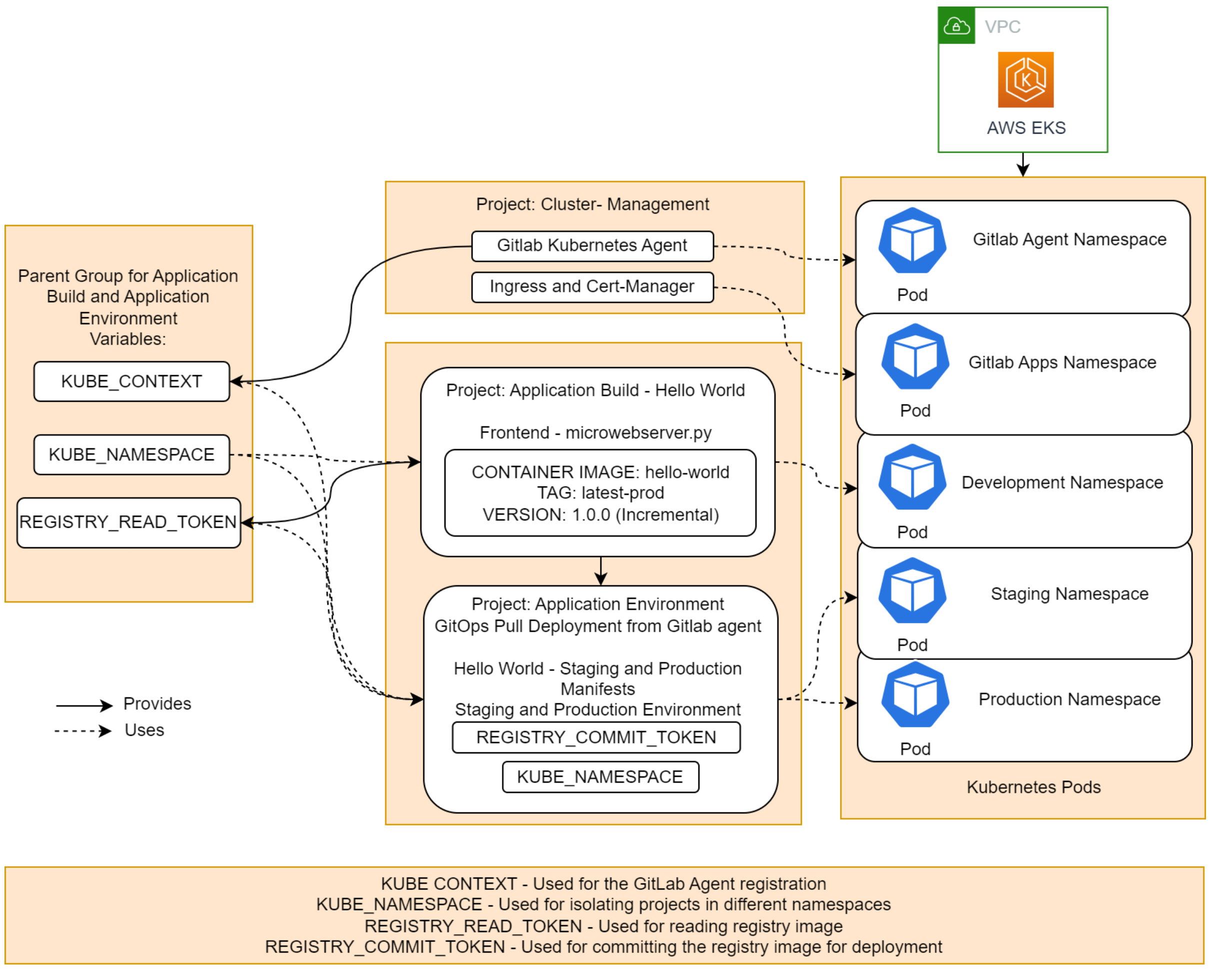

VPC
AWS EKS
Project: Cluster- Management
Gitlab Kubernetes Agent
Ingress and Cert-Manager
Parent Group for Application Build and Application Environment Variables:
KUBE_CONTEXT
KUBE_NAMESPACE
REGISTRY_READ_TOKEN
Project: Application Build - Hello World
Frontend - microwebserver.py
CONTAINER IMAGE: hello-world
TAG: latest-prod
VERSION: 1.0.0 (Incremental)
Project: Application Environment
GitOps Pull Deployment from Gitlab agent
Hello World - Staging and Production Manifests
Staging and Production Environment
REGISTRY_COMMIT_TOKEN
KUBE_NAMESPACE
Gitlab Agent Namespace
Pod
Gitlab Apps Namespace
Pod
Development Namespace
Pod
Staging Namespace
Pod
Production Namespace
Pod
Kubernetes Pods
Provides
Uses
KUBE CONTEXT - Used for the GitLab Agent registration
KUBE_NAMESPACE - Used for isolating projects in different namespaces
REGISTRY_READ_TOKEN - Used for reading registry image
REGISTRY_COMMIT_TOKEN - Used for committing the registry image for deployment

Figure 3: Implemented System Architecture

5.1 Tools and Technologies Used

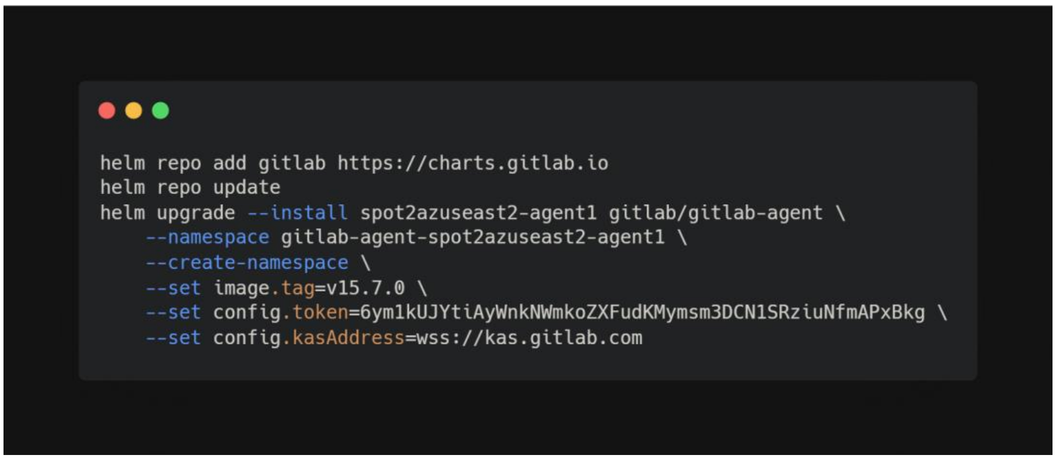
For this research, the implementation was carried out using two Amazon Web Services (AWS) EC2 instances: **a t3.micro EKS Bastion (Linux) and a m5.large Linux Docker**, as well as an **EKS cluster with Kubernetes v1.21**. The Gitlab Kubernetes Agent was deployed in one of the Kubernetes cluster pods. The microservice application was developed using frontend HTML, and Python was used as the micro server backend.

