National College of Ireland

Container Image Security with Trivy and Istio Inter-Service Secure Communication in Kubernetes

MSc Research Project
Cloud Computing

Prawal Saxena
Student ID: 21151334

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Prawal Saxena
Student ID:	21151334
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	01/02/2023
Project Title:	Container Image Security with Trivy and Istio Inter-Service Secure Communication in Kubernetes
Word Count:	7900
Page Count:	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.

Signature:	Prawal Saxena
Date:	27th January 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 the 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):	

Container Image Security with Trivy and Istio Inter-Service Secure Communication in Kubernetes

Prawal Saxena
21151334

Abstract

Containers eliminate the need for manual deployment by using images that have already been constructed. However, the images that are associated with a particular service have a risk of being corrupted, which may result in damage to the whole system. This case study describes the security measures where Trivy tool will scan a Docker Image, and it will connect with an AWS CI/CD pipeline. This ensures that the image will be scanned each time the pipeline is run, and it will be uploaded to the ECR repository. This will prevent any harm from coming to the system. The methods consist of first scanning the image to check for any vulnerabilities and then saving the results to the repository for container images. Because Kubernetes has a complicated design and one of its primary concerns is its level of security, the platform includes a mesh of services, but interestingly its internal services are not encrypted. It is composed of a large number of distinct clusters that are separated from one another and provide a variety of services, however, each and every one of them is linked to the same network. If a malicious request is submitted to the cluster and bypasses the front-end security, it has a good chance of causing harm to one of the services, and if even one service or component of the cluster is compromised, the entire cluster is at risk. It is critical to ensure that the cluster is protected, and as part of this investigation, I am using an Istio tool implementation for service mesh, which will configure rules and proxies for incoming requests. The request will be validated when it passes through these predetermined rules, and then it will be sent to the appropriate request pod.

1 Introduction

Containers are little virtual machines (VM) that are both highly lightweight and more efficient than traditional VMs since they are managed by a single operating system. Container technology gives the appearance of being secure since it contains all the components, yet these components have certain vulnerabilities, and the system does not guarantee complete safety. These tools are part of the CI/CD pipeline flow, which entails the containerization of application deployment, the creation of a docker image containing all of the dependencies and libraries needed to run the application website or any other service, and the subsequent storage of that image in a repository such as DockerHub or ECR or any other cloud repository that supports the pushing and pulling of images.
Integration of tools is simple, but configuring tools such that they work with the service being deployed is difficult. Although the deployment procedure seems to be quite straightforward at first glance, upon closer inspection, it is very intricate. Security is a

crucial component that many developers and DevOps team members overlook due to its complexity, and it will become a huge problem once it is released to production. This is owing to the fact that if one service is hacked, the other services are also at risk of getting corrupted.

1.1 Motivation

The application may easily get damaged if the vulnerability is present in the Container image that contains the application code. The application code is included in the dependency and library packages that are included in the image. Before the image is deployed, it is essential to first ensure its safety and conduct a scan. The opensource program may scan the images, and the pipeline will fail if it includes malware or a vulnerability. It demonstrates the problem with the image and how to address it using the Trivy tool and console output. Some thresholds can be set to examine common vulnerability exposure and determine what action is required to remedy it. As a consequence, environmental security will become normal practice.
Following the completion of the operation to secure the image, the next big and essential step will be to secure Kubernetes. Although Kubernetes comes pre-configured with a firewall and proxy rules, it is not sufficiently protected against hacker assaults. In addition, even if the services are separated from one another, their internal communications are not encrypted by default. This is despite the fact that all services share a single connection to the same network. Once a web request has been validated, it will go inside the cluster with no rules or proxies. Because of this, if a hacker compromises just one service, there is a good chance that they will be able to compromise the entire cluster. When a web request is received by a cluster, some rules are first set outside the cluster, and the request must then pass these rules in order to enter the cluster. Due to the cluster's complexity as a mesh of services, it is difficult to coordinate any security in the cluster. As a result, a paradigm known as service mesh with Istio tool implementation has been established with the intention of addressing the issue of security. It will protect the inter-service connection via rules, proxies, and mutual TLS (mTLS). The incoming request will be verified in terms of the established rules, proxies, and Oauth, and then it will be sent to the particular pod after validation has been completed.

1.2 Objective

Research Question: How can a Container Image be secured and obtain a secure inter-service communication in Kubernetes?
This study will concentrate on providing the most effective answer to the topic that was just posed by taking into account the future work and constraints that were outlined in previously published research papers and articles. This article will provide the typical approach to scan the container image in CI/CD pipelines with Trivy, which will subsequently be followed by deployment in Kubernetes clusters. The primary focus of the study will be to demonstrate, in a real-world setting, how the service mesh Istio tool can be used to ensure the safety of inter-service communication and the request flow to the application pod through the Istio gateway.

2 Related Work

In this part, a critical evaluation of the primary research article and all other publications that contributed to this study will be found. It will provide a critical analysis of the paper, outlining the work that has been done and the directions that need to be taken or the restrictions that have been identified. Containers in Cloud Computing are currently essential for deploying online applications since they are beneficial for constructing blocks and contribute to operational efficiency, productivity, and environment consistency while being lighter than virtual machines. The container is the foundation of the platform as a service (PaaS) and is utilized in the packaging and deployment of infrastructure. DevOps Continuous Integration and Deployment pipelines may use containers to facilitate continuous deployment of cloud-native platform architectures. Container Orchestration is vital to the reduction of complexity in development and operations via the automation of its processes and is a natural match for the DevOps way of working. It streamlines operations, offers elastic scalability, and restarts and scales container clusters automatically. Container-based software application cluster management, including start/stop operations, as well as continuous deployment. Orchestration aids the user in defining the availability, scalability, and network coordination of containers in the cloud when a multi-container application must be deployed there.

2.1 Containers and Container Image Security

Containers do the virtualization at the operating system level, they use the OS resources and they are also isolated from each other. Virtual machines that use the hardware of the physical machine have their own operating system and are isolated from other virtual machines. Many VMs can run on the same system or physical machines but they are not connected to each other. The primary distinction between containers and virtual machines is that virtual machines are able to virtualize a full computer all the way down to the hardware layers, but containers are only able to virtualize software layers that are above the level of the operating system. Containers have gained a lot of popularity in the field of software development because they relieve developers from worrying about libraries and dependencies, which can be quite time-consuming to configure. Images are what is responsible for creating containers, and inside the image are all of the libraries and dependencies that need to be present for an application's successful executions. Containers may be utilized in systems that make use of microservices. Containers are relatively lightweight since they do not need a copy of each image. The need for ephemeral state containers in a microservice architecture is driven home by the fact that any data that has to be stored for an extended period of time must be sent to another data store or service. (Yasrab (2018))
Docker is a Platform as a Service (PaaS) that makes use of OS-level virtualization to either execute applications or distribute software packages in the form of containers. Docker is very popular, which is why the developer community has created many repositories of Docker images, such as DockerHub and AWS ECR. Docker is now the most popular containerization solution, yet there will always be concerns over security. There are a variety of potential security flaws that might manifest themselves, including kernel exploits, DOS attach, container-poisoned images, compromised secrets, container breakouts, and so on. One of the most typical assaults, known as a denial-of-service attack (DOS), is used when there are problems with the system's external security. If the container image is contam-

inated with a virus or trojan horse program, then the whole of the application will be put in jeopardy. There are a few possible circumstances in which the libraries and dependencies are listed in the Docker file that I use. However, if an out-of-date version is installed, a hacker will have an easier time breaking into the image and container. Docker never conducts an authentication of the image, making it possible for an adversary to send or corrupt the image. In the article, the aforementioned problem's preventative measures and safety protocols were dissected and analyzed in detail. (Henriksson and Falk (2017)) There are many different approaches that I may take to protect ourselves from the vulnerabilities, such as Common Vulnerabilities and Exposures, or CVE, which is used to determine where vulnerabilities exist. Docker images will be analyzed by the CVE system, which will then provide an alert if anything malicious is discovered. A scan will be performed on a regular basis, and any issues or vulnerabilities will be rectified before deployment. Vulnerabilities may be handled by adopting the appropriate precautions. There are a number of tools available on the market that can perform a scan on the Docker image before publishing it to Docker Hub or any other docker repository. The image may be scanned using numerous tools on a local PC before being submitted for testing. This is okay since several images will not be deployed, and production will not be disturbed. However, if I need to scan in a project, then there will be multiple environments and stages, and so I need to find a way that all of the steps will run in a flow sequentially, which is what is meant by continuous integration and deployment. (Chamoli et al. (2021))

