

## *My Portfolio*

### Hello I am MANISHA SHARMA

*I'm a full-stack developer and DevOps enthusiast with 3+ years of experience across the automotive, e-commerce, and banking sectors. My strength lies in building scalable web applications using Java, Spring Boot, ReactJS, and microservices—while leveraging CI/CD, Docker, Kubernetes, and cloud-native tools to drive modern software delivery. With a Master's in Applied Computer Science from Germany and a passion for innovation, I thrive at the intersection of clean code, system design, and emerging technologies like Web3 and serverless computing. I'm a proactive problem solver who values collaboration, continuous learning, and delivering real-world impact through tech.*

# Project 1: Docker-Java- Kubernetes

## Problem Statement

- Deploying and scaling Java-based microservices manually led to inconsistent environments, delayed releases, and operational inefficiencies.

## Challenges

- Manual setup and deployment pipelines
- Difficult to ensure consistent environments across dev, test, and production
- Lack of horizontal scalability

## Approach

- Containerized the Java microservice using Docker and deployed it on Kubernetes. YAML-based deployment scripts were created to handle configuration, health checks, and scaling rules.

## Solution

- Built a production-ready CI/CD workflow using Jenkins and Docker
- Deployed Java microservices to Kubernetes with rolling updates
- Enabled high availability, simplified deployment, and container orchestration

**Stack:** Java, Spring Boot, Docker, Kubernetes, Jenkins, Maven, YAML

## Screenshot 1: Docker Image Build and Tagging

### Description:

Terminal output showing the successful Docker image build for the stockmanager, product catalogue and shopfront service using a custom Dockerfile, followed by a list of all Docker images available on the system.

*Highlights my ability to create optimized Docker images for microservices and manage local container repositories.*

```
root@control-plane:~/java_docker/docker-java-kubernetes-project/stockmanager
[root@control-plane stockmanager]# docker build -t thetips4you/stockmanager:latest .
Sending build context to Docker daemon 43.57MB
Step 1/4 : FROM openjdk:8-jre
----> 155efed40fd4
Step 2/4 : ADD target/stockmanager-0.0.1-SNAPSHOT.jar app.jar
----> 91e2188edc77
Step 3/4 : EXPOSE 8030
----> Running in 8c76a37e5383
Removing intermediate container 8c76a37e5383
----> 98f56494d1c2
Step 4/4 : ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
----> Running in 36840cd6266a
Removing intermediate container 36840cd6266a
----> 14d4b4001380
Successfully built 14d4b4001380
Successfully tagged thetips4you/stockmanager:latest
[root@control-plane stockmanager]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
thetips4you/stockmanager	latest	14d4b4001380	3 seconds ago	317MB
thetips4you/productcatalogue	latest	e3fea39201ce	2 minutes ago	291MB
thetips4you/shopfront	latest	8e41d1cdc23e	4 minutes ago	320MB
openjdk	8-jre	155efed40fd4	3 weeks ago	274MB
bitnami/kube-state-metrics	2.4.2-debian-10-r18	5c9d2e6d338c	2 months ago	122MB
k8s.gcr.io/kube-state-metrics/kube-state-metrics	v2.4.1	74a618d7bae7	4 months ago	38.7MB
prom/pushgateway	v1.4.2	a0ebc6b7eba3	8 months ago	19.9MB
jimmidyson/configmap-reload	v0.5.0	d771cc9785a1	17 months ago	9.99MB
k8s.gcr.io/kube-proxy	v1.20.0	10cc881966cf	18 months ago	118MB
k8s.gcr.io/kube-scheduler	v1.20.0	3138b6e3d471	18 months ago	46.4MB
k8s.gcr.io/kube-apiserver	v1.20.0	ca9843d3b545	18 months ago	122MB
k8s.gcr.io/kube-controller-manager	v1.20.0	b9fa1895dcaa	18 months ago	116MB
gcr.io/k8s-minikube/storage-provisioner	v4	85069258b98a	18 months ago	29.7MB
k8s.gcr.io/etcd	3.4.13-0	0369cf4303ff	22 months ago	253MB
bitnami/prometheus-operator	0.41.0-debian-10-r5	2404b4396a48	22 months ago	128MB
bitnami/kube-state-metrics	1.9.7-debian-10-r51	1bef6cdfecbb	22 months ago	127MB
k8s.gcr.io/coredns	1.7.0	bfe3a36ebd25	2 years ago	45.2MB
k8s.gcr.io/pause	3.2	80d28bedfe5d	2 years ago	683kB

## Screenshot 2: Kubernetes Pod and Deployment Status

### Description:

kubectl commands showing successful deployment and live pod status for the Product Catalogue, Shopfront, and Stock Manager services in a Kubernetes cluster.

*Demonstrates service orchestration, deployment creation, and pod lifecycle management.*

