
# Streamlining Software Release Process and Resource Management for Microservice based Architecture on Multi-Cloud

23-193

# Our Team

**Dr. Nuwan Kodagoda**
**Supervisor**

**Mr. Udara Samarathunge**
**Co-Supervisor**

**Herath H.M.I.P.**
**IT20125516**

**Jayawardena R.D.S.H**
**IT20074968**

**Fadhil M.R.A.**
**IT20784720**

**Gunarathne M.V.M.P.**
**IT19963402**

SLIIT
FACULTY OF COMPUTING

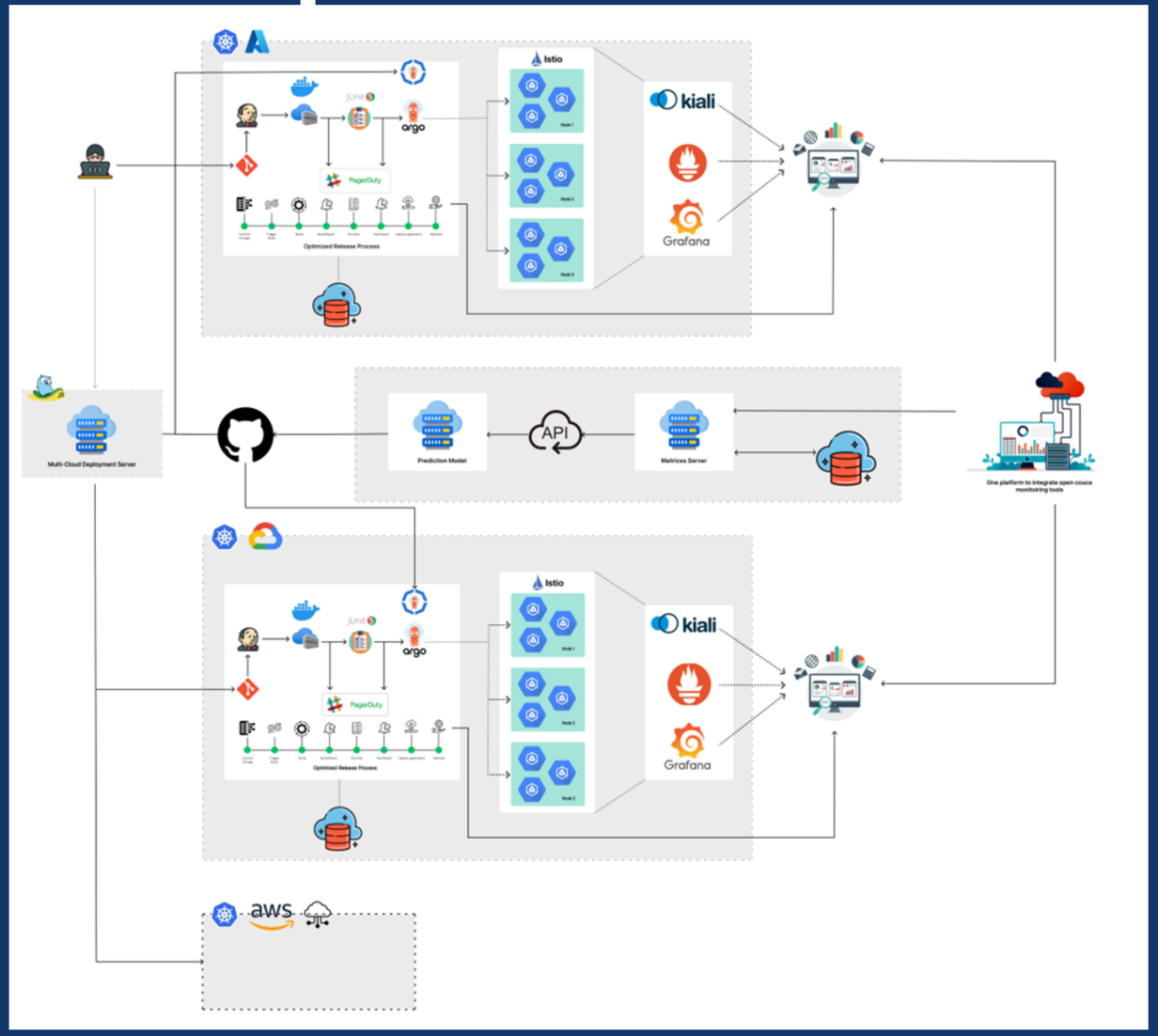
# Problems

- Available software release process systems are not fully automated and optimized.
- Available open-source monitoring tools' metrices need to be monitored in an one single platform.
- Cannot predict the necessary resource request and resource limit for VPAs.
- Available technologies are not focused to deploy applications in multiple cloud providers.