Demonstrating the technologies and tools used:

- **AWS Elastic Kubernetes Service (EKS)**: Kubernetes v1.21 has been deployed using AWS EKS, which automatically deploys a Kubernetes cluster.
- **Kubectl:** kubectl is a command-line tool for interacting with a Kubernetes cluster. kubectl allows you to run commands against a Kubernetes cluster, such as deploying applications, inspecting the status of components, and viewing logs. EC2 Bastion Host is deployed for firing kubectl commands.
- **Docker**: Docker is used to run and deploy GitLab runner for the research Also, docker is used for microservices-based applications, enabling to scale and update individual services independently and deploy the overall application with all necessary dependencies to different environments. Docker version 20.10.17 has been used for the research.
- **GitLab**: GitLab platform is used for storing and managing code repositories (using Git technology) through a web interface.
- **GitLab Runner**: GitLab Runner is a tool used to run jobs and execute scripts in GitLab as part of CI/CD pipelines and has been configured with AWS for the research.
- **GitLab Kubernetes Agent**: To securely connect Kubernetes cluster with GitLab without exposing the Kubernetes API, we have deployed the GitLab Kubernetes Agent. This agent ensures that the connection between the cluster and GitLab is secure and that only authorized users and processes have access to the cluster maintaining the RBAC rules.
- **HTTP and Python**: A simple HTTP page has been set up as a frontend using Python HTTP server.
- **YAML**: YAML manifests have been used for defining state of resources, such as pods, services, and deployment strategies, allowing easy deployment and management of applications.

5.2 Deploying the Gitlab Agent for Kubernetes.

After the deployment of the EKS and Docker, the Gitlab Agent for Kubernetes is deployed in one of the pods of the Kubernetes Cluster. To deploy the Agent, the following commands were fired into the EKS Bastion Host:



```
helm repo add gitlab https://charts.gitlab.io
helm repo update
helm upgrade --install spot2azuseast2-agent1 gitlab/gitlab-agent \
    --namespace gitlab-agent-spot2azuseast2-agent1 \
    --create-namespace \
    --set image.tag=v15.7.0 \
    --set config.token=6ym1kUJYtiAyWnkNWmkoZXFudKMymsm3DCN1SRziuNfmAPxBkg \
    --set config.kasAddress=wss://kas.gitlab.com
```

Figure 4: Code to Deploy Gitlab Kubernetes Agent

This code will add the GitLab Helm repository to the local Helm configuration, update the local repository list to include the latest versions of the charts from the GitLab repository, and then install a Helm release called "spot2azuseast2-agent1" from the GitLab chart for the GitLab Agent.

The upgrade command includes several flags and arguments:

- **--install**: This flag tells Helm to install the chart if it is not already installed, or upgrade it if it is already installed.
- **--namespace gitlab-agent-spot2azuseast2-agent1**: This flag specifies the namespace in which the GitLab Agent should be deployed.
- **--create-namespace**: This flag tells Helm to create the specified namespace if it does not already exist.
- **--set image.tag=v15.7.0**: This flag sets the image tag for the GitLab Agent Docker image. This determines which version of the GitLab Agent will be deployed.
- **--set config.token=6ym1kUJYtiAyWnkNWmkoZXFudKMymsm3DCN1SRziuNfmAPxBkg**: This flag sets the GitLab token that will be used to authenticate the GitLab Agent with the GitLab instance.
- **--set config.kasAddress=wss://kas.gitlab.com**: This flag sets the address of the Kubernetes API Server (KAS) that the GitLab Agent will use to communicate with the Kubernetes cluster.

5.3 Isolating the Application Build and Application Environment.

To ensure the security and isolation of the "Application Build - Hello World" and "Application Environment - Hello World" projects, separate nodes were chosen for their deployments. This allowed the projects to run independently and potentially scale differently, while also preventing potential security vulnerabilities or resource contention between the two. In this approach, the "Application Build" repository contains the source code and build configuration for the application, while the "Application Environment" repository contains the configuration and resources needed to deploy the application to a Kubernetes Cluster.

5.4 Running the Pipeline and Deploying the Application.

The “Application Build – Hello World” runs the CI/CD pipeline and as part of this process, the pipeline builds a container image for the application. This registry can be accessed by the application environment in order to retrieve the necessary container image for deployment. The container image contains all of the code, dependencies, and other resources required to run the application, and is stored in a portable package that can be easily deployed by the application environment. By using the container image stored in the registry, the application environment can ensure that it is deploying the correct and updated version of the application.

The "Application Environment - Hello World" is responsible for deploying the application to a live staging or production environment by running a CI/CD pipeline. To do this, it uses a GitLab Kubernetes Agent which was deployed in one of the Kubernetes pods. The GitLab Kubernetes Agent uses a pull-based approach to deploy applications to the Kubernetes cluster. This means that the Agent sends a request to the Kubernetes cluster to pull the necessary container image from a container registry and deploy the application within the cluster. To deploy the application, the "Application Environment - Hello World" will use a configuration file written in the YAML (YAML Ain't Markup Language) format. This file specifies the details of the deployment, such as the container image to use and the resources that the application will need. To retrieve the container image for the application, the "Application Environment - Hello World" will access the container registry where the image is stored. Once the container image has been retrieved from the registry and the YAML configuration file has been applied, the "Application Environment - Hello World" will use the GitLab Kubernetes Agent to deploy the application live in the staging and production environment.

Overall, isolating the build and environments can also help to improve the overall security posture of the application by making it easier to apply security patches and updates. By isolating the build environment, developers can update and test the patches and updates without affecting the running application, which can help to reduce the risk of downtime or other disruptions caused by security issues.

5.5 Adding the Security Scanning for Application, Kubernetes Manifests and Container.

Application Build Security Scanning was added to ensure that the integrity and stability of the application remains intact by identifying and addressing potential vulnerabilities and to Protect sensitive data from being accessed or compromised by unauthorized parties. Kubernetes Manifests security was added to ensure that they are configured in a secure manner. This includes checking for the use of secure protocols, the use of strong passwords or keys, and the proper configuration of access controls.

Container Scanning was done with the help of Gitlab Agent for Kubernetes. Using the GitLab Agent for Kubernetes to scan container images helped ensuring that the security and stability of the Kubernetes cluster remains intact by identifying and addressing potential security vulnerabilities before they can be exploited. It also helped to comply with security and compliance standards by ensuring that only secure container images are deployed in the Kubernetes cluster.

6 Evaluation

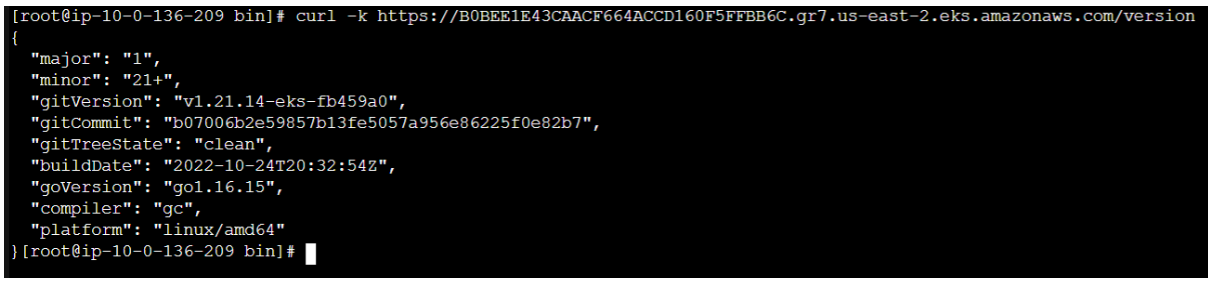
For the evaluation, the assessment of the GitLab Kubernetes Agent, including authentication, pull-based approach, environmental isolation, and security scanning features, has been done. The effectiveness of these features was determined and their suitability for use in various scenarios was evaluated. The aim was to provide a comprehensive evaluation of the GitLab Kubernetes Agent and to provide insights and recommendations for its use in managing applications on a Kubernetes cluster.

