National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Manu Kumar Kudaragundi Channappa
Student ID: 23200341

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Manu Kumar Kudaragundi Channappa
Student ID:	23200341
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Ahmed Makki
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1559
Page Count:	10

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:	Manu Kumar Kudaragundi Channappa
Date:	12th December 2024

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):	

Configuration Manual

Manu Kumar Kudaragundi Channappa
23200341

1 Introduction

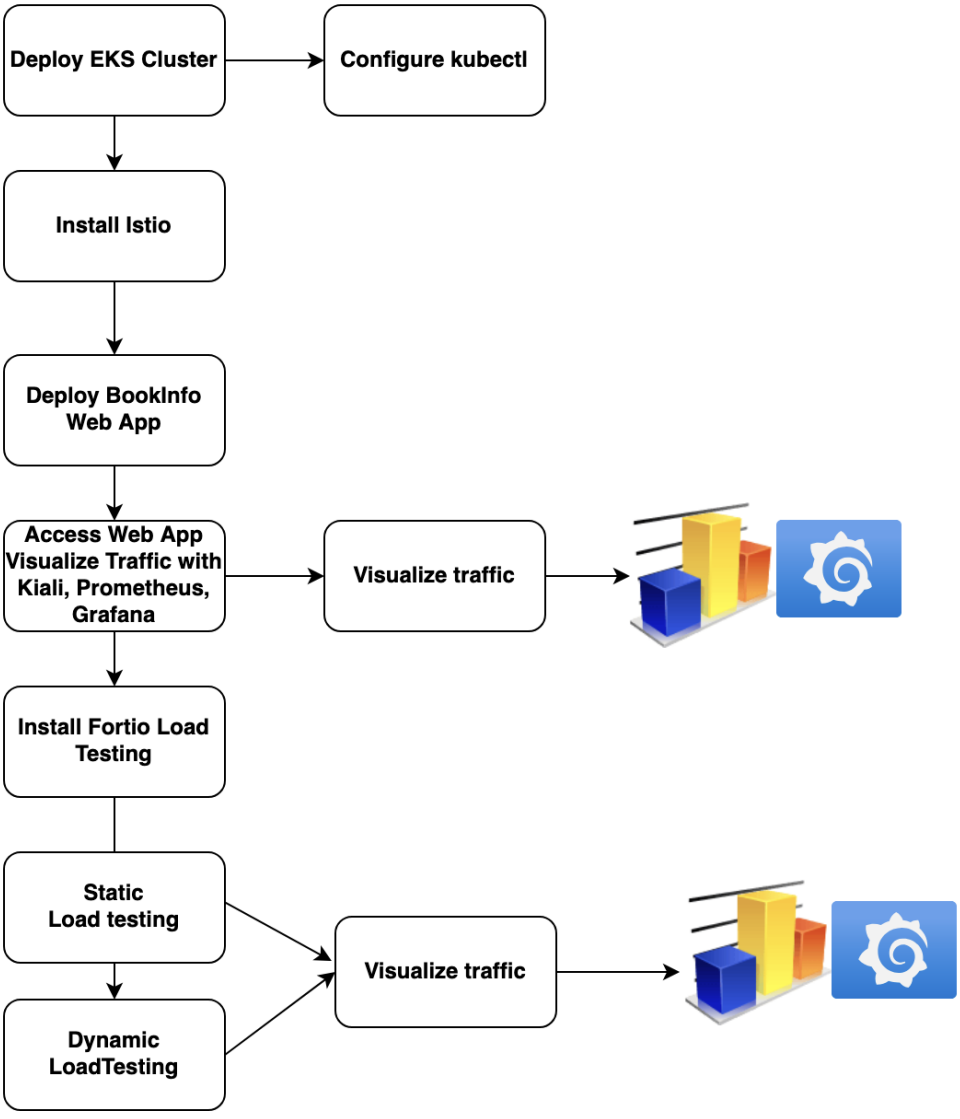

Figure 1: Flow Chart of the Project Configuration

The implementation of the Project "Smart Circuit Management: Enhancing Kubernetes Service Mesh through Dynamic Circuit Breaker" is focused on providing resilient

and robust microservice architec- ture. The process begins with deployment of Kubernetes Cluster on AWS EKS, establishment of solid foundation of microservices. Integrating with Istio Service Mesh follows by providing advanced traffic management and enhancing the security. The metric monitoring is collected by real time enablement from Grafana, set to Visualize for the observability offering critical insights. Rigorous testing is conduc- ted on the Istio Service Mesh Sheikh et al. (2018)by providing variable load on application,