# Main Objective

**The overall objective of the research is to improve software development and deployment processes by addressing limitations in software release process automation, monitoring tool integration, resource allocation, and multi-cloud deployment.**

# Proposed Solution



# OPTIMIZING AND AUTOMATING SOFTWARE RELEASE PROCESS

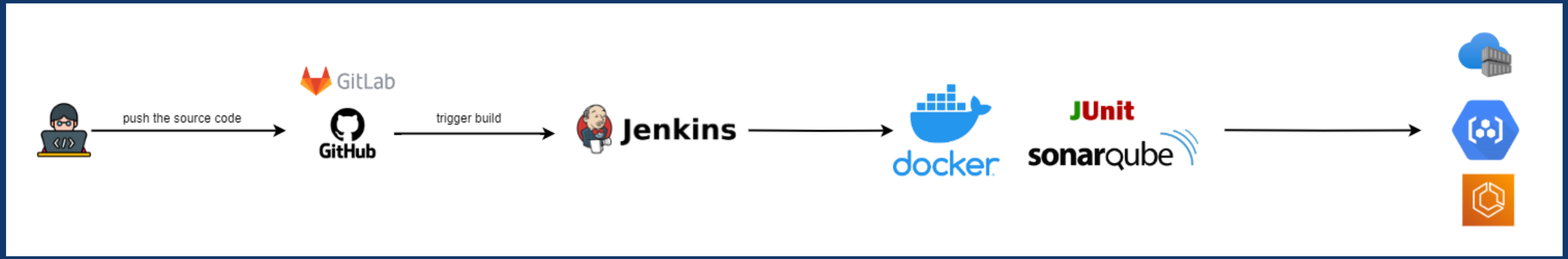
**Herath H.M.I.P. - IT20125516**
**BSc. Software Engineering (SLIIT)**

# Target Problem

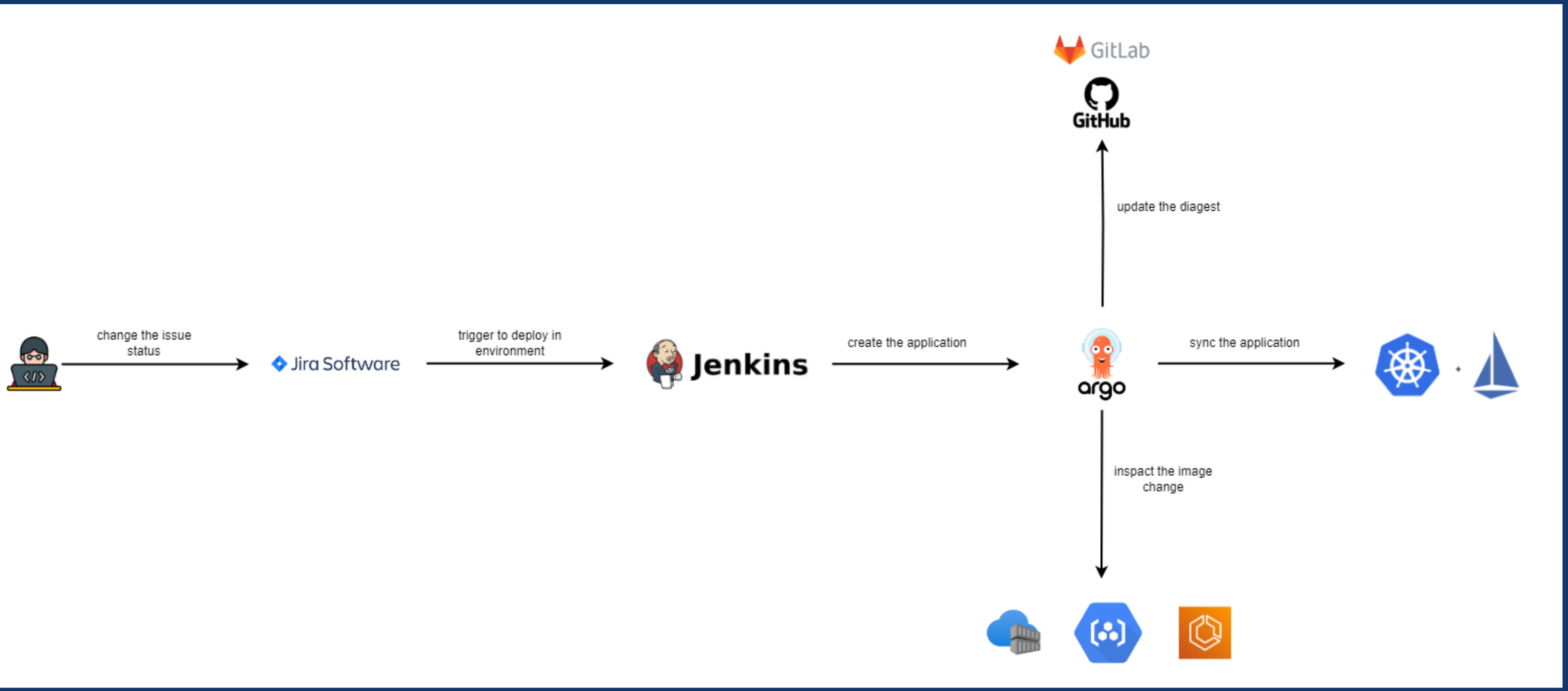
The existing methodologies for software release processes predominantly rely on manual execution, which poses several challenges such as suboptimal architecture, increased likelihood of human errors, excessive time consumption, and susceptibility to mistakes.