6.1 Case Study 1: Control Node API Exposure

There are several potential risks associated with using the old method i.e., certificate-based integration for accessing the Kubernetes API. One of the main risks is that it relies on direct access to the API which must be set to global IP and on public internet, and that's a big security vulnerability as discussed earlier.

To mitigate this risk, the Gitlab Kubernetes Agent has been used in the proposed architecture. Using the GitLab Kubernetes Agent can help to mitigate the risks associated with exposing the Kubernetes API. Gitlab Kubernetes Agent has been installed in of the Kubernetes cluster pod, where there is no need to exposed the API to the public internet, which can remove the risk of unauthorized access to the cluster and the resources within it. Also, the Kubernetes cluster has been installed in an AWS VPC which has been created with an aspect of a secure, isolated network environment that is separate from the public internet.

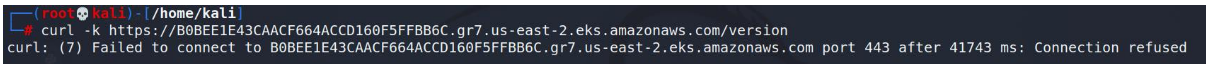
Deployed Kubernetes API: **https://B0BEE1E43CAACF664ACCD160F5FFBB6C.gr7.us-east-2.eks.amazonaws.com/**
Accessing Kubernetes API through EKS Bastion Host, i.e., Internal Network:



```
[root@ip-10-0-136-209 bin]# curl -k https://B0BEE1E43CAACF664ACCD160F5FFBB6C.gr7.us-east-2.eks.amazonaws.com/version
{
  "major": "1",
  "minor": "21+",
  "gitVersion": "v1.21.14-eks-fb459a0",
  "gitCommit": "b07006b2e59857b13fe5057a956e86225f0e82b7",
  "gitTreeState": "clean",
  "buildDate": "2022-10-24T20:32:54Z",
  "goVersion": "go1.16.15",
  "compiler": "gc",
  "platform": "linux/amd64"
}[root@ip-10-0-136-209 bin]#
```

Figure 5: Accessing Kubernetes API through EKS Bastion Host

Accessing Kubernetes API through Kali OS i.e., External Network:



```
┌──(root💀kali)-[/home/kali]
└─# curl -k https://B0BEE1E43CAACF664ACCD160F5FFBB6C.gr7.us-east-2.eks.amazonaws.com/version
curl: (7) Failed to connect to B0BEE1E43CAACF664ACCD160F5FFBB6C.gr7.us-east-2.eks.amazonaws.com port 443 after 41743 ms: Connection refused
```

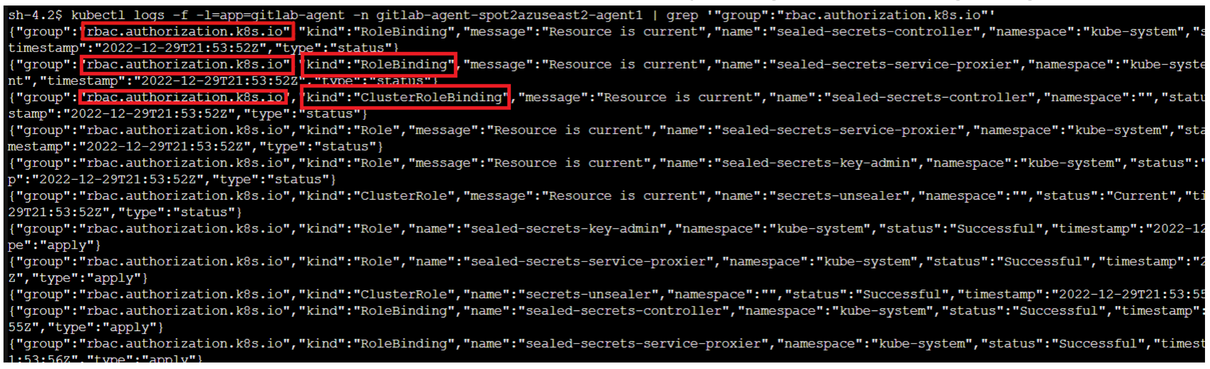
Figure 6: Accessing Kubernetes API through Kali OS

6.2 Case Study 2: Agent Authentication

For the Agent Authentication, the AAA (Accounting, Authentication and Authorization) method has been used in the research. The AAA (Accounting, Authentication, and Authorization) method is a security model that is used to verify the identity of a user or system and grant it access to resources based on its permissions.

In the context of the GitLab Kubernetes Agent, the AAA method involves the following steps:
Authentication: The GitLab Kubernetes Agent authenticates using a Kubernetes Service Account token. This token is used to verify the identity of the agent and establish its credentials. **Authorization**: The agent's permissions are determined by the RoleBinding that is created when setting up the GitLab Kubernetes Agent. This RoleBinding grants the agent the necessary permissions to access the resources on the cluster, such as pods, services, and deployments. **Accounting**: The actions performed by the GitLab Kubernetes Agent on the cluster are tracked and logged to ensure accountability and visibility.

The authentication and authorization can be viewed by using the 'kubectl get logs' command:



```
sh-4.2$ kubectl logs -f -l=app=gitlab-agent -n gitlab-agent-spot2azuseast2-agent1 | grep '"group":"rbac.authorization.k8s.io"'
{"group":"rbac.authorization.k8s.io" "kind":"RoleBinding","message":"Resource is current","name":"sealed-secrets-controller","namespace":"kube-system","s
timestamp":"2022-12-29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io" "kind":"RoleBinding","message":"Resource is current","name":"sealed-secrets-service-proxier","namespace":"kube-syste
nt","timestamp":"2022-12-29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io","kind":"ClusterRoleBinding","message":"Resource is current","name":"sealed-secrets-controller","namespace":"","statu
stamp":"2022-12-29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io","kind":"Role","message":"Resource is current","name":"sealed-secrets-service-proxier","namespace":"kube-system","sta
mestamp":"2022-12-29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io","kind":"Role","message":"Resource is current","name":"sealed-secrets-key-admin","namespace":"kube-system","status":"
p":"2022-12-29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io","kind":"ClusterRole","message":"Resource is current","name":"secrets-unsealer","namespace":"","status":"Current","ti
29T21:53:52Z","type":"status"}
{"group":"rbac.authorization.k8s.io","kind":"Role","name":"sealed-secrets-key-admin","namespace":"kube-system","status":"Successful","timestamp":"2022-12
pe":"apply"}
{"group":"rbac.authorization.k8s.io","kind":"Role","name":"sealed-secrets-service-proxier","namespace":"kube-system","status":"Successful","timestamp":"2
Z","type":"apply"}
{"group":"rbac.authorization.k8s.io","kind":"ClusterRole","name":"secrets-unsealer","namespace":"","status":"Successful","timestamp":"2022-12-29T21:53:55
{"group":"rbac.authorization.k8s.io","kind":"RoleBinding","name":"sealed-secrets-controller","namespace":"kube-system","status":"Successful","timestamp":
55Z","type":"apply"}
{"group":"rbac.authorization.k8s.io","kind":"RoleBinding","name":"sealed-secrets-service-proxier","namespace":"kube-system","status":"Successful","timest
1:53:56Z","type":"apply"}
```

Figure 7: Agent Authentication

The figure shows that the agent first checks the permissions before proceeding with any further actions. If the necessary permissions are granted, the agent will continue with the intended actions.