Fortio load testing tool implied to produce testing environment on circuit breaking principle of Service Mesh, which continuously tests the resiliency of the system, Dynamic Circuit breaker rule is carefully configured to produce dynamic capacity to the environment with settings automatically adjust to testing and fine-tuning of the system. Continuous monitoring With Metrics optimizing performance and reliability handling the system with variable load and swiftly recovering resilient from disruption environment. This is a systematic approach to significantly enhancing the stability of services, the configuration of implementation model procedures is given below for creating effective advanced circuit breaker for cloud-native environment. Throughout the Project process referred official Istio document and its reference given for Circuit breaking rules.

2 Deploying the AWS EKS cluster:

The foundation for the project is robust and scalable Kubernetes environment, using AWS EKS deploying Kubernetes Cluster by managed service which provides high availability and scalability. The facility of AWS EKS is to provide less operational overhead and complexity in managing the control plane of Kubernetes, thus for much complex microservice project using managed cluster service is preferred. The use AWS EKS cluster in

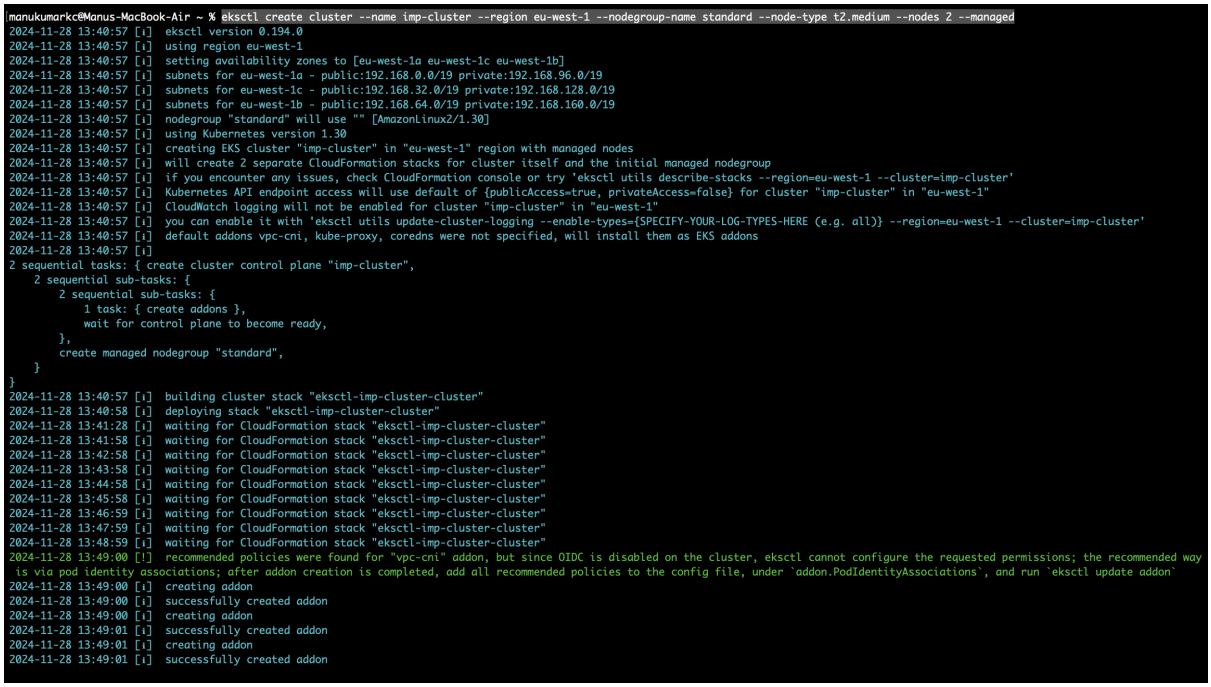

Figure 2: Deploying the AWS EKS cluster using CLI

eu-west-1 region in specific for less latent and high availability feature in the system, the