```
root@control-plane:~/java_docker/docker-java-kubernetes-project/kubernetes
[root@control-plane kubernetes]# ls
productcatalogue-service.yaml shopfront-service.yaml stockmanager-service.yaml
[root@control-plane kubernetes]# kubectl apply -f productcatalogue-service.yaml
-bash: kubectl: command not found
[root@control-plane kubernetes]# kubectl apply -f productcatalogue-service.yaml
service/productcatalogue created
deployment.apps/productcatalogue created
[root@control-plane kubernetes]# kubectl apply -f stockmanager-service.yaml
service/stockmanager created
deployment.apps/stockmanager created
[root@control-plane kubernetes]# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
productcatalogue-fbdbd9f95-j2xf4    0/1     ContainerCreating   0           12s
shopfront-66675cbc9f-g2qt1          1/1     Running             0           109s
stockmanager-bffcc5fbc-8dm1z        0/1     ContainerCreating   0           5s
[root@control-plane kubernetes]# kubectl get deployments
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
productcatalogue 1/1     1            1           20s
shopfront        1/1     1            1           117s
stockmanager     0/1     1            0           13s
[root@control-plane kubernetes]# kubectl get deployments
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
productcatalogue 1/1     1            1           35s
shopfront        1/1     1            1           2m12s
stockmanager     0/1     1            0           28s
[root@control-plane kubernetes]# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
productcatalogue-fbdbd9f95-j2xf4    1/1     Running            0           39s