6.3 Case Study 3: Pull-Based GitOps

Using the Kubernetes API to update the Kubernetes cluster through GitLab CI/CD's push-based approach carries potential security risks. One of the main risks is that the K8s credentials used to access the API could be exposed in GitLab, potentially allowing unauthorized access to your cluster. Additionally, storing the kubeconfig file containing the

API on the GitLab server side could also pose a security risk, as it could potentially be accessed by anyone with access to the repository if the file is accidentally committed and pushed to a public repository.

In the research setup, a pull-based approach has been implemented. The GitLab agent, located within the cluster, initiates change from within the cluster to avoid exposing cluster-admin level credentials to GitLab.com. The GitLab agent is responsible for pulling updates from the GitLab repository and applying them to the cluster. This ensures that the Kubernetes cluster-admin level credentials are not exposed in GitLab. The agent also reads updates from the YAML manifests file stored in the "Application Environment" project, and if any updates are found, it pulls and applies them to the Kubernetes cluster. The figure below illustrates the method clearly:

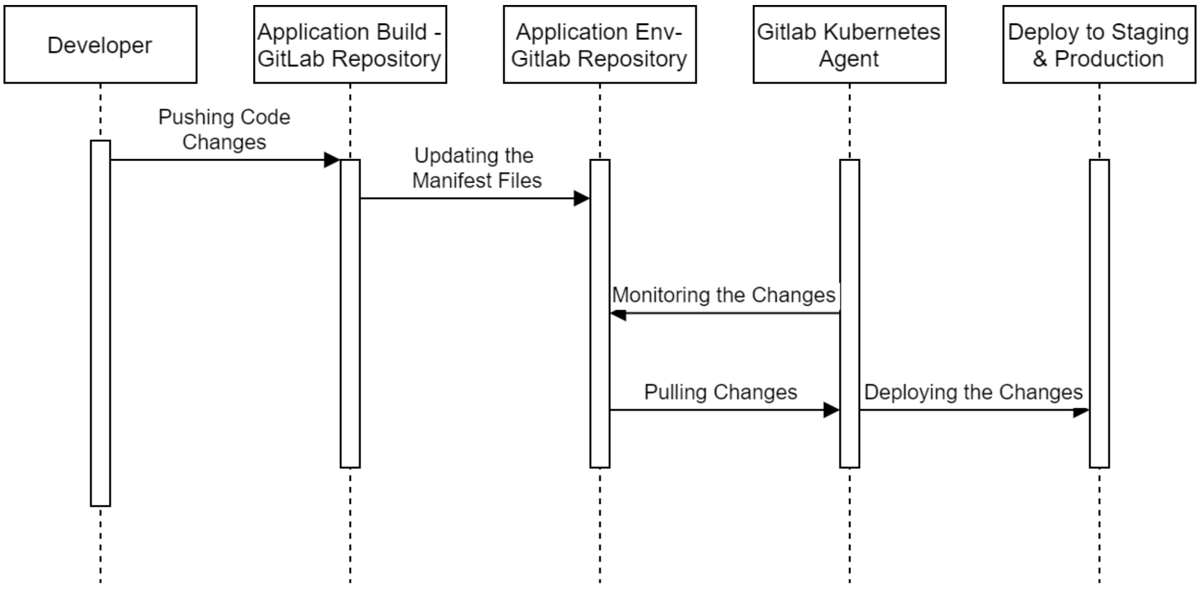
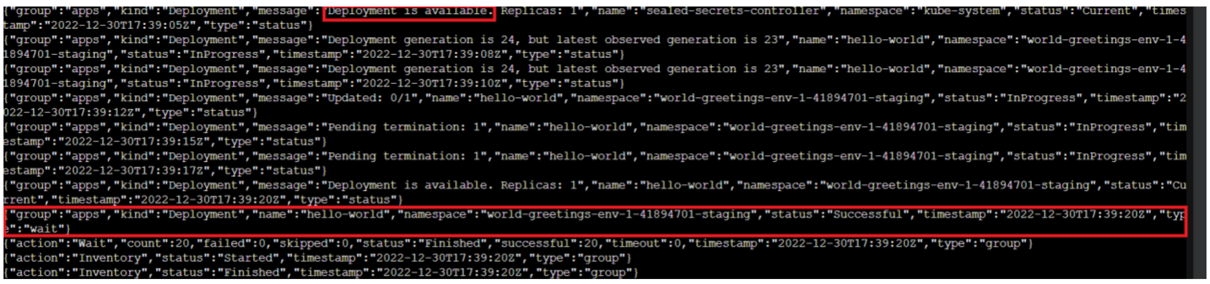

Developer
Application Build - GitLab Repository
Application Env- Gitlab Repository
Gitlab Kubernetes Agent
Deploy to Staging & Production
Pushing Code Changes
Updating the Manifest Files
Monitoring the Changes
Pulling Changes
Deploying the Changes

Figure 8:Pull-Based Approach



```
{"group":"apps","kind":"Deployment","message":"Deployment is available. Replicas: 1","name":"sealed-secrets-controller","namespace":"kube-system","status":"Current","times
tamp":"2022-12-30T17:39:05Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Deployment generation is 24, but latest observed generation is 23","name":"hello-world","namespace":"world-greetings-env-1-4
1894701-staging","status":"InProgress","timestamp":"2022-12-30T17:39:08Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Deployment generation is 24, but latest observed generation is 23","name":"hello-world","namespace":"world-greetings-env-1-4
1894701-staging","status":"InProgress","timestamp":"2022-12-30T17:39:10Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Updated: 0/1","name":"hello-world","namespace":"world-greetings-env-1-41894701-staging","status":"InProgress","timestamp":"2
022-12-30T17:39:12Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Pending termination: 1","name":"hello-world","namespace":"world-greetings-env-1-41894701-staging","status":"InProgress","tim
estamp":"2022-12-30T17:39:15Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Pending termination: 1","name":"hello-world","namespace":"world-greetings-env-1-41894701-staging","status":"InProgress","tim
estamp":"2022-12-30T17:39:17Z","type":"status"}
{"group":"apps","kind":"Deployment","message":"Deployment is available. Replicas: 1","name":"hello-world","namespace":"world-greetings-env-1-41894701-staging","status":"Cu
rrent","timestamp":"2022-12-30T17:39:20Z","type":"status"}
{"group":"apps","kind":"Deployment","name":"hello-world","namespace":"world-greetings-env-1-41894701-staging","status":"Successful","timestamp":"2022-12-30T17:39:20Z","typ
e":"wait"}
{"action":"Wait","count":20,"failed":0,"skipped":0,"status":"Finished","successful":20,"timeout":0,"timestamp":"2022-12-30T17:39:20Z","type":"group"}
{"action":"Inventory","status":"Started","timestamp":"2022-12-30T17:39:20Z","type":"group"}
{"action":"Inventory","status":"Finished","timestamp":"2022-12-30T17:39:20Z","type":"group"}
```