Kubernetes service offering container orchestration and enabling efficient management of micro-services, helps in automated scaling and seamless deployments. Below is the descriptive image of the EKS cluster deployed on AWS by CLI, with deploying AWS CLISubramanian (2023), configuring "eksctl" command line tool for EKS accessing from the terminal and "kubectl" command line tool for Kubernetes cluster for accessing nodes and interacting with pods created as part of the project.

CLI Command to deploy AWS EKS Cluster in eu-west-1 region:
eksctl create cluster –name imp-cluster –region eu-west-1 –nodegroup-name standard – node-type t2.medium –nodes 2 –managed

3 Deploying Istio Service Mesh:

The next process in implementation is Deploying Istio Service Mesh, As it is a Open source Service Mesh which seamlessly integrates with Kubernetes services to manage, secure and monitor communication in microservice environment. The Istio Service mesh with the component Envoy proxyMara Jösch (2020) to handle traffic and providing advance features for traffic splitting, retry mechanism and intelligent time-out en- hancing system reliability. As the main feature of Istio is its ability in circuit breaking feature with the Destination rule configured, The Destination rule policy as maximum connections, pending request and outlier detection to identify and isolating unhealthy instances helps in preventing cascading inter- connected micro-services failure. The Methodology in adapting to the real-time environment to adjust these parameters with metrics change, Istio ensures microservice system remain resilient and its op- timal performance under variable load conditions, thus maintaining reliable secure conditions and secure connections. Command to Deploy Istio Service Mesh in EKS Cluster:

Figure 3: Deploying Istio Service Mesh using CLI

curl -L https://istio.io/downloadIstio — sh -

4 Deploying BookInfo Application:

The BookInfo application is a microservice application which is designed to showcase the features and ability of Istio Service Mesh. The Total application consists of four main microservices, Product Page, Details Reviews and Ratings. The Product Page microservice

is the entry point of the application, from product page calling Details and Reviews which provides further information and user reviews in the application load balancer. All four microservices are in together, with Reviews Service has three Versions helps in demonstrating in Istio traffic management. On Deploying Istio-Kubernetes cluster enabled, 2 Envovy proxy are automatically injected on each pod microservice pod for advanced traffic management, Observability and security feature. This overall setup of Istio Service mesh helps in exploring Istio functionality for traffic routing, fault injection and Circuit breaking, Understanding the overall BookInfo application and its architecture makes an excellent choice for Istio Service Mesh performance testing.
Command to Create BookInfo Application and to Open the Service to Access the application: Install BookInfo Application:
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.24/samples/bookinfo/platform/kube/bookinfo.yaml