shopfront-66675cbc9f-g2qt1          1/1     Running            0           2m16s
stockmanager-bffcc5fbc-8dm1z        0/1     ContainerCreating   0           32s
[root@control-plane kubernetes]# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
productcatalogue-fbdbd9f95-j2xf4    1/1     Running            0           55s
shopfront-66675cbc9f-g2qt1          1/1     Running            0           2m32s
stockmanager-bffcc5fbc-8dm1z        1/1     Running            0           48s
[root@control-plane kubernetes]#
```

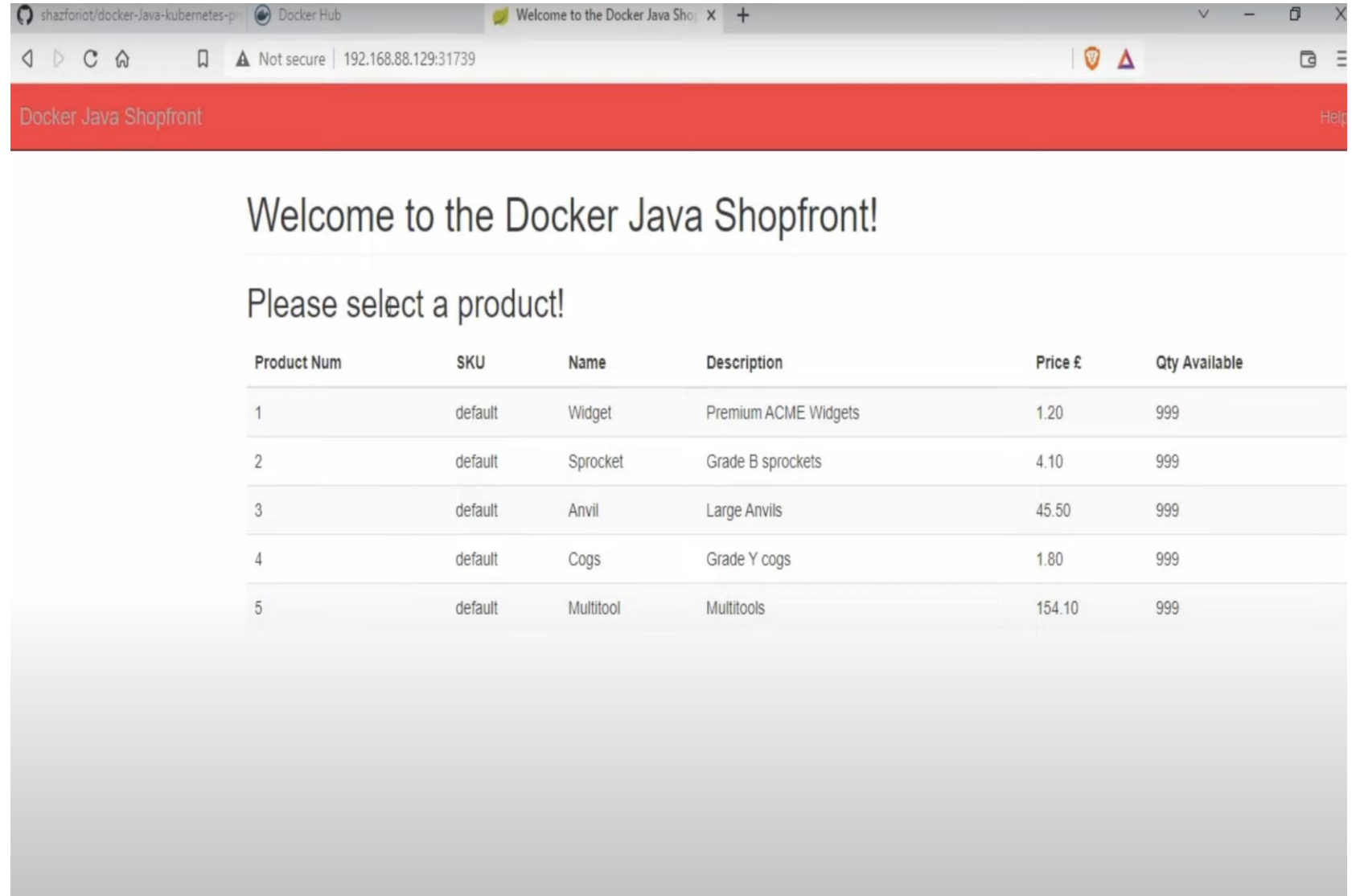


### Screenshot 3: Deployed Shopfront Web App

#### Description:

Live browser view of the Dockerized Java Shopfront application running inside Kubernetes, displaying dynamic product data

*Represents the final output of the CI/CD pipeline and a working microservices deployment.*



The screenshot shows a web browser window with the following details:

- Browser Tabs:** shazforiot/docker-java-kubernetes-pr, Docker Hub, Welcome to the Docker Java Sho...
- Address Bar:** Not secure | 192.168.88.129:31739
- Page Header:** Docker Java Shopfront (on a red background) with a Help link on the right.
- Main Content:**
  - Heading: Welcome to the Docker Java Shopfront!
  - Text: Please select a product!
  - Table with 5 products:

Product Num	SKU	Name	Description	Price £	Qty Available
1	default	Widget	Premium ACME Widgets	1.20	999
2	default	Sprocket	Grade B sprockets	4.10	999
3	default	Anvil	Large Anvils	45.50	999
4	default	Cogs	Grade Y cogs	1.80	999
5	default	Multitool	Multitools	154.10	999