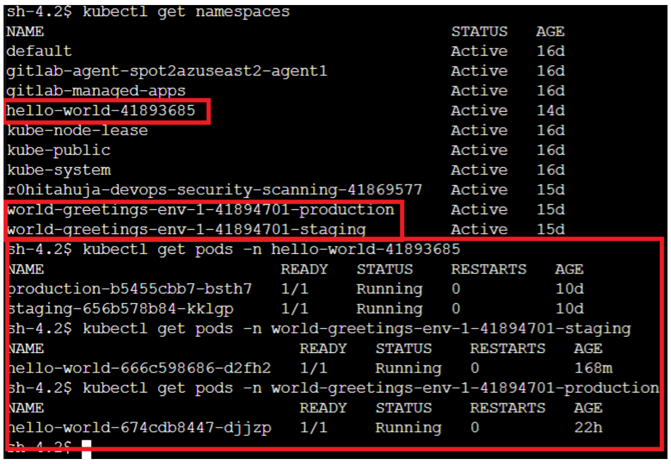
Figure 9: Gitlab Agent Pulling Changes

From the above logs, it can be seen that the agent is pulling changes and deploying the application into staging environment successfully.

6.4 Case Study 4: Environment Isolation

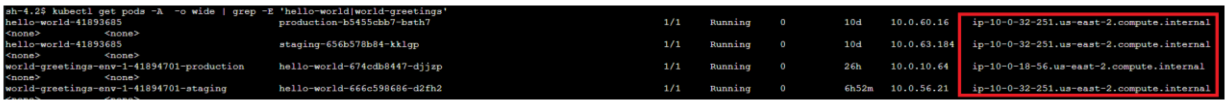
The “Application Build – Hello World” and “Application Environment – Hello World” are both isolated from each other. To ensure the security and isolation of the "Application Build -

Hello World" and "Application Environment - Hello World" projects, separate nodes were chosen for their deployments.



```
sh-4.2$ kubectl get namespaces
NAME                                              STATUS   AGE
default                                           Active   16d
gitlab-agent-spot2azuseast2-agent1                Active   16d
gitlab-managed-apps                               Active   16d
hello-world-41893685                              Active   14d
kube-node-lease                                   Active   16d
kube-public                                       Active   16d
kube-system                                       Active   16d
r0hitahuja-devops-security-scanning-41869577      Active   15d
world-greetings-env-1-41894701-production         Active   15d
world-greetings-env-1-41894701-staging            Active   15d
sh-4.2$ kubectl get pods -n hello-world-41893685
NAME                             READY   STATUS    RESTARTS   AGE
production-b5455cbb7-bsth7       1/1     Running   0          10d
staging-656b578b84-kklgp         1/1     Running   0          10d
sh-4.2$ kubectl get pods -n world-greetings-env-1-41894701-staging
NAME                               READY   STATUS    RESTARTS   AGE
hello-world-666c598686-d2fh2       1/1     Running   0          168m
sh-4.2$ kubectl get pods -n world-greetings-env-1-41894701-production
NAME                               READY   STATUS    RESTARTS   AGE
hello-world-674cdb8447-djjzp       1/1     Running   0          22h
sh-4.2$
```

Figure 10: Isolated Pods



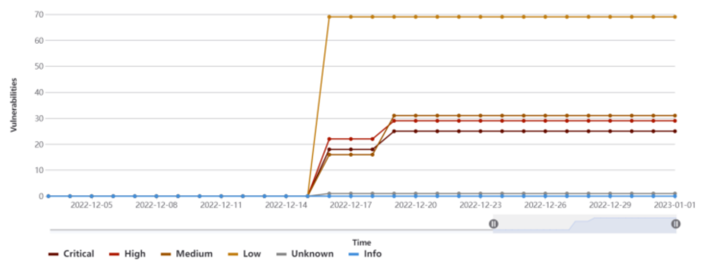
```
sh-4.2$ kubectl get pods -A  -o wide | grep -E 'hello-world|world-greetings'
hello-world-41893685                        production-b5455cbb7-bsth7        1/1   Running   0   10d     10.0.60.16    ip-10-0-32-251.us-east-2.compute.internal
<none>           <none>
hello-world-41893685                        staging-656b578b84-kklgp          1/1   Running   0   10d     10.0.63.184   ip-10-0-32-251.us-east-2.compute.internal
<none>           <none>
world-greetings-env-1-41894701-production   hello-world-674cdb8447-djjzp      1/1   Running   0   26h     10.0.10.64    ip-10-0-18-56.us-east-2.compute.internal
<none>           <none>
world-greetings-env-1-41894701-staging      hello-world-666c598686-d2fh2      1/1   Running   0   6h52m   10.0.56.21    ip-10-0-32-251.us-east-2.compute.internal
```

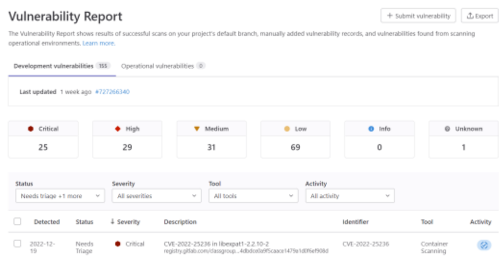
Figure 11: Isolated Nodes

The above figure shows pods were deployed in different nodes for the “Application Build – Hello World” and “Application Environment – Hello World” GitLab repositories.

6.5 Case Study 5: Security Scanning using GitOps

The GitLab platform offers a feature called a "dashboard" where developers can see a report that details any vulnerabilities present in their project. By reviewing this report, the developers can identify and fix any issues before the project is deployed as a product. This helps to ensure that the application is secure and free of vulnerabilities when it is released to users.