# Proposed Solution

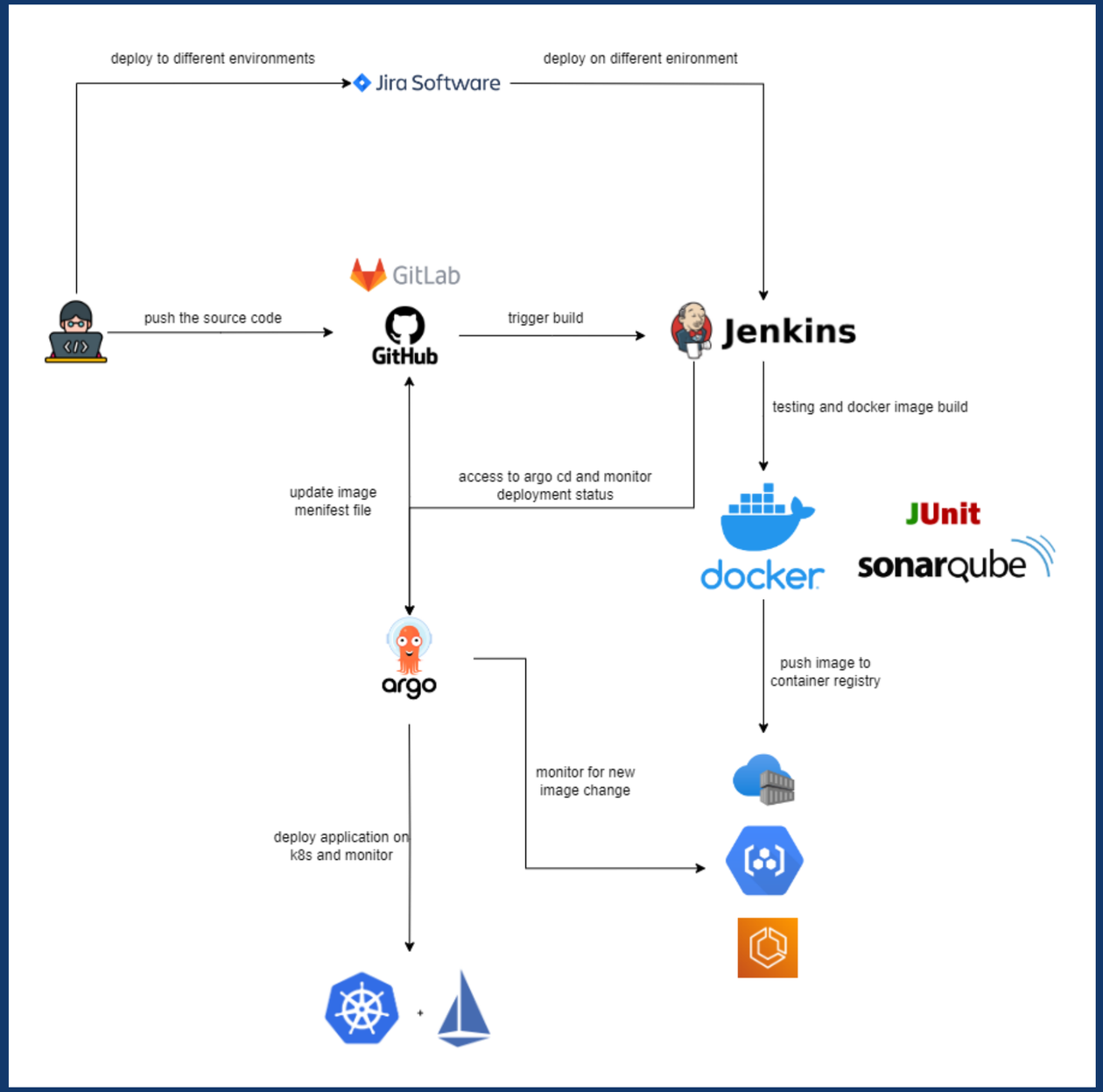
Rearchitect a software release process that can be applied commonly for most of the software release processes by optimizing the Software Release Process according to the architectural attributes such as performance, security, extendibility, reliability, and modifiability and fully automating the proposed software release process architecture.