2.2 Container Image scanning with Trivy

Docker has a crew that is completely dedicated to the process of vetting and releasing content in the official repository. Beginning from one of the official repositories for extra use cases commons, which provide extensive documentation and support for best practices, is a good choice. These repositories may be found on the Internet. It's risky to put your faith in container photos seen on the internet. Adding image scanner tool integration to the continuous integration and continuous delivery pipeline is a recommended best practice. Because Continuous Integration allows us to generate and submit images to Docker Hub or any other container repository, I can take use of this feature. (Abhishek and Rao (2021))
When there is a change to the code, the CI/CD pipeline is performed, and the steps that scan the image are mentioned in the build step of the pipeline. If the image is successful in all of the tests, the pipeline will either upload it or save it to a repository. The image may be scanned with the assistance of third-party tools, and the tools themselves can be merged using continuous integration. Some well-known examples of such tools include Clair, Anchor, and Trivy. In addition to supporting the Jenkins pipeline, they also support AWS and Azure DevOps pipelines. After the integration with the Cloud pipeline is complete when the pipeline is run, the tool will scan the image using the commands that I have defined for each step of the pipeline, and it will also provide the report on its website. While some programs allow users to share the user interface view of the report, others provide the ability to send the report after it has been prepared. In addition, I am able to inspect the output of the report as well as the output of the instructions in the pipeline. (Valance (2019))

2.3 Microservices and Kubernetes

The software known as microservices deconstructs and modularizes larger software applications and systems. As a natural progression from service-oriented architecture, this idea emerged at the same time around 10 years ago. As a natural progression from Service-Oriented Architecture (SOA), this idea emerged at the same time around 10 years ago. DevOps can provide quick delivery cycles via the usage of refactored service-oriented architecture by merging the previously separated business processes of development and operation. Microservices are small applications that may be managed separately from one another in terms of deployment, scaling, and testing. The deconstruction of a monolithic system into a granular system in which components communicate with one another through messages facilitates an organization's capacity to bring products to market more quickly and with more consistency (for example, via RPC-based APIs or RESTful web services). In addition to the fact that they are developed independently, microservices provide agile teams the opportunity to structure their work around them. (Larrucea et al. (2018))

Microservices and development operations (DevOps) working together will result in increased effectiveness and advantages from software engineering. On the other hand, microservices come with their very own unique difficulties and disadvantages. Some of the challenges that come along with deconstructing the monolith include microservices, continuous architecture monitoring, sophisticated testing, versioning, and deprecation, as well as state management. (Pautasso et al. (2017))

This piece of software is called a container orchestrator, and it is used to automate the deployment and deployment of service containers. This method has the potential to enable scalability for applications that are operating in containers and have requirements for load balancing, fault tolerance, and horizontal scaling. Direct support for Docker containers is included in Kubernetes's feature set. Kubernetes has a feature called Pods that assists with the administration of containers. A Pod is a logical envelope that encloses one or more containers that are firmly connected together. A Pod may contain anywhere from one to many containers. Flannel may be used to build up a software-defined network (SDN), which enables all pods to communicate with one another via the use of their own individual IP addresses. The architectural components of Kubernetes may be broken down into two distinct categories: components at the master and node levels. A Kubernetes cluster's master node is responsible for making major decisions on a global scale. Etcd, Kube-scheduler, Kube-controller-manager, and Kube-apiserver are the four individual components that make up the master. The Kube-apiserver is responsible for monitoring and coordinating all Cluster-Activities. An application program interface is provided to Kubernetes via the API server, which allows Kubernetes to use its features (API). (Beltre et al. (2019))

The term "controller" refers to the component on the master that checks the status of the cluster via the API server and updates it to the desired state. On a dispersed network, the allotment of time slots for pods falls under the purview of this particular part of the control plane. etcd is a key-value store that serves as the repository for all of the Kubernetes cluster configuration data. Workers are constructed using kube-proxy, kubelet, and pod components respectively. Kube-proxy is responsible for ensuring that the network rules of the nodes are always current. The Kubelet agent is responsible for checking to see whether any containers are currently active inside a certain pod. A pod is the smallest Kubernetes entity, and it only has one operational container. The software can

operate in any computer environment since the source code and all of its dependencies are contained inside a container. ETCD serves as both data storage and a backup for the key value distribution. The overlay network functions as a Kubernetes Pod network, connecting each pod to the rest of the cluster via SDN as well as a distinct IP address for each pod. Within the scope of this article is a comprehensive analysis of Kubernetes and all of its individual parts. (Miles (2020))

2.4 Security Practices in Kubernetes

Because Kubernetes is so complicated, it might be challenging to protect it from being compromised by malicious actors. Kubernetes is comprised of a large number of moving parts, each of which has to have its own Pod and network. The papers presented several procedures that should be carried out to ensure safety. Policies that are particular to the network Kubernetes pod communication may be protected against unwanted network traffic by establishing a policy that is specific to the network. All pods in Kubernetes have the ability to interact with one another by default. As a method of providing security for the network, practitioners advocate for the adoption of rules that restrict communication between pods, limit access to API servers, and reduce the amount of time spent exposed to the network. If there are no network limitations or firewalls in place, any IP address may launch an assault against the API server. Practitioners also propose restricting pods' access to databases by using appropriate firewalls, in addition to the usage of network policy plugins like Calico to prevent unwanted network connections. This may be accomplished by preventing unauthorized network connections. (Shamim et al. (2020))
Policies that are unique to pods allow for the application of a security context to pods and containers via the use of a pod policy. Pod rules are what ultimately decide how a workload is executed inside a Kubernetes cluster. It is possible for a container to operate as root and have write access to the root file system if a secure context for the pod is not provided. This would result in the Kubernetes cluster being vulnerable. Protecting the Kubernetes cluster components by putting in place a generic security policy is what is meant by the phrase "generic policies." Authentication is required for all network activity, therefore network plug-ins like kubelets, API servers, etcd should not have any of their TCP ports exposed. Every user that logs into the system needs to be restricted to the most basic degree of access. It is recommended that access to SSH be limited on cluster nodes. Users of Kubernetes are strongly encouraged to implement an audit policy for logging, which is something that has to be done on the API server level for each and every Kubernetes cluster, mentioned in the paper. (Kalubowila et al. (2021))
Vulnerabilities and malicious software may be hiding within containers. Vulnerabilities and harmful malware may also be housed inside containers. In the event that a Kubernetes cluster has vulnerabilities, the whole container orchestration system as well as the apps that are linked with it are put into jeopardy. If the continuous deployment components images and deployment settings are not checked, the Kubernetes cluster may become susceptible to attack by hostile users. It is advised that you use a reputable private registry, in addition to inspecting the code and photos for any vulnerabilities. The technique of separating namespaces in such a way that no resources from one namespace may be utilized by another namespace is referred to as namespace separation. In Kubernetes, a namespace is a name given to an individual virtual cluster that is contained inside a larger physical cluster. The default namespace will be utilized for any resources that do not have their own namespace; in its place, I will use the default namespace. This is a

really good example to follow. If there aren't any distinct namespaces for each of the multiple teams, then any malicious user may target the default namespace, which leaves the whole resource open to assault. Utilizing the -namespace option in Kubernetes allows the creation of unique namespaces inside the system. (Hussain et al. (2019))