Vulnerability Report

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 155 | Operational vulnerabilities 0

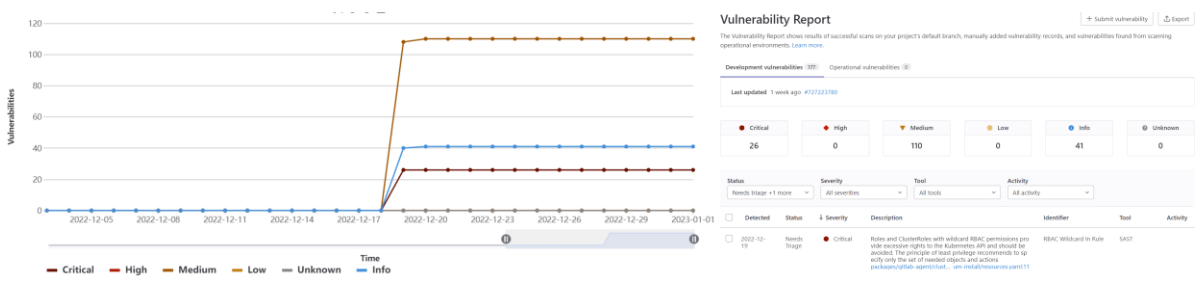
Last updated 1 week ago #727266340

Critical	High	Medium	Low	Info	Unknown
25	29	31	69	0	1

Status: Needs triage +1 more | Severity: All severities | Tool: All tools | Activity: All activity

Detected	Status	Severity	Description	Identifier	Tool	Activity
2022-12-19	Needs Triage	Critical	CVE-2022-25236 in libexpat1-2.2.10-2 registry.gitlab.com/dassgroup...4dbdca0a9f5caace1479a1d0f6ef908d	CVE-2022-25236	Container Scanning	

Figure 12: Application Build Security Scanning



Vulnerability Report

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 177 Operational vulnerabilities 0

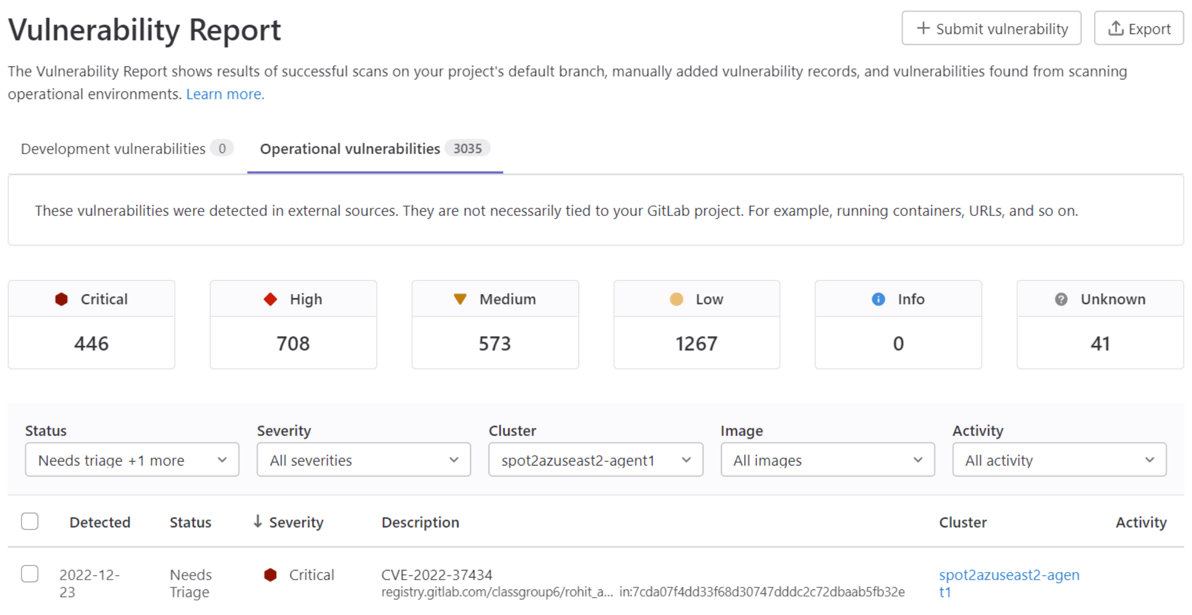
Last updated 1 week ago #727223780

Critical	High	Medium	Low	Info	Unknown
26	0	110	0	41	0

Status: Needs triage +1 more · Severity: All severities · Tool: All tools · Activity: All activity

Detected	Status	Severity	Description	Identifier	Tool	Activity
2022-12-19	Needs Triage	Critical	Roles and ClusterRoles with wildcard RBAC permissions provide excessive rights to the Kubernetes API and should be avoided. The principle of least privilege recommends to specify only the set of needed objects and actions packages/gitlab-agent/clust... um-install/resources.yaml:11	RBAC Wildcard In Rule	SAST	

Figure 13: Application Environment Security Scanning



Vulnerability Report

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. Learn more.

Development vulnerabilities 0 Operational vulnerabilities 3035

These vulnerabilities were detected in external sources. They are not necessarily tied to your GitLab project. For example, running containers, URLs, and so on.

Critical	High	Medium	Low	Info	Unknown
446	708	573	1267	0	41

Status: Needs triage +1 more · Severity: All severities · Cluster: spot2azuseast2-agent1 · Image: All images · Activity: All activity

Detected	Status	Severity	Description	Cluster	Activity
2022-12-23	Needs Triage	Critical	CVE-2022-37434 registry.gitlab.com/classgroup6/rohit_a... in:7cda07f4dd33f68d30747dddc2c72dbaab5fb32e	spot2azuseast2-agent1	

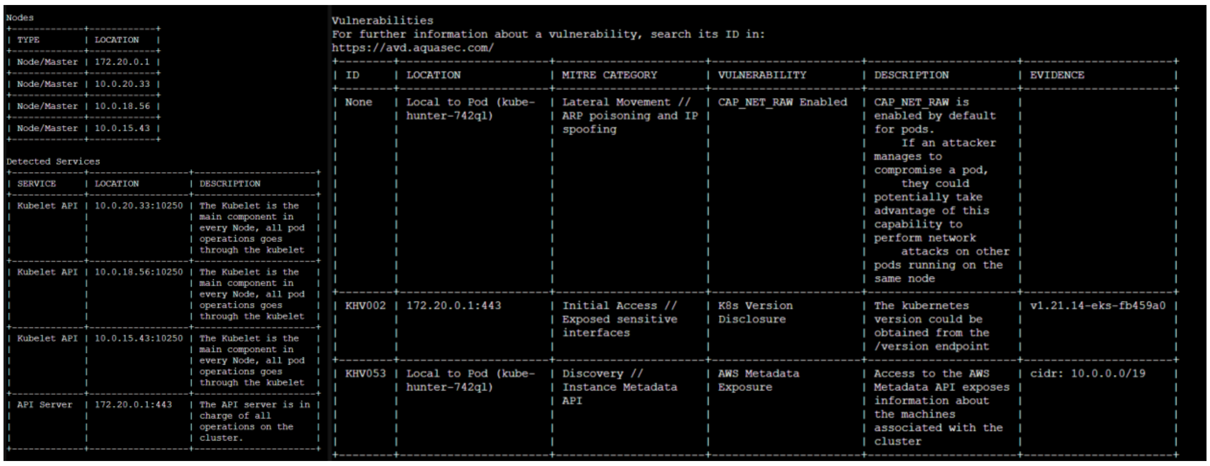
Figure 14: Container Scanning

6.6 Scanning using Kube-Hunter and Kube-Scape Tool.

The Kubernetes API and cluster were scanned using open-source tools to determine that the API is not exposed to the outside world when connecting Kubernetes and GitLab. Additionally, several other important tests were conducted to ensure that the Kubernetes cluster is protected from potential attack vectors and risk exposure.