# Project 2: Google Cloud API Gateway

## Problem Statement

- The client needed a low-maintenance, fast backend without traditional server overhead, to expose APIs securely and efficiently.

## Challenges

- Traditional server setups required constant maintenance
- Slow response time and infrastructure cost concerns
- Need for API standardization and secure access

## Approach

- Designed a serverless backend using Google Cloud Functions and exposed APIs through GCP API Gateway. Used OpenAPI specifications for standardized interface.

## Solution

- Built lightweight cloud functions for key logic
- Secured endpoints using GCP IAM + Gateway
- Achieved high performance with low operational costs and minimal latency
- **Stack:** Google Cloud Functions, API Gateway, OpenAPI (Swagger), Node.js/Python, GCP IAM

## Screenshot 1: API Gateway Creation Screen

### Description:

Creating a new API Gateway in Google Cloud Console with user-defined API ID and service endpoint.

*Demonstrates the setup of a managed API Gateway with custom configuration for scalable and secure backend routing.*

← → ↻ console.cloud.google.com/api-gateway/gateway/create?project=api-gateway-360218

Google Cloud api-gateway Search Products, resources, docs (/)

← Create gateway

### API

Select an API  
Create new API

Display Name \*  
user-api

API ID \*  
user-api

API ID can have lowercase letters, digits or hyphens. It must start with a lowercase letter and end with a letter or number.

### Managed service

user-api.apigateway.api-gateway-360218.cloud.goog

### Labels

+ ADD LABEL

### API Config

Select a Config  
Create new API config

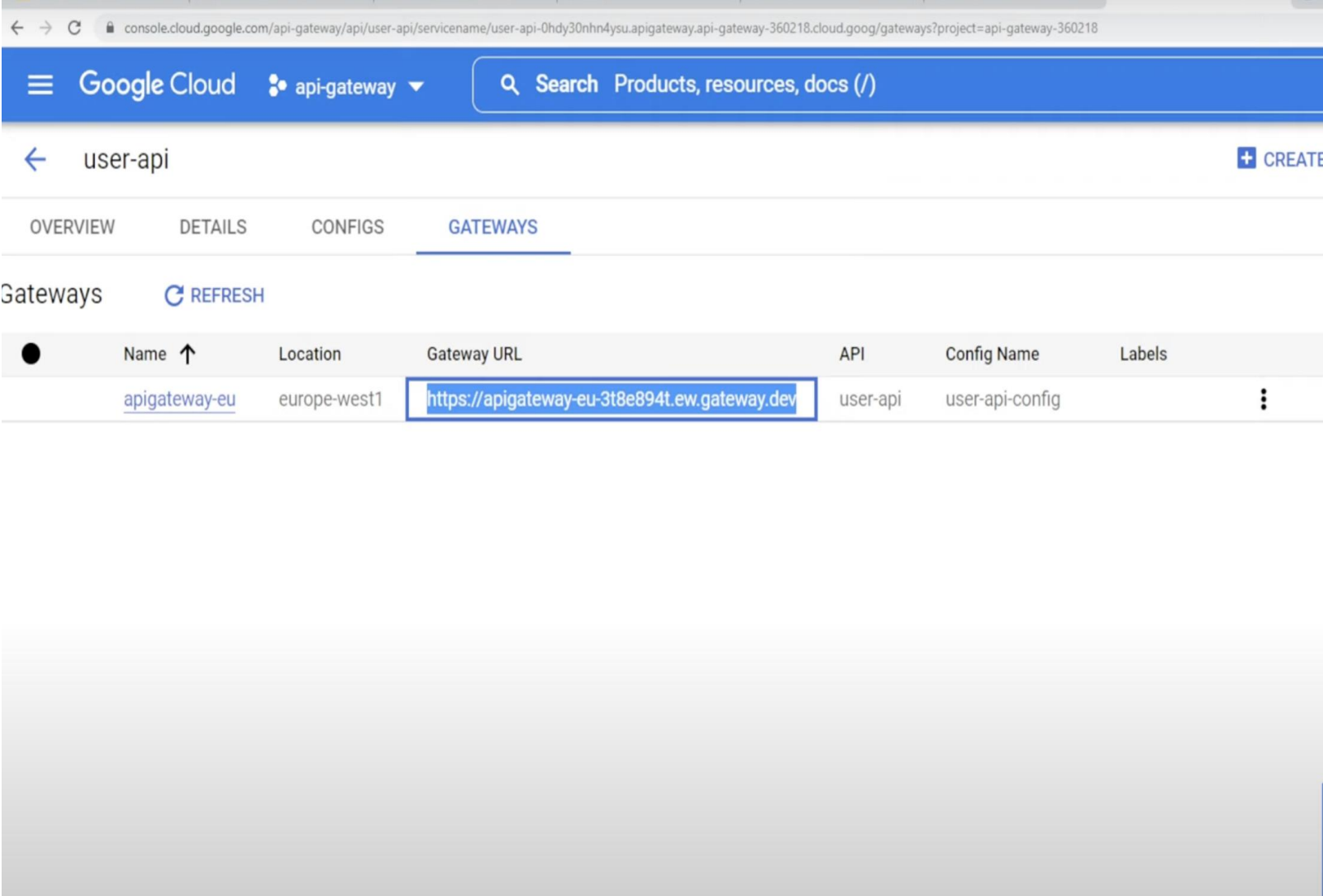
Lea