2.5 Istio inter-service communication security in Kubernetes

The paper explains how they have used Istio in the company and perform some tests and on the basis of the result prepared an evaluation report. In the paper, it is shown that they have used certain rules and gateway for redirecting the traffic application traffic through istio. Istio uses service mesh for the configuration, service mesh is a very new technology, and some developers have speculated that it may be used in business logic. Istio's fundamental capabilities may be broken down into the following three categories: traffic management, security, and observability. The features of Istio that enable its expansion have been ignored since, on their own, these features do not offer any functions for inter-service communication. Hence, discussing them is outside the purview of this paper. Istio, on the other hand, has characteristics that make it possible to expand the mesh to VMs located outside of the Kubernetes cluster. The protocols gRPC-Web, HTTP, HTTP2, HTTPS, MongoDB, MySQL, Redis, TCP, TLS, and UDP are all able to have the rules and policies applied to them. Complexity, independence from programming languages, the possibility of further failures, performance, flexibility, and the primary aspects of Istio's characteristics were the primary concerns brought up (traffic control, observability and security). (Li et al. (2019))
It is expected that Istio would minimize the complexity of applications as well as the amount of work required of developers to guarantee reliable communication between services. Istio may eliminate the need for network function libraries, which would result in a reduction in the complexity of the code. However, it increases complexity in other areas. The developers working of the company are concerned about the extra expertise that is needed in order to set up the functionality of Istio. The worry is compounded by the fact that Istio is meant to be the most advanced service mesh. The functionalities of Istio may be centralized and handled in a manner that is distinct from the application code. One of the advantages that were noted was that developers were able to concentrate on the business logic rather than the problems that span across several services (inter-service communication). Istio's biggest drawback is the added latency that comes from using proxies. The amount of additional delay varied dramatically from trial to experiment. Because the tests were run using a variety of configurations, with a variety of service typologies, and on top of a variety of settings, it is difficult to compare the outcomes of the two sets of testing. When new features are added, the proxies may have to carry out more checks, which causes them to spend extra time before sending the request to its final destination. Istio has the potential to make features that are presently impossible and feasible. These functionalities are currently impossible because either the existing environment does not permit them or the implementation using the library method would be too difficult. This may make it possible for developers to add features across teams if the process of implementing Istio's features was not too complicated and if good examples were supplied. The capabilities may improve the entire architecture's observability as well as its resistance to failure. (Mara Jösch (2020))

3 Methodology

The research has two tasks, first is to scan the docker image if it's secure then push it to the ECR registry then deploy the image to the Kubernetes cluster by creating some pods. The next task it to install Istio service mesh and show the inter-service communication is secured and the request that is coming from the internet to the pod or cluster is going through the ingress gateway with rules and destination. Multiple AWS services and Kubernetes components have been used during the course of this study. Trivy is an open-source third-party scanning tool that will first check to see whether the image is safe or not. If it is, then it will be deployed into our environment if it is not secure, then it will not be uploaded to the ECR repository until it passes the validation. GitHub will be a source code repository and with the help of GitHub webhooks it is integrated with the Code pipeline. For automating the process AWS CI/CD pipeline has been used and deploying the code with the docker image in the Kubernetes cluster. AWS EKS has been a managed Kubernetes service with two nodes and two namespaces created for istio pods and web application pods.

3.1 AWS Service and components of Kubernetes

Code Build:- Once the Code Build project has been created and integrated with the Code pipeline. This will also run in some operating systems so while creating the project operating system needs to be selected I have selected Ubuntu and enabled the option for docker to automatically install in the backend so that installation is not required. The Buildspec file is the main element in the code build that has all the commands and steps which will run in the project. There are a few steps like pre-build, build, and post-build where it mentions installing the required tools, building the dockerfile and pushing it to the ECR repository. For deployment also code build will be needed as the post-build step has steps to login to the EKS cluster and run the deployment files. There are some IAM roles also needed that I have discussed further.

Code Pipeline:- In these two stages will be there first is the source that has been integrated with GitHub with the specific branch next will be the code build. The build project has been created that needs to be chosen and created the pipeline. Whenever there is a code change then the pipeline will trigger.

AWS ECR:- It is a docker repository that saves the docker images. One repository will be created and the details need to be put in the buildspec file of code build after successful login to the ECR repository image can be pushed and pulled.

AWS CloudWatch:- When creating the code build cloudwatch log stream name should be given as the code build logs will be shown in cloudwatch. If the option is not enabled then no one can see the logs of the code build. Additional metrics of the Kubernetes cluster need to be checked, such as health and compute metrics of Ec2.

AWS SNS:- When the code pipeline has an error, the simple notification service is responsible for delivering notifications to the various teams. In addition to that, it will attach the output file to the email, which will include any mistakes as well as any other

details that were extracted from the scanned image.

IAM:- This is very important in the Project as there are many services that have been used and by default they cannot communicate with each other so roles and policy will be required to make a connection between them. In buildspec for logging to ECR repository, an IAM role is created and attached to code build that has the policy to access the created repository and push and pull enable. Similarly, for EKS as well first enter the cluster and then run the deployment file so policy needs to add for the authorization and in buildspec AWS authentication steps also need to be written. For cloudwatch SNS as well policy needs to be attached to the code pipeline role to have access to send notifications and show logs.

AWS EKS:- In this research managed Kubernetes service EKS has been used. The advantage is that I don't have to install all the small configurations and dependencies. Kubernetes is now able to provide an even wider range of advantages in terms of the management, deployment, and scalability of containerized applications thanks to Amazon EKS. EKS is the most effective method for running containers on Amazon Web Services. It provides a large level of freedom, a diverse ecosystem, and many other advantages. Additionally, the advantage of integration with AWS networking and security services, such as application load balancer (ALBs) for load distribution, AWS Identity and Access Management (IAM) integration with role-based access control (RBAC), and AWS Virtual Private Cloud (VPC) support for pod networking. Two nodes cluster are used here and eksctl is to create the clutser and nodes because it is easy to configure and manage.

GitHub:- It is a source code repository where the code is being pushed and deployed in kubernetes. It contains buildspec.yml file that will be used for running code build, Dockerfile that will be used in docker images, deployment.yml file that will be used in kubernetes to create multiple pods. Some other files like gateway.yml which has code for the ingress gateway of istio and virtual service. Application folders like product page, details, ratings, and reviews. It is integrated in the code pipeline as the source and with the help of webhooks whenever there is a change in the code repository automatically the pipeline will trigger.

Docker:- Docker is an application containerization technology that speeds up the software development life cycle. Docker compartmentalizes programs into portable containers. These containers hold the program and all of the libraries, system tools, code, and runtime that it needs to work properly. Docker makes it possible to rapidly deploy and grow apps in any environment while ensuring that your code will continue to execute. Containers work to virtualize a server's operating system in a manner similar to that of a virtual machine, which frees administrators from the need to directly control the underlying hardware of the server. Docker is a program that comes pre-installed on every server and provides simple instructions to create, run, and stop container processes. In the project building the docker images that are mentioned in Dockerfile then scan with Trivy and after a successful scan deploying it to the EKS cluster.

Trivy:- It is an open-source tool that scans the container images and finds any vulnerabilities present and shows output in json format. Trivy is compatible with a wide variety

of output formats, including table, json, sarif, cosign-vuln, github, spdx, and cyclonedx, among others. It is dependable, quick, and really simple to use, and it functions anywhere you want it to. Trivy has the capability to scan the GitHub repository, directory, filesystem, container image, Kubernetes cluster, or its resource. It also shows common vulnerabilities exposure (CVE), misconfiguration of IaC, and any sensitive information exposed or secrets exposed. It was developed specifically for use in CI. Before committing changes to a container registry or releasing an application, it is easy to do a scan on your local container image and any associated artifacts. Trivy integrated with AWS has the capability to scan the entire region of AWS and show the vulnebrities, it can also scan the S3 objects. In this project, I am integrating trivy in code build, installing trivy in the pre-build step then scanning the images in the post-build after validation moving it to the next step otherwise it will fail.