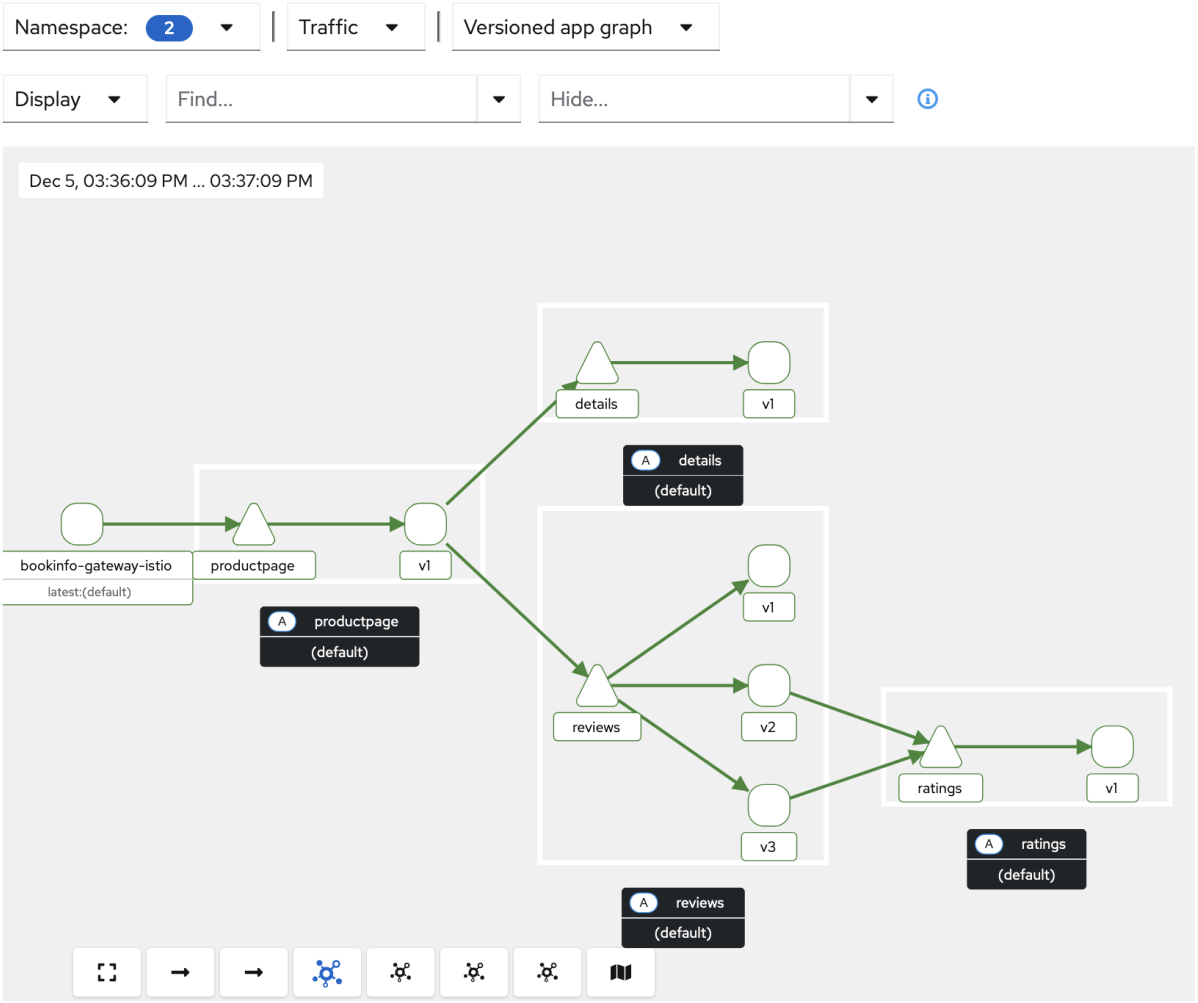

Figure 4: Deploying BookInfo Application in AWS CLI visualized via Kiali Dashboard

5 Creating Destination Rule (Static Rule):

The most Impactful process in the Istio Service Mesh is a step in creating Destination Rule, Managing traffic policies and ensuring resilient interactionsWang and Ma (2019).

A Destination Rule defines the traffic intended for Service Mesh policies which allows setting up load balancing, connections pool size and circuit breaking, The Rule defines the set of attributes in managing maximum connections, outlier detection and pending request in the connection pool size, helping the outlier and other identity which helps in eject unhealthy instance ensuring they do not effect overall system performance. By defining Destination rule in Yaml file, we gained fine grained control over how request are handled, enhancing the system performance and reliability in the microservice architecture. Thus Destination rule in Istio enabled you to implement advanced traffic management strategies ensures secure, reliable and effective microservice communication.

```
manukumarkc@Manus-MacBook-Air ~ % kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
EOF
```

Figure 5: Static Circuit Breaking rule created at destinationrule.yml file

Command to create Circuit Breaker Rule with Static Circuit Breaking Configuration: Command to Deploy httpbin service: Below is the Screenshot of Destination Rule created with Static Configuration:
kubectl apply -f samples/httpbin/httpbin.yaml
Command to Create Destination Rule in YAML file, Configure YAML file with Static rule provided by Istio Service Mesh:
Step1: Create YAML file with name destinationrule.yml
Step2: Apply the rule with Kubectl apply –f destinationrule.yml