## Screenshot 2: Gateway Endpoint URL

**Description:**

Successful creation of the API Gateway with a live public-facing endpoint URL.

*Validates deployment success and production-readiness by exposing a secure gateway endpoint.*

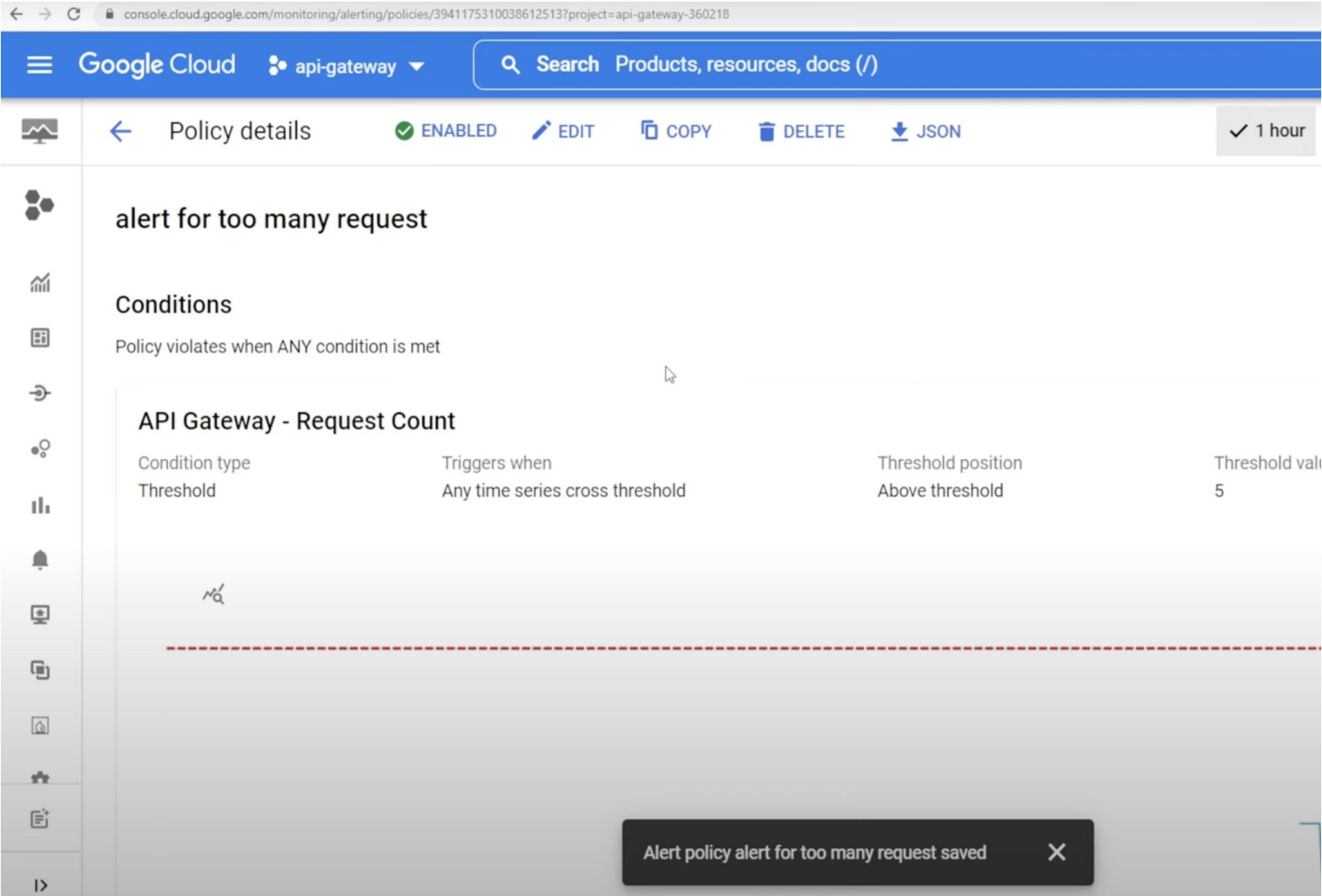




### Screenshot 3: API Gateway Alert Policy

**Description:**  
Monitoring alert setup in Google Cloud to track high request volume on the API Gateway.

*Highlights proactive observability and alerting for performance and usage tracking—critical for production stability.*



# Project 3: GitHub Actions CI/CD Pipeline

## Problem Statement

- Manual testing and deployment of Angular applications caused delays, human error, and inconsistent build environments.

## Challenges

- Time-consuming manual builds
- No automated test verification
- High chance of errors during deployment

## Approach

- Implemented GitHub Actions workflows to automate the build and testing phases of the Angular frontend on every push.

## Solution

- Automated CI/CD pipeline with GitHub Actions
- Ensured fast, consistent, and error-free builds
- Reduced manual effort and enabled fast iteration

**Stack:** Angular, GitHub Actions, YAML, npm, Jest, GitHub CI/CD

# Project 4: Web3 Supply Chain DApp (Master Thesis)

## Problem Statement

- Traditional supply chains lack transparency and are vulnerable to fraud, especially in multi-party automotive networks.

## Challenges

- No trust between suppliers, vendors, and customers
- Centralized systems hard to audit
- Manual tracking of orders and refunds

## Approach

- Developed a decentralized application (DApp) using Django and Ethereum smart contracts to manage and track supply chain events securely.