Istio:- Istio is a free and open-source service mesh that makes it possible to connect, monitor, and secure individual services at the microservice level. These microservices, whether they are hosted locally or in the cloud, can be utilized by an orchestration platform such as Kubernetes regardless of where they are hosted. Service mesh is an infrastructure layer is responsible for managing the communication between the many services that make up the network. It is a layer of the architecture of the network that is dedicated to handling communication from one service to another service. Common uses of this technology include those that run in the cloud, those that utilize containers, and those that provide microservices. Some entities of istio are Envoy sidecar proxies, the information on the nature of service requests is made available through the Istio data plane. Also, it automatically takes care of things like load balancing, service discovery, and failure handling. Next is mTLS is used in situations requiring authentication or authorization between proxies. Requests for services, audits, and authorizations are all meticulously recorded and maintained by it. Pilot utilizes authentication rules and name requirements for distribution in order to perform proxy load balancing and traffic management. Citadel is applied in the process of verifying identities and controlling access permissions between envoy proxies. It offers a robust and policy-driven security layer that is responsible for handling keys and certificates throughout the whole mesh.

4 Design Specification

The architecture diagram shows the flow of the project. As this project has two security aspects, container image scanning and inter-service communication of Kubernetes by Istio which is clearly seen in the diagram. First, the code is there in GitHub and it is integrated with the code pipeline, code build has buildspec file where steps for installation of trivy is written, ECR is also integrated with code build and for pushing the docker image to ECR is also written in buildpsec. The next step is to deploy the scanned image to the EKS cluster that steps in present in buildpsec file. The second security aspect is by Istio so after installing Istio I need to enable the istio injection to the specified namespace then mTLS communication will be there inside the Kubernetes cluster and whatever request is coming inside the cluster it will go through istio gateway. proxy and rules are also set on top of pods to make the connection secure.

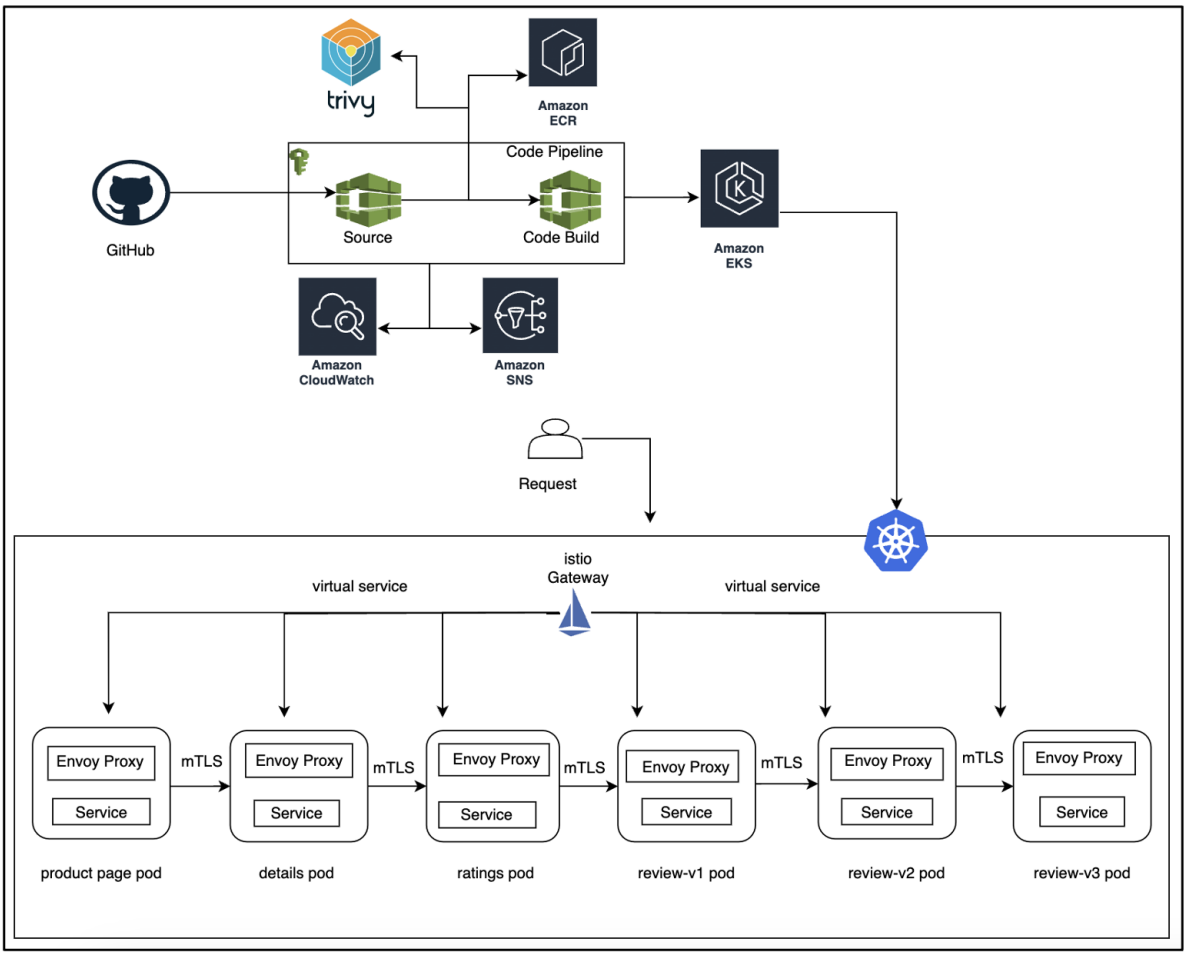


Figure 1: Architecture Diagram

5 Implementation

To implement this research I have created an AWS account and used AWS services that is mentioned in the above section. Starting with Container security Trivy scanner tool is included in the build stage of the CI/CD pipeline. This way, everytime the pipeline is executed, the image scans will also be executed. If the tool does not discover any vulnerabilities, the image will be uploaded to the container repository. Docker, which is the most popular and well-known container platform, will be used for all of the commands that need to be written since this is a requirement for uploading the image to the repository and requires the writing of a command. Docker will be in charge of creating the image and uploading it to the repository, while the scanning command will be tailored to the tool in use. The Continuous Integration and Continuous Delivery (CI/CD) pipeline technique has been chosen for this study since it is a common practice in DevOps to automate a process and to carry out the activity step by step without any interference from a human being. The application code I am using is from the Istio GitHub public repository and the link I have updated the configuration manual.
Here are the step mentioning the implementation:-

- First step is to create the Code Pipeline in that as a source mentioned the GitHub

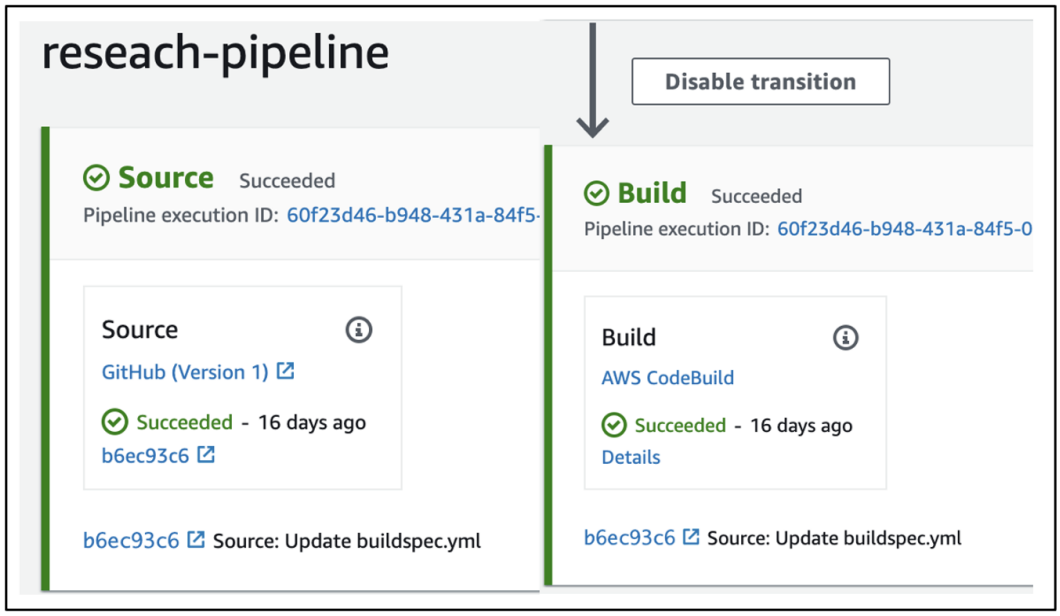
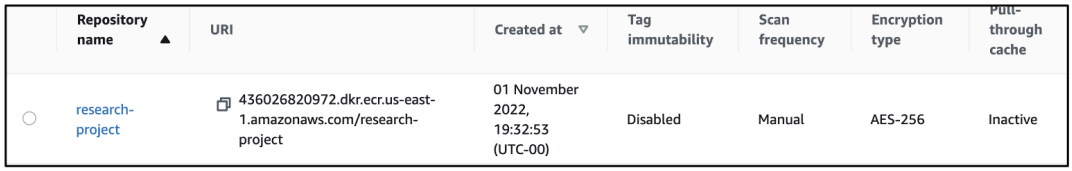


Figure 2: Code-Pipeline

repository with the specific branch then the next stage will be to build. For that Code, Build Project has been created which has IAM role to make a connection with ECR and EKS. Code Build requires the operating system to run at the backend where all the commands will run so ubuntu's latest docker image has been chosen and enables the option for installing already at the backend of the code build machine. Logs will be saved in the cloudwatch and logs can be fetched in the build project run status.