6.6.1 Kube-Hunter

Kube-Hunter is a tool that can be used to scan a Kubernetes API and cluster for vulnerabilities. After running Kube-Hunter, no significant vulnerabilities were found in the API or cluster, indicating that the architecture is secure against API exposure. While there are some vulnerabilities present that should be addressed, but they are not considered to be major attack vectors. One of them is Kubernetes API version disclosure but that’s internal to the Kubernetes cluster not to the outside world as the API is not exposed publicly . Considering the experiential setup to align with our research methodology, the tool produces the following output regarding to Kubernetes API and cluster, along with the other warnings which are not in the scope of the research.



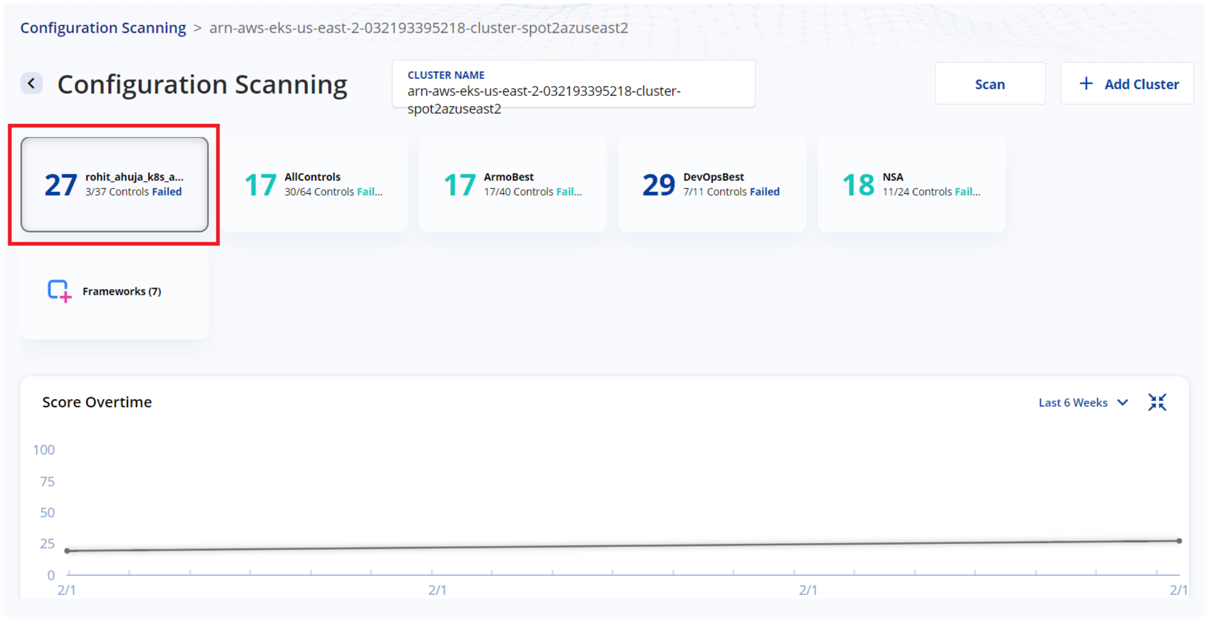
```
Nodes
+-------------+------------+
| TYPE        | LOCATION   |
+-------------+------------+
| Node/Master | 172.20.0.1 |
+-------------+------------+
| Node/Master | 10.0.20.33 |
+-------------+------------+
| Node/Master | 10.0.18.56 |
+-------------+------------+
| Node/Master | 10.0.15.43 |
+-------------+------------+

Detected Services
+-------------+------------------+----------------------+
| SERVICE     | LOCATION         | DESCRIPTION          |
+-------------+------------------+----------------------+
| Kubelet API | 10.0.20.33:10250 | The Kubelet is the   |
|             |                  | main component in    |
|             |                  | every Node, all pod  |
|             |                  | operations goes      |
|             |                  | through the kubelet  |
+-------------+------------------+----------------------+
| Kubelet API | 10.0.18.56:10250 | The Kubelet is the   |
|             |                  | main component in    |
|             |                  | every Node, all pod  |
|             |                  | operations goes      |
|             |                  | through the kubelet  |
+-------------+------------------+----------------------+
| Kubelet API | 10.0.15.43:10250 | The Kubelet is the   |
|             |                  | main component in    |
|             |                  | every Node, all pod  |
|             |                  | operations goes      |
|             |                  | through the kubelet  |
+-------------+------------------+----------------------+
| API Server  | 172.20.0.1:443   | The API server is in |
|             |                  | charge of all        |
|             |                  | operations on the    |
|             |                  | cluster.             |
+-------------+------------------+----------------------+
```

```
Vulnerabilities
For further information about a vulnerability, search its ID in:
https://avd.aquasec.com/
+--------+----------------------+----------------------+----------------------+----------------------+----------------------+
| ID     | LOCATION             | MITRE CATEGORY       | VULNERABILITY        | DESCRIPTION          | EVIDENCE             |
+--------+----------------------+----------------------+----------------------+----------------------+----------------------+
| None   | Local to Pod (kube-  | Lateral Movement //  | CAP_NET_RAW Enabled  | CAP_NET_RAW is       |                      |
|        | hunter-742ql)        | ARP poisoning and IP |                      | enabled by default   |                      |
|        |                      | spoofing             |                      | for pods.            |                      |
|        |                      |                      |                      |     If an attacker   |                      |
|        |                      |                      |                      | manages to           |                      |
|        |                      |                      |                      | compromise a pod,    |                      |
|        |                      |                      |                      |     they could       |                      |
|        |                      |                      |                      | potentially take     |                      |
|        |                      |                      |                      | advantage of this    |                      |
|        |                      |                      |                      | capability to        |                      |
|        |                      |                      |                      | perform network      |                      |
|        |                      |                      |                      |     attacks on other |                      |
|        |                      |                      |                      | pods running on the  |                      |
|        |                      |                      |                      | same node            |                      |
+--------+----------------------+----------------------+----------------------+----------------------+----------------------+
| KHV002 | 172.20.0.1:443       | Initial Access //    | K8s Version          | The kubernetes       | v1.21.14-eks-fb459a0 |
|        |                      | Exposed sensitive    | Disclosure           | version could be     |                      |
|        |                      | interfaces           |                      | obtained from the    |                      |
|        |                      |                      |                      | /version endpoint    |                      |
+--------+----------------------+----------------------+----------------------+----------------------+----------------------+
| KHV053 | Local to Pod (kube-  | Discovery //         | AWS Metadata         | Access to the AWS    | cidr: 10.0.0.0/19    |
|        | hunter-742ql)        | Instance Metadata    | Exposure             | Metadata API exposes |                      |
|        |                      | API                  |                      | information about    |                      |
|        |                      |                      |                      | the machines         |                      |
|        |                      |                      |                      | associated with the  |                      |
|        |                      |                      |                      | cluster              |                      |
+--------+----------------------+----------------------+----------------------+----------------------+----------------------+
```

Figure 15:Kube-Hunter Results

6.6.2 Kube-Scape

Kube-Scape offers security scanning as a feature to help users identify and fix potential vulnerabilities in their Kubernetes clusters. This can include checks for misconfigurations, insecure resource settings, exposed secrets, and API exposure. With Kube-Scape, users have the ability to customize their own framework for security scanning. This allows them to choose specific controls or areas to focus on during the scan. The K8s_api_scan framework was created within the Kube-Scape portal and includes important API scanning controls. This framework, when applied to an experiential setup, produces output related to the Kubernetes API and cluster, as well as any other warning messages that fall outside the scope of the research. Out of 27 controls, 3 failed in the K8s_api_scan framework.