## Solution

- Enabled traceability of transactions using blockchain ledger
- Designed smart contracts for order/refund tracking
- Improved network-wide visibility and reduced fraud by 30%

**Stack:** Django, Web3.py, Solidity, Ethereum (Testnet), PostgreSQL, Python, Smart Contracts, MetaMask

## Screenshot 1: Blockchain Visualization in Supply Chain DApp

### Description:

A frontend view of the blockchain representation module in the decentralized supply chain application. It visualizes the sequence of blocks including transaction metadata such as sender, receiver, amount, and hash linkage.

*Demonstrates how blockchain ensures transparency and tamper-proof record-keeping across supplier and merchant transactions.*

Home Browse BMW Arihaat Vilosma BlockChain Log Out

### Block-Chain Representation

Search by Request ID or Order ID

Block Type : Genesis

Previous Hash : None

Hash : 41cfc0d1

Sender : Customer

Receiver : Company

Amount : € 89843.81

➔

Block Type : Block

Previous Hash : 41cfc0d1

Hash : b6c50821

Sender : Company

Receiver : Merchant

Amount : € 510.00

➔

Block Type : Block

Previous Hash : b6c50821

Hash : a6df854a

Sender : Merchant

Receiver : Fuel Supplier

Amount : € 400

#### What does block consist of?

1. A block consist of various details regarding itself and transactions.
2. A block has its hash code for data which can be used to verify with next block to check if data is tampered or not.
3. It also contains previous block hash to provide linkage and verify the previous block.
4. It contains other data like transactions, timestamp, etc.