- Second step to create private ECR Repository.



Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type	Pull-through cache
research-project	436026820972.dkr.ecr.us-east-1.amazonaws.com/research-project	01 November 2022, 19:32:53 (UTC-00)	Disabled	Manual	AES-256	Inactive

Figure 3: ECR Private Repository

- Next EKS cluster to be created. Eksctl command is used to create the cluster. For this research two nodes has been created. After creating the cluster. kubectl get nodes will show the status of the nodes.
- After creating the nodes create two namespaces istio-system and research. A mechanism for isolating groups of resources within a single cluster is made available through the use of namespaces. It is only necessary for the names of resources to be unique within a given namespace; this requirement does not apply across namespaces. Only namespace-based scoping can be applied to namespaced objects

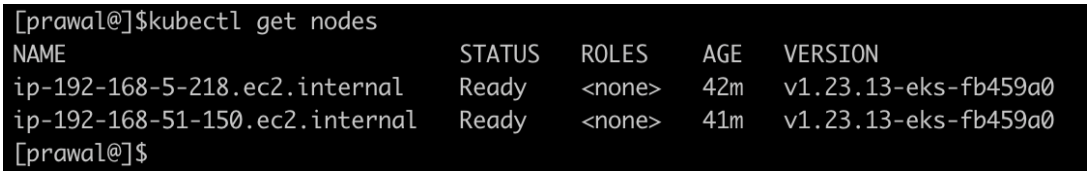


Figure 4: Kubernetes Cluster Nodes

(like Deployments and Services, for example), while cluster-wide objects are excluded from its remit.

- Now, Istio tool needs to be installed, for that istioctl command needs to be run that has been added in the configuration manual. After installing istio some istio addons needs to be installed as well, I can spin up the addon by running the kubectl command.
- I can check Pods and Services created in istio-system namespace using kubectl command.
- After installing istio next step is to label the research namespace with istio enabled true, this is because when the request is going to the cluster it will go through the istio gateway.

- In the buildspec file need to add stages like pre-build, build, post-build. In the stages commands and steps to install various tools are written like Trivy installation, kubectl and update the machine with apt-get then giving the aws ecr command to login to the repository that is created. ECR repository is a private repository as images should not expose publicly so full command needs to be given to login to ECR including username and password. Docker will be preinstalled when the build project is created i have selected docker prebuild install option. In build stage shows he steps of building the Docker image so Dockerfile path needs to be given that is present in the folder. In this project four Dockerfile are present in different folder to build the images for four different container pods docker build commands need to be given. With creating the image a tag is given with all the Dockerfile build command, which will be helpful for the identification of the specific Dockerfile image so that the tag can be easily given to the specific image.

```
docker build -t research-product -f productpage/Dockerfile .
```

- Post build stage shows the scanning of docker images with trivy that is built in the build stage, output can be seen in the code build project build history segment, it will show the docker image scanning result that any critical or severe. Output will be in the Json format showing the severities and CVEs of the docker image. The –severity tag can be used to show particular severity like HIGH, CRITICAL, LOW. In our scenario to know all the severity and CVEs –severity tag will not be given. Output format can be changed to tabular or sarif, cosign-vuln, github as trivy supports many formats but Json is the clearer and most common used format so json format is used in the research. After scanning the image output can be seen

on the code build output page in detail. For just checking the severity of the image trivy command is running in the codebuild buildspec file and see if the image is with less severity or not. Once the output result shows the images are secured then it can be pushed to ECR repository.

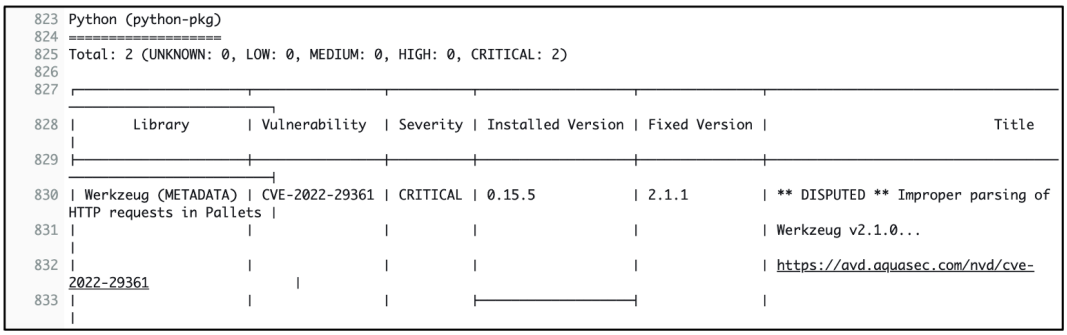


Figure 5: Trivy scan output

- Now, the next step is to login to EKS cluster and deploy the docker image. It can be done using Kubectl apply command, -n represent namespace kubectl, it means deploying all the pods and services in the research namespace.
- After successful login to will validate it through kubectl config view –minify and run the deployment.yml and gateway.yml file with kubectl command.

- After the pipeline successfully completed Figure [6] shows the pods health and if they are correctly deployed or not:-

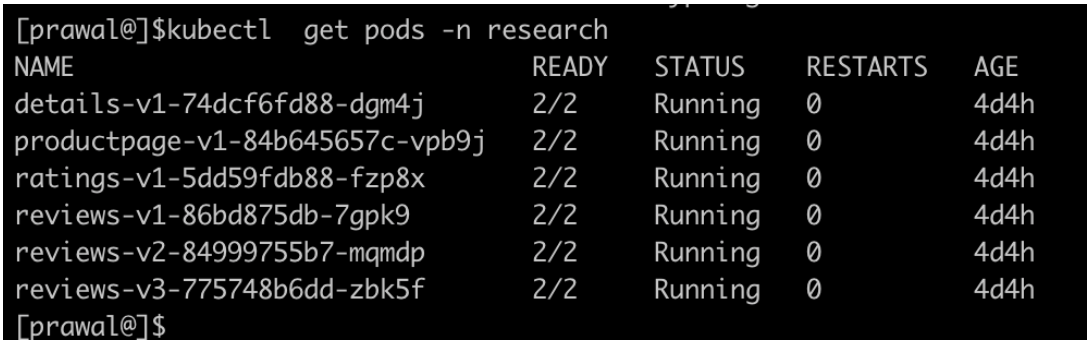


Figure 6: Pods in research namespace

- Virtual service and gateway are also created that contains destination rules that will define how the request is flowing in the cluster. The main application page is the product page and in the destination rules and gateway will redirect to product page only. Secure connection to service mesh will be establish by ingress gateway. After the service mesh ingress gateway terminated the inbound request, the mesh sidecar proxy crafted a new secure request and transmitted it to the destination service. Istio ingress communicates with the Istio control panel, routing requests to healthy back-end pods. As seen previously in this article, the Istio proxy will acquire the request from the backend pod rather than the real operating service.

Through the use of the Istio control panel, network and security policies may be implemented on a pod proxy Figure [14].