6 Testing the circuit breaking in Service Mesh with Static and Dynamic Principle rule:

Steps for Load Testing:
Step1:Creating a Fortio Client:
kubectl apply -f samples/httpbin/sample-client/fortio-deploy.yaml
Step2: test the Client Fortio:
export FORTIOPOD = (kubectlgetpodslapp= f ortioojsonpath= .items[0].metadata.name)
ortio/usr/bin/f ortiocurlquiethttp : //httpbin : 8000/get
Step3: Tripping the Circuit with Load injecting on Application:
Part1: Accessing the application and Producing Load with 100 requests with 1 concor- rent connections per request.
for i in(seq 1 100); do curl -s -o /dev/null "http://localhost:8080/productpage"; done
Part2: with –c2 and 20 requests.
kubectl exec "FORTIOP OD" //httpbin : 8000/get cf ortio/usr/bin/f ortio-loadc2qps0n20loglevelW arninghttp :
Part3:-c3 and 30 requests.
kubectl exec "FORTIOP OD"cf ortio/usr/bin/f ortioload c3qps0n30loglevelW arninghttp : //httpbin : 8000/get
Part4:Check the number of Pending Request per connections:
kubectl exec "FORTIOP OD" cistioproxypilotagentrequestGET stats—grephttpbin—grep
Step4: Note down the readings of Static Circuit Breaker and Dynamic Circuit Breaker:
A tabular results is listed to identify the nature of Circuit Breaking principles and number of Pending request remaining which is a direct marker for resiliency in the system and maintaining reliability.

6.1 Static Principles:

Defining Istio's Destination Rule setting up the static parameters with the static rule of Destination Rules which are predefined by Istio Service Mesh and are set to static parameter conditions with maximum number of connections, pending request and error thresholds. The set parameters with Destination Rule with limited connections for preventing overload. Testing this system with static parameter rule with loadtesting is generated by Fortio load testing tool, Which generates a consistent variable load on the Service Mesh. The real time metrics are visualized for latency and error rates are observed to ensure the circuit breaking trips and testing the environment with the static load system, circuit breaking trips are carefully examined and assessed to set of features which check threshold limit isolating fault services and maintaining system stability.

6.2 Dynamic Principles:

Dynamic Circuit breaker principles are defined on carefully examining the static circuit breaker tripping rate which has the capacity to adjust real-time based load on the system, thus changing the Destination Rule with capacity to adjust variable dynamic load will increase the efficiency of the system to adapt to real-time environment in microservice

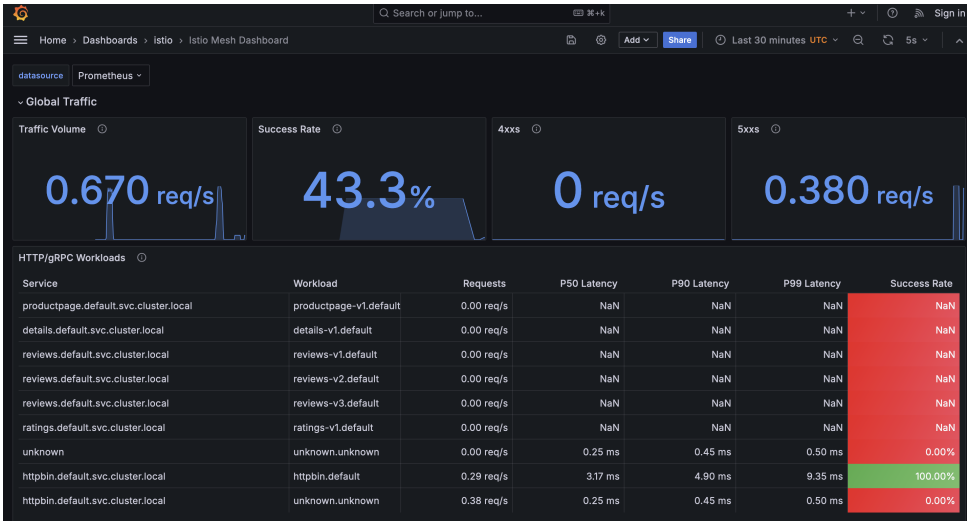

Figure 6: Tripping of Static Circuit Breaker Rule

architecture, thus connection limits and thresholds are calculated with real-time data collected by monitoring tools and assessed.With changing the Dynamic load on the system influenced by Fortio for variable traffic patterns continuous supply of load to the Istio Mesh is processed and adjust to set the closed state of the system with very minimal amount of pending connection waiting request set. The Results discussed on the report for