Configuration Scanning > arn-aws-eks-us-east-2-032193395218-cluster-spot2azuseast2

Configuration Scanning

CLUSTER NAME: arn-aws-eks-us-east-2-032193395218-cluster-spot2azuseast2

27	rohit_ahuja_k8s_a...	3/37 Controls Failed
17	AllControls	30/64 Controls Fail...
17	ArmoBest	17/40 Controls Fail...
29	DevOpsBest	7/11 Controls Failed
18	NSA	11/24 Controls Fail...

Frameworks (7)

Figure 16: Kube-Scape Portal Results

6.7 Discussion

After conducting various case studies, it was determined that the most effective security measures involve preventing API exposure to the public internet, implementing a pull-based approach, and implementing additional security controls such as isolating the application build and environment and adding security scans. These measures were found to be particularly beneficial in reducing risk exposure for the application.

Case Study 1 demonstrates that the API is not publicly exposed when using the method, which helps to keep the deployed application secure. If a hacker were able to access the Kubernetes API, they could potentially compromise the security of the entire system by modifying or deleting resources, escalating their privileges, or executing malicious code within containers. This could lead to various security incidents, such as data breaches, unauthorized access to sensitive systems, or disruption of critical services. On the other hand, if a hacker is unable to access the Kubernetes API, they will be unable to take any action on it. Case Studies 2 and 3, shows how the Gitlab Agent uses RBAC rules and AAA authorization to grant only the necessary permissions. Additionally, it employs a pull-based approach instead of a weaker push-based approach. Case Study 4 shows that the application build and application environments are isolated by being deployed in different pods and nodes, which adds a layer of security to the implementation. Case Study 5 demonstrates the use of the Gitlab Agent to perform security scans and identify vulnerabilities that must be addressed before the application is moved to production in order to reduce the risk of exposure.
Talking about the improvement, network segmentation can be added to Kubernetes cluster in which the developer can specify rules for which pods are allowed to communicate with each other and with other resources in the cluster. This can help prevent unauthorized access and reduce the attack surface of the cluster.

7 Conclusion and Future Work

The research described aimed to address the issue of securely deploying microservices in a cloud-native environment using GitLab client-based authentication and Kubernetes. To achieve this, the GitLab agent was deployed in a Kubernetes pod and other necessary implementations were deployed. The evaluation of the proposed solution demonstrated that the API was not exposed to the public internet and the use of a pull-based approach, environment isolation, and security scanning added an extra layer of security.

However, the limitation of this work is that it is only designed for use with the AWS cloud provider, as there are many other cloud providers available. As a result, future work could include deploying the GitLab agent to different Kubernetes cloud service providers to assess how efficiently they integrate and work together. Additionally, creating a helm package of the GitLab Agent and limited RBAC rules could make it easier to deploy in the future.

References

Agrawal, M. and Abhijeet, K. (2020) ‘Security Audit of Kubernetes based Container Deployments: A Comprehensive Review’, 07(06), p. 6.

Autio, T. (2021) *Securing a Kubernetes Cluster on Google Cloud Platform*. Available at: http://www.theseus.fi/handle/10024/501918 (Accessed: 24 November 2022).

Bose, D.B., Rahman, A. and Shamim, S.I. (2021) ‘“Under-reported” Security Defects in Kubernetes Manifests’, in *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS). 2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, pp. 9–12. Available at: https://doi.org/10.1109/EnCyCriS52570.2021.00009.

Dakic, V., Redzepagic, J. and Basic, M. (2022) ‘CI/CD Toolset Security’, in B. Katalinic (ed.) *DAAAM Proceedings*. 1st edn. DAAAM International Vienna, pp. 0161–0164. Available at: https://doi.org/10.2507/33rd.daaam.proceedings.022.

Felix, C., Garg, H. and Dikaleh, S. (2019) ‘Kubernetes security and access management: a workshop exploring security & access features in Kubernetes’, in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*. USA: IBM Corp. (CASCON ’19), pp. 395–396.

Giangiulio, F. and Malmberg, S. (2022) *Testing the Security of a Kubernetes Cluster in a Production Environment*. Available at: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-319492 (Accessed: 23 November 2022).

Gupta, S. *et al.* (2022) ‘Prevalence of GitOps, DevOps in Fast CI/CD Cycles’, in *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON). 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON),* pp. 589–596. Available at: https://doi.org/10.1109/COM-IT-CON54601.2022.9850786.

Montalbano, E. (2022) *380K Kubernetes API Servers Exposed to Public Internet*. Available at: https://threatpost.com/380k-kubernetes-api-servers-exposed-to-public-internet/179679/ (Accessed: 20 December 2022).

Poniszewska-Marańda, A. and Czechowska, E. (2021) ‘Kubernetes Cluster for Automating Software Production Environment’, *Sensors,* 21(5), p. 1910. Available at: https://doi.org/10.3390/s21051910.

Ramadoni, Utami, E. and Fatta, H.A. (2021) ‘Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD’, in *2021 4th International Conference on Information and Communications Technology (ICOIACT). 2021 4th International Conference on Information and Communications Technology (ICOIACT),* pp. 186–191. Available at: https://doi.org/10.1109/ICOIACT53268.2021.9563984.

Shodan Facet Analysis (2022). Available at: https://www.shodan.io/search/facet?query=product%3A%22Kubernetes%22&facet=cloud.service (Accessed: 4 January 2023).

Sinde, S.P. *et al.* (2022) 'Continuous Integration and Deployment Automation in AWS Cloud Infrastructure', *International Journal for Research in Applied Science and Engineering Technology*, 10(6), pp. 1305–1309. Available at: https://doi.org/10.22214/ijraset.2022.44106.

STAMFORD (2022) *Gartner Forecasts Global iPaas End-User Spending to Grow 18.5% in 2022*, *Gartner*. Available at: https://www.gartner.com/en/newsroom/press-releases/2022-08-04-cloud-platform-hc-press-release (Accessed: 25 November 2022).

Sultan, S., Ahmad, I. and Dimitriou, T. (2019) 'Container Security: Issues, Challenges, and the Road Ahead', *IEEE Access*, 7, pp. 52976–52996. Available at: https://doi.org/10.1109/ACCESS.2019.2911732.

Throner, S. *et al.* (2021) 'An Advanced DevOps Environment for Microservice-based Applications', in *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 134–143. Available at: https://doi.org/10.1109/SOSE52839.2021.00020.

Дарвеш, Г., Хаммуд, Д. and Воробьева, А.А. (2022) 'SECURITY IN KUBERNETES: BEST PRACTICES AND SECURITY ANALYSIS', *Вестник УрФО. Безопасность в информационной сфере*, (2(44)), pp. 63–69.