- By describing the pods it can be seen that one Istio proxy image will also be there on top of the application container. That means the request is flowing through istio proxy to the container. Istio has the capability to individually protect each pod within a service mesh by utilizing ingress and pod proxies. As long as the ingress gateway and pod proxies are active, any breach of the mesh network will be segregated and monitored by the control panel that monitors the network as well as the Istio proxies:-

Figure 7: Istio proxy on top of container

6 Evaluation

This section shows the result of the evaluation that is done on the security aspect of Trivy and Istio implementation.

- Firstly, Docker image needs to be build and scanned which contains some CVEs to show that if this kind of image is deployed the cluster risk of exposing the environment is high. This is because the code is not well written many parsing errors are there and dependencies and libraries are quite old and expired. Output can be seen in the code build logs page, trivy also shows what CVEs are present and the errors in the code. So, the image has been not been deploying, just building and scanning images. At the industry level as well whatever image needs to be deployed first needs to be scanned and check the vulnerabilities then with the other CI/CD pipeline push the image to ECR and then deploy it.

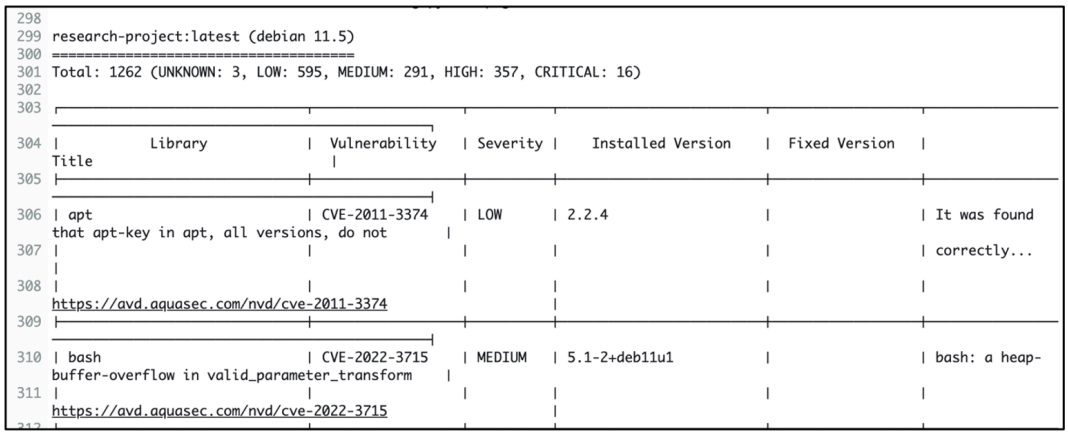


Figure 8: Docker Image with high vulnerabilities

- Now Docker image shows that has very less vulnerabilities. Less CVE means that the code is well written and the dependencies and library used are the latest and not expired.

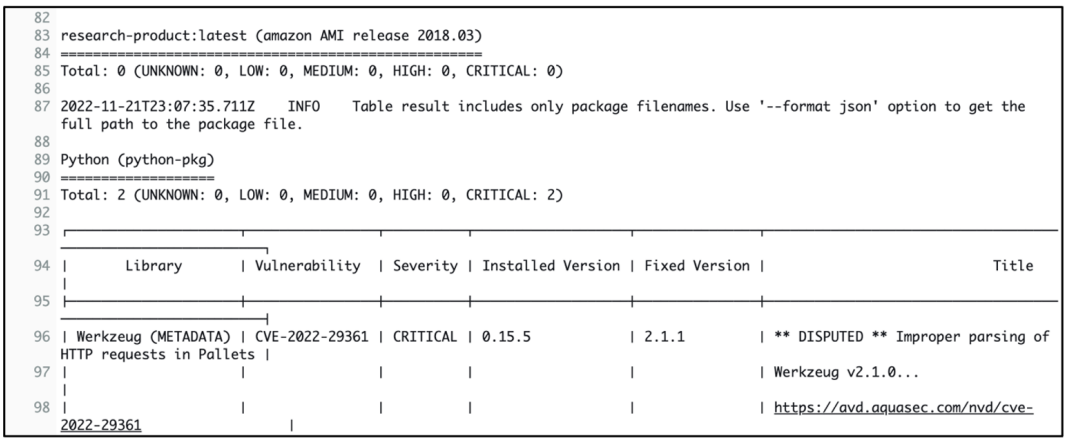


Figure 9: Docker image with less vulnerabilities

- Second step is to deploy the scanned docker image in the cluster and see how traffic is flowing in the cluster and with istio pods are secured or not. Istio tool has been installed before the deployment and injected the label into the research namespace. After describing the pod of the product page it can be seen that the istio-proxy will be at top of the service pod Figure [7].
- Some add-ons need to be installed that include Kiali dashboard and Grafana dashboard. Istioctl will enable the dashboards and it will open in the browser with different ports. kiali dashboard can be accessed on 20001 port.
- Both the namespace can be seen in Kiali Dashboard Figure [10].

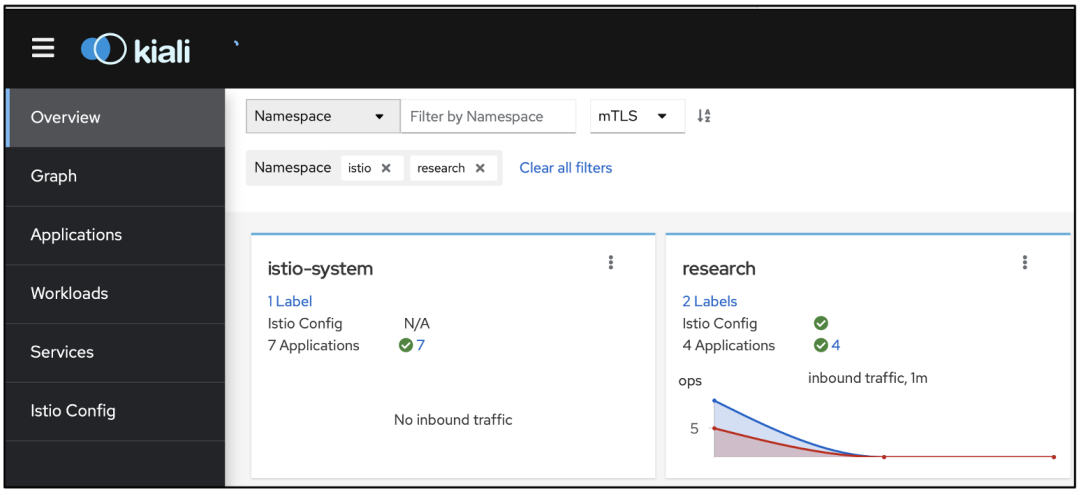


Figure 10: Kiali Overview

- In the kiali dashboard Application tab all the application pods can be seen that have been deployed in the cluster.

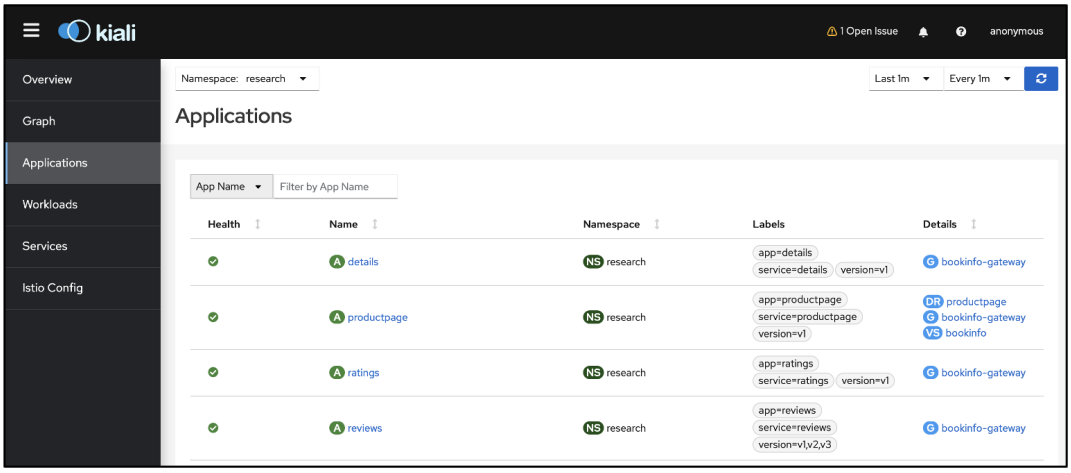


Figure 11: Kiali Application Pods

- For accessing the application in the browser use the Load Balancer DNS/productpage and with curl command keep sending traffic to the application.
- Figure [12] Kiali Graph dashboard shows the flow of requests in the cluster. It can see that the request is flowing through the istio gateway to the application pods. The request will be validated by istio gateway after validation it will go inside the cluster. This is done by the virtual service istio gateway that is included in the configuration manual.