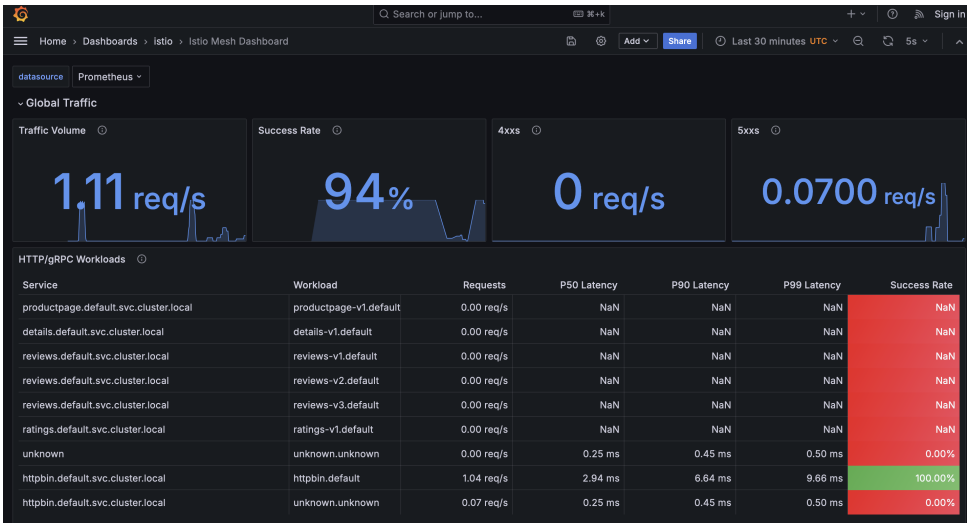

Figure 7: Tripping of Dynamic Circuit Breaker Rule

load testing from Fortio on Both Static and Dynamic Circuit breaker are helped in making a lime marker for changes that needs to be integrated as part of static configuration rule to changed into dynamic rule for adapting its natures variable environment making the system more reliant and reliable. The patterns of the result collected and examined in Tabular format as shown in the Project Report, The Results shows the nature of Static Circuit Breaking strategy and Dynamic Circuit Breaking Strategy for which the number of Pending requests are the deciding factor for continuous load testing on the micro-service system Architecture.

7 GitHub Repository:

All the Bash Code of this Project is Uploaded to GitHub Public Repository with Step-wise naming of each integration and declaration: GitHub Repository:

https://github.com/manukumarkc/ThesisCodeArtifact.git

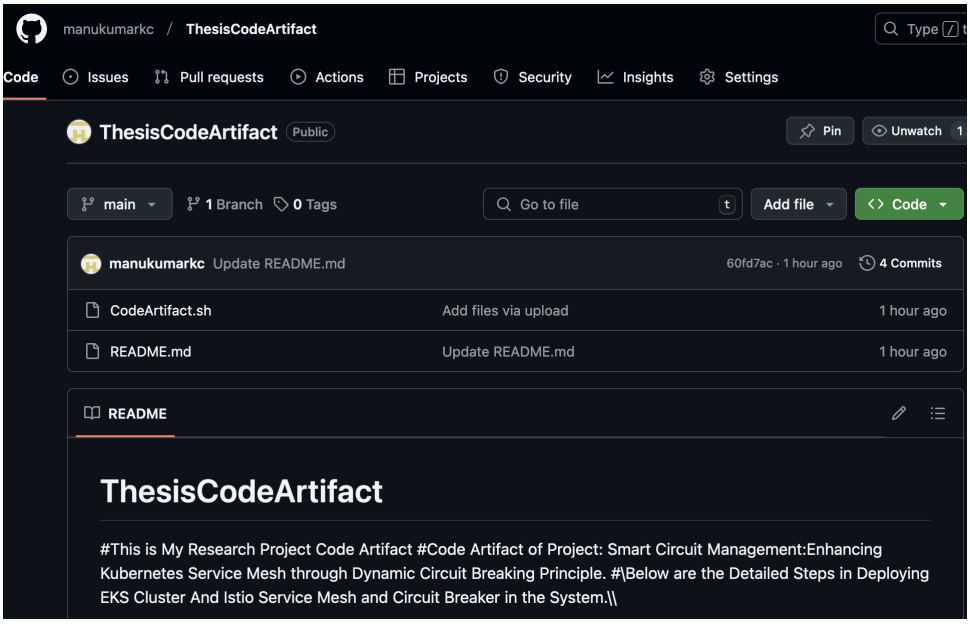

Figure 8: Tripping of Dynamic Circuit Breaker Rule

8 Documents Referred:

All the Documents used for creating EKS pod, Creating Istio Service Mesh and Creating Static configuration rule, Deploying BookInfo application and Load testing with Fortio is given in the below for Reference:

Kubernetes in AWS Cloud:
https://aws.amazon.com/eks/

Kubernetes Cluster Installation:
https://kubernetes.io/docs/tutorials/hello-minikube/

Istio Service Mesh:
https://istio.io/latest/docs/setup/getting-started/

Istio Circuit Breaking Rule:
https://istio.io/latest/docs/tasks/traffic-management/circuit-breaking/

Istio Real-Time Data Visualization:
https://istio.io/latest/docs/tasks/observability/metrics/using-istio-dashboard/

9 Complete Execution of Bash Commands:

From Deploying EKS cluster in AWS EKS to creating Istio Service Mesh and Creating Static Circuit Breaking rule to Istio Service Mesh, tetsing the Features of Istio Service Mesh with Fortio Loading of the System. Dynamically adjusting the system with checking incoming Load, Compute capacity, overall conditions, Below are the detailed steps from step1-step35 which covers entire execution of Bash commands in installing.

to start the installation of the Project, We need to have the access of AWS CLI or Command Line Terminal:

From the GitHub Repository :
https://github.com/manukumarkc/ThesisCodeArtifact.git

all the commands are set step-wise to install EKS Cluster, Install Istio and Create Circuit BReaking BookInfo application: Below is the Image of Repository for commands from Step1 to Step 35, which complete the complete installation of Istio Service Mesh and Complete its Dynamic circuit breaking rules with current compute load trends.

```
# Step 1: Download AWS CLI (Mac OS)
echo "Downloading AWS CLI..."
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws
sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/aws_completer

# Step 2: Download EKSCTL Version
echo "Downloading EKSCTL..."
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.gz"
# (Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check
tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
sudo mv /tmp/eksctl /usr/local/bin

# Step 3: Install Kubernetes Kubectl version 1.30
echo "Installing Kubernetes Kubectl..."
curl -O "https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.6/2024-11-15/bin/darwin/amd64/kubectl"
# Execute permission for binary
chmod +x ./kubectl
# Export the path
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH

# Step 4: Check Version of EKSCTL and KUBECTL
echo "Checking versions..."
eksctl version
kubectl version --client

# Step 5: Deploy Nodes on the EKS cluster
echo "Deploying nodes on EKS cluster..."
eksctl create cluster --name project-cluster --region eu-west-1 --nodegroup-name standard --node-type t2.medium --nodes 2 --managed
eksctl get cluster

# Step 6: Update the EKS Configuration
echo "Updating EKS configuration..."
```

Figure 9: GitHub Repository Commands file for complete installation:

References

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

Sheikh, O., Dikaleh, S., Mistry, D., Pape, D. and Felix, C. (2018). Modernize digital applications with microservices management using the istio service mesh, *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, pp. 359–360.

Subramanian, S. (2023). Operating the eks cluster, *Deploy Container Applications Using Kubernetes: Implementations with microk8s and AWS EKS*, Springer, pp. 287–328.

Wang, Y. and Ma, D. (2019). Developing a process in architecting microservice infrastructure with docker, kubernetes, and istio, *arXiv preprint arXiv:1911.02275* .