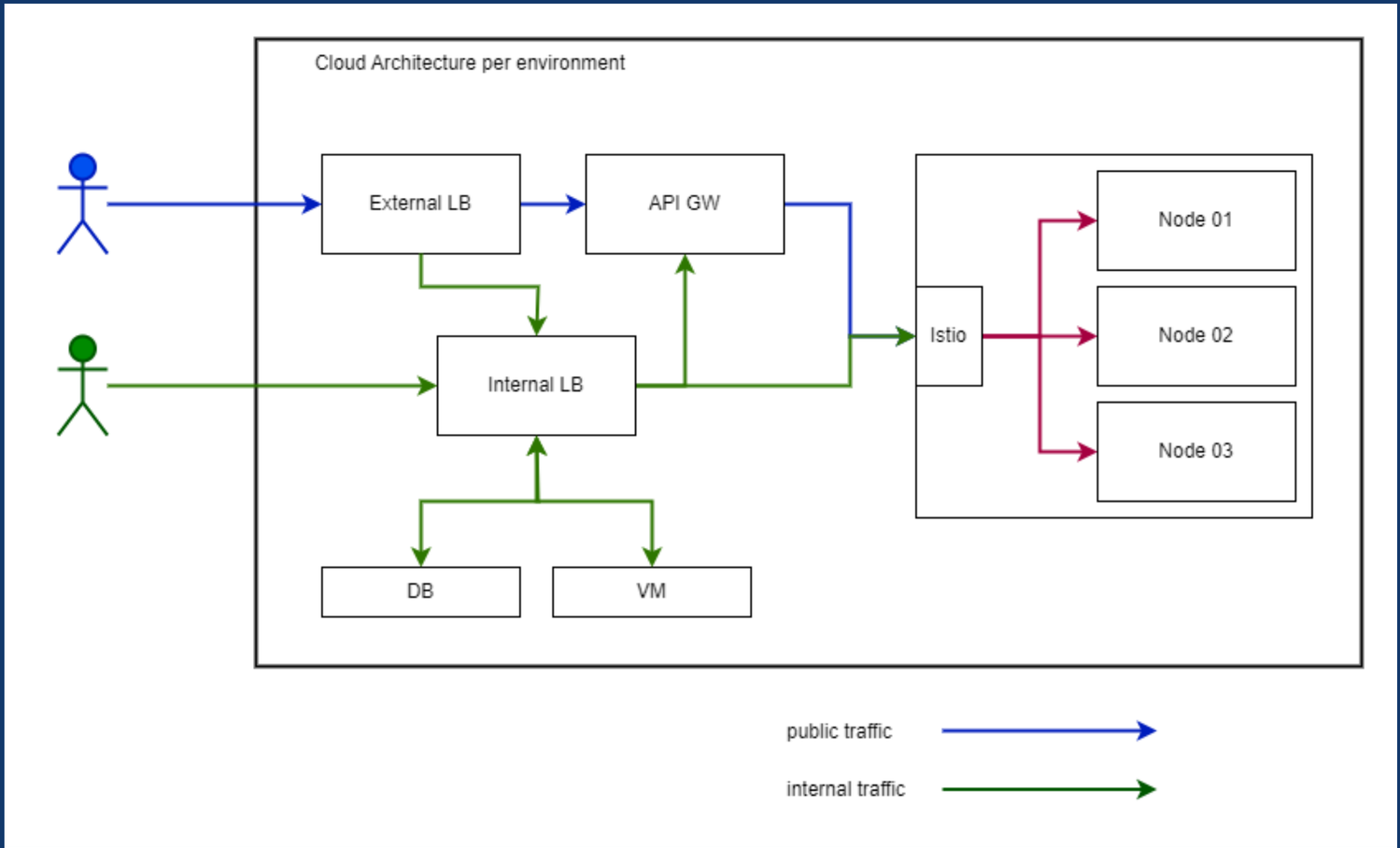

proposed CI Architecture




# proposed CD Architecture




# proposed CI/CD Architecture

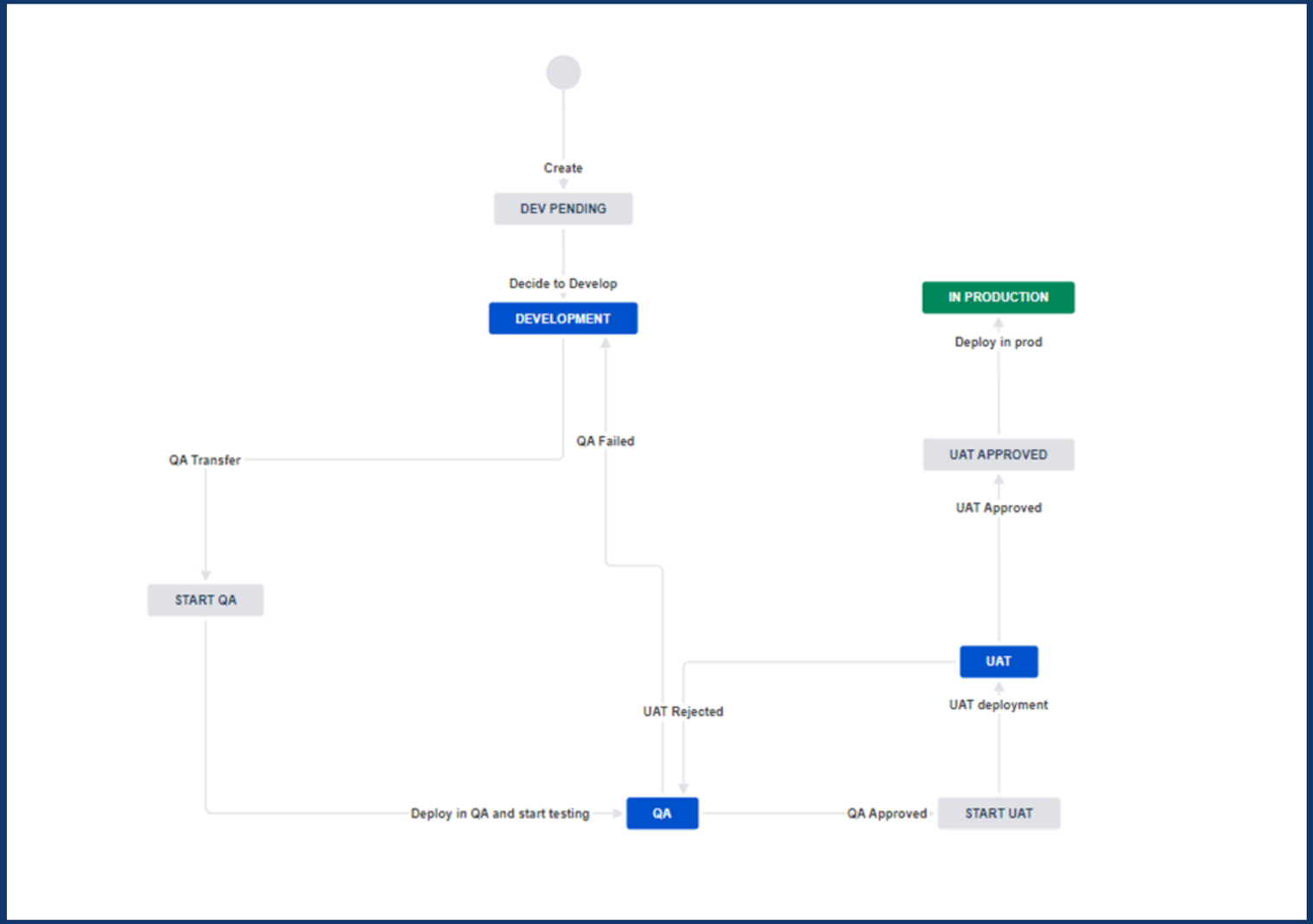



# Cloud Environment Architecture

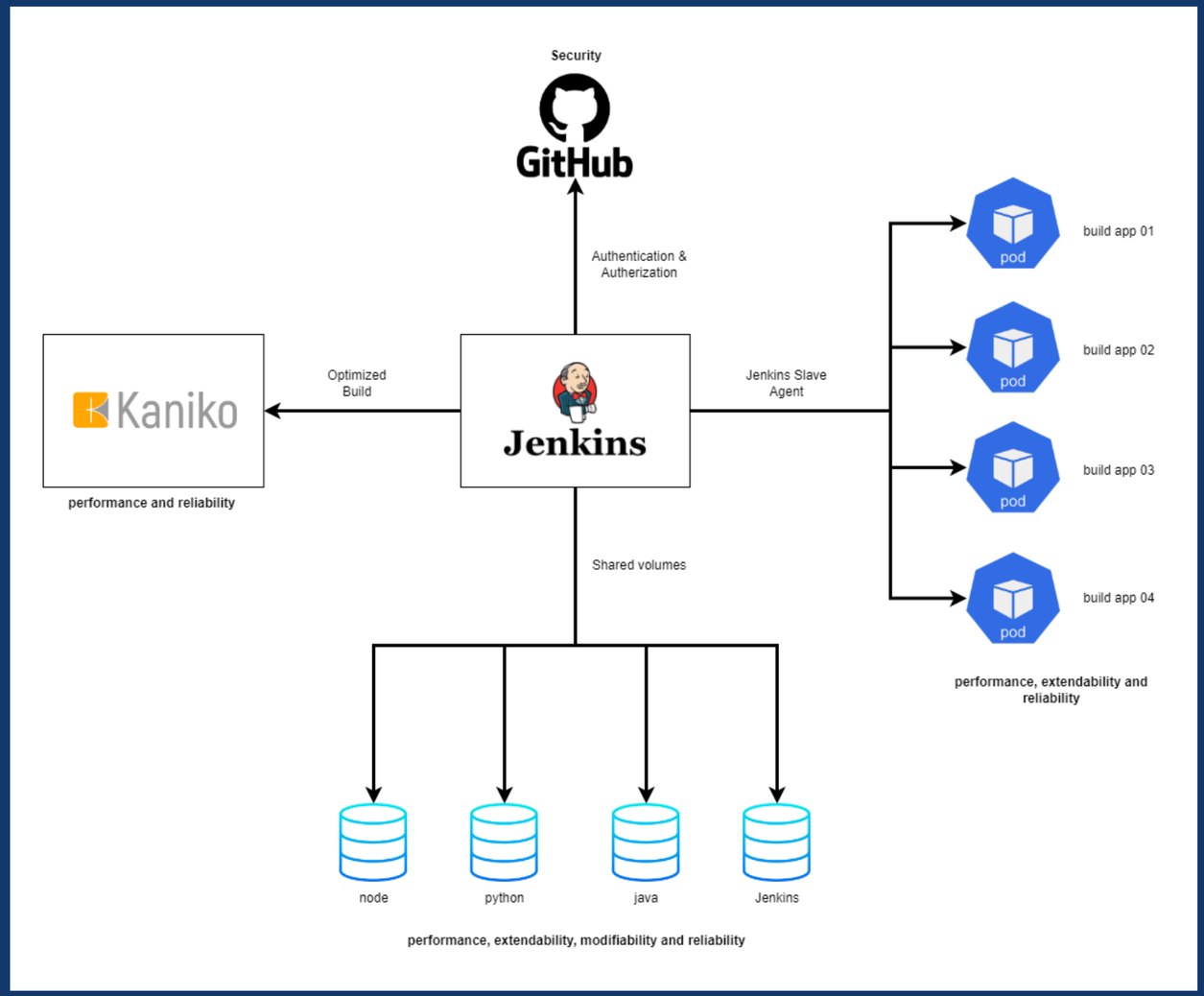
# Current Progress (50%)

- Created a GKE (Google Kubernetes Engine) Cluster as the primary cluster to deploy servers and applications
- Installed and configure Istio on Cluster to expose applications and servers from Kubernetes.
- Exposed the istio-ingress gateway under a public IP address with a TCP load balancer.
- Assigned an SSL certificate with a domain name (releasex.tech) on public IP.
- Deployed primary CI and CD servers (Jenkins Server & Argo CD Server) and configured with necessary configurations for the proposed software release process
- Implemented the proposed CI architecture on Jenkins and finalize the docker image pushing with ACR (Azure Container Registry)
- Started the configuration of CD architecture based on Argo CD and started configuration Jira and Jenkins