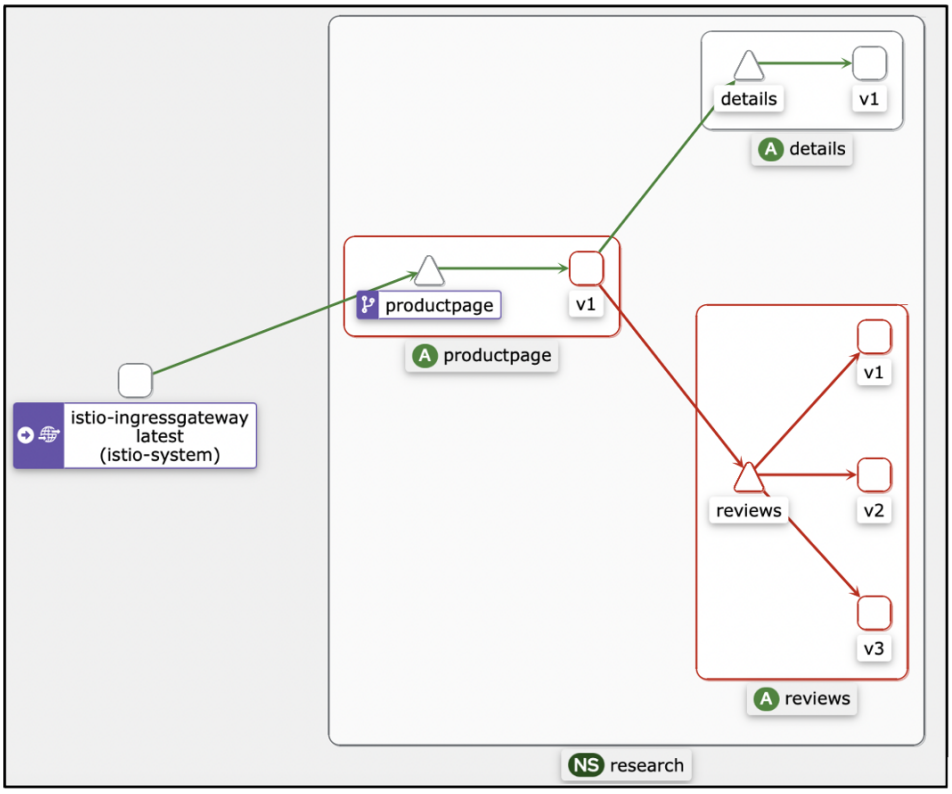


Figure 12: Request Flow in the cluster

- Figure [13] shows application pod details and request traffic flow. Figure [14] shows the request flow through the product page to the other details and reviews.

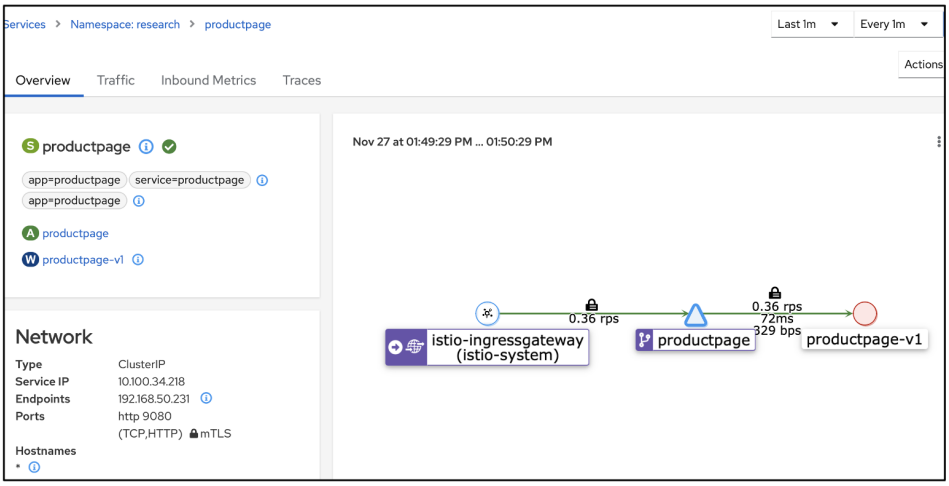


Figure 13: Details of product page

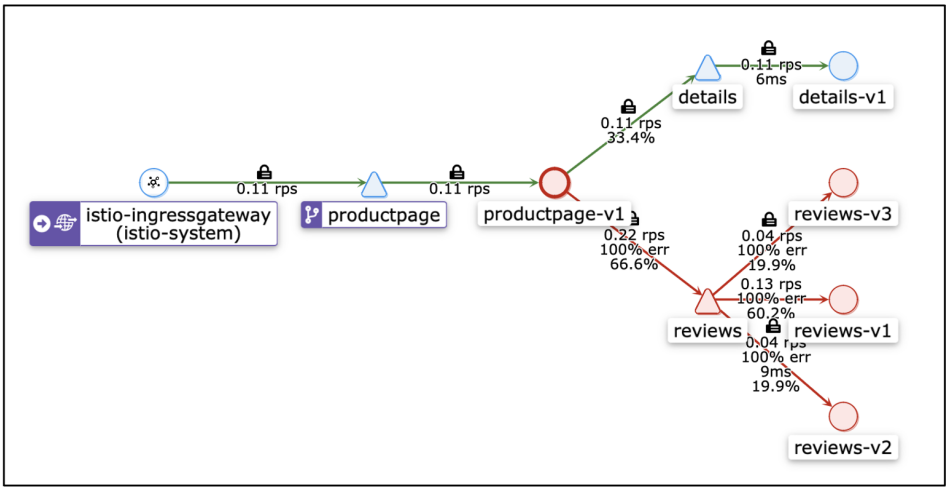


Figure 14: Request Traffic flow

- Figure [15] shows the CPU and Memory usage in the Grafana dashboard.

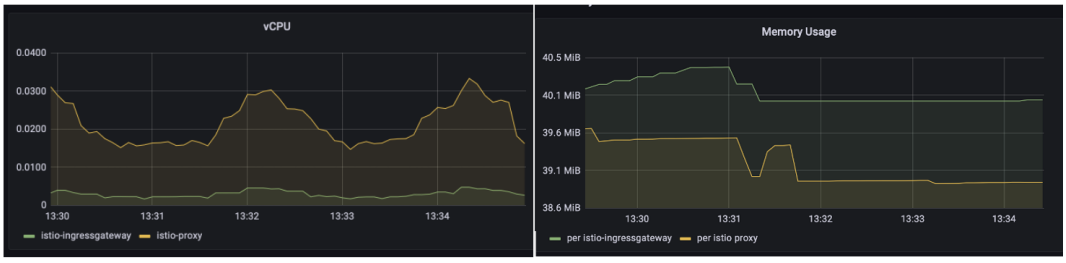


Figure 15: CPU and Memory usage

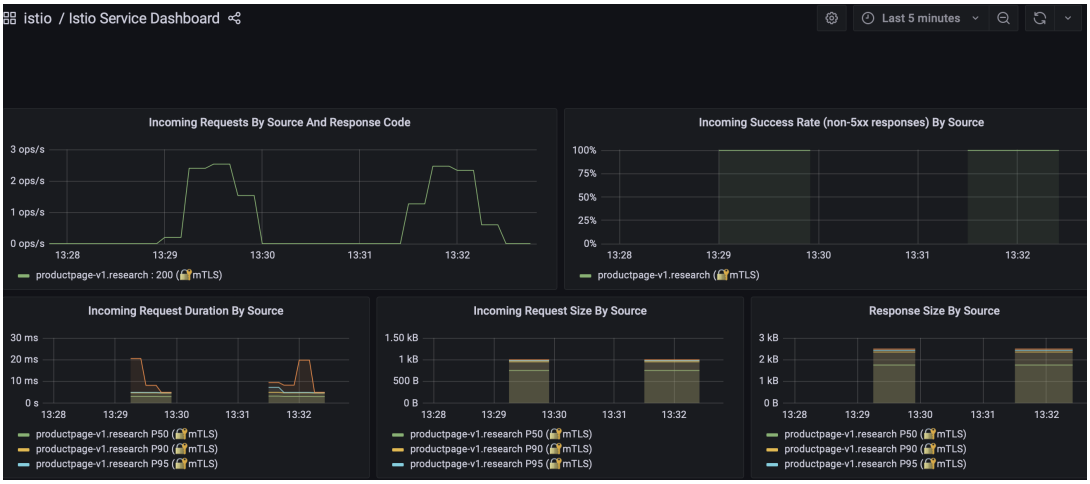


Figure 16: mTLS Connection Between Pods

- Next check the mTLS secured communication in the cluster. It can be seen in the istio workload dashboard and the mTLS is linked with application pods.
- The Istio control plane components are consolidated into a single binary through the use of Istiod. Figure [17] shows the Istiod graph in terms of CPU and Memory.

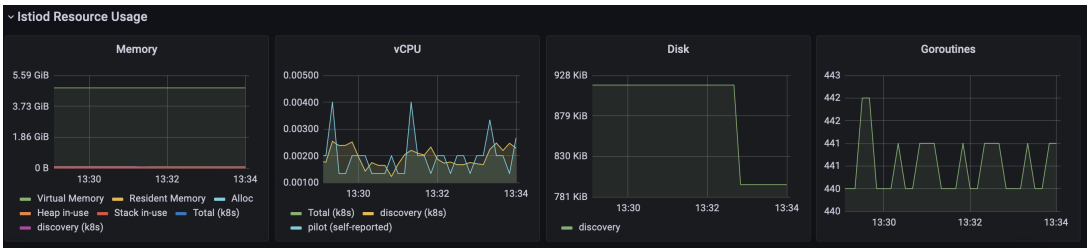


Figure 17: Resource Usage

- Istio proxy also contains some amount of space and size in the cluster and because it is at the top of the container it is important to see the consumption Istio proxy is taking in terms of CPU and Memory. Figure [18] shows the usage of Istio Proxy. It can be seen that Istio proxy is taking very less space and it's not affecting the application in terms of latency and delay.

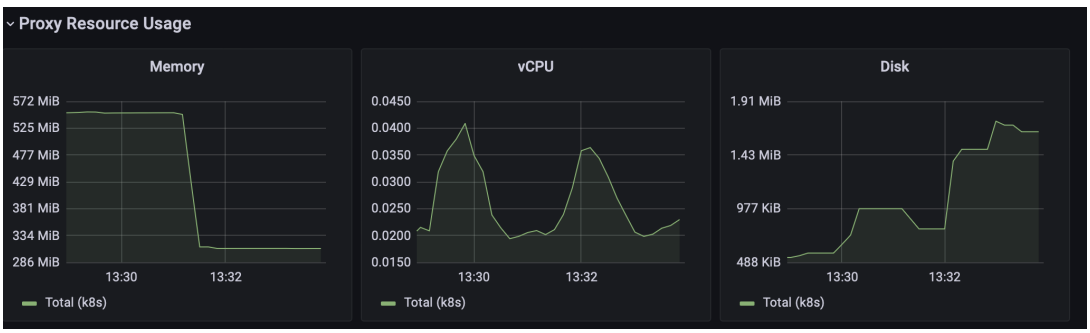


Figure 18: Istio Proxy Resource Usage

6.1 Discussion

The evaluation shows that even after installing Istio tool and configuring it with the application the CPU and Memory usage of the infrastructure machines are very less, which will not affect the application performance. It can be seen in Figure [16] that the internal service and pods connection within the cluster is mTLS secured and Figure [12] shows the request flowing to the cluster is through Istio ingress gateway. The application used in the research is very lightweight and does not contain complex code but still Memory and CPU usage of the application and Istio is very minimal so it can be concluded that complex application and Istio usage will also be less and will not affect the performance of the application. Regarding Trivy scanning Figure [8] [9] shows the comparison of the CVEs of different docker images. If dependencies, library, and code are written in the correct manner then the vulnerability will be very less, and if the exposure of the vulnerability is high then it can be seen in the output of the scan. This research solves the security problem of the Docker image and Kubernetes. The objective of the research question has been successfully implemented.

7 Conclusion and Future Work

The goals of this study, which were to scan container images and ensure safe communication between different services in Kubernetes, were accomplished. The Trivy tool has been used for both the scanning of containers and the deployment of Istio along with service mesh for Kubernetes. In the course of this investigation, standard procedure was carried out, and the remaining works on the previously published articles were brought to a successful conclusion.

There are some study has been done for Istio implementation but they have focused on the specific company and its infrastructure. The study shows the perception of developers that how Istio is beneficial for them and the result after implementation shown in terms of CPU and Memory usage, they have covered latency and delay to the application pod as well. But this research is completely different from the paper as it uses AWS cloud and a completely different infrastructure with different tools and services. Evaluation is also different from the online study as the metrics have been used if the dashboard of the Istio add-ons takes the metrics from the application and showing on the dashboard.

In the course of the study, I came into a few obstacles but, thanks to the helpful assistance of the local community as well as online journals and articles, I was able to overcome these obstacles and discover answers. The work that needs to be done in the future for this includes notifying the DockerHub repository about the malicious image, as well as developing a reporting process and conducting an investigation into these images. And some more evaluation can be done in Istio if the infrastructure is big and the application has more load.

References

Abhishek, M. K. and Rao, D. R. (2021). Framework to secure docker containers, *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, IEEE, pp. 152–156.

Beltre, A. M., Saha, P., Govindaraju, M., Younge, A. and Grant, R. E. (2019). En-

abling hpc workloads on cloud infrastructure using kubernetes container orchestration mechanisms, *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, IEEE, pp. 11–20.

Chamoli, S. et al. (2021). Docker security: Architecture, threat model, and best practices, *Soft Computing: Theories and Applications*, Springer, pp. 253–263.

Henriksson, O. and Falk, M. (2017). Static vulnerability analysis of docker images.

Hussain, F., Li, W., Noye, B., Sharieh, S. and Ferworn, A. (2019). Intelligent service mesh framework for api security and management, *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, pp. 0735–0742.

Kalubowila, D., Athukorala, S., Tharaka, B. S., Samarasekara, H. R., Arachchilage, U. S. S. S. and Kasthurirathna, D. (2021). Optimization of microservices security, *2021 3rd International Conference on Advancements in Computing (ICAC)*, IEEE, pp. 49–54.

Larrucea, X., Santamaria, I., Colomo-Palacios, R. and Ebert, C. (2018). Microservices, *IEEE Software* **35**(3): 96–100.

Li, W., Lemieux, Y., Gao, J., Zhao, Z. and Han, Y. (2019). Service mesh: Challenges, state of the art, and future research opportunities, *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, pp. 122–1225.

Mara Jösch, R. (2020). Managing microservices with a service mesh: An implementation of a service mesh with kubernetes and istio.

Miles, S. (2020). Kubernetes: A step-by-step guide for beginners to build, manage, develop, and intelligently deploy applications by using kubernetes (2020 edition), *Independently Published. isbn: 9798613623808 https://books. google. com/books* .

Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J. and Josuttis, N. (2017). Microservices in practice, part 1: Reality check and service design, *IEEE software* **34**(01): 91–98.

Shamim, M. S. I., Bhuiyan, F. A. and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices, *2020 IEEE Secure Development (SecDev)*, IEEE, pp. 58–64.

Valance, J. (2019). Adding container security and compliance scanning to your aws codebuild pipeline", anchore blog.

Yasrab, R. (2018). Mitigating docker security issues, *arXiv preprint arXiv:1804.05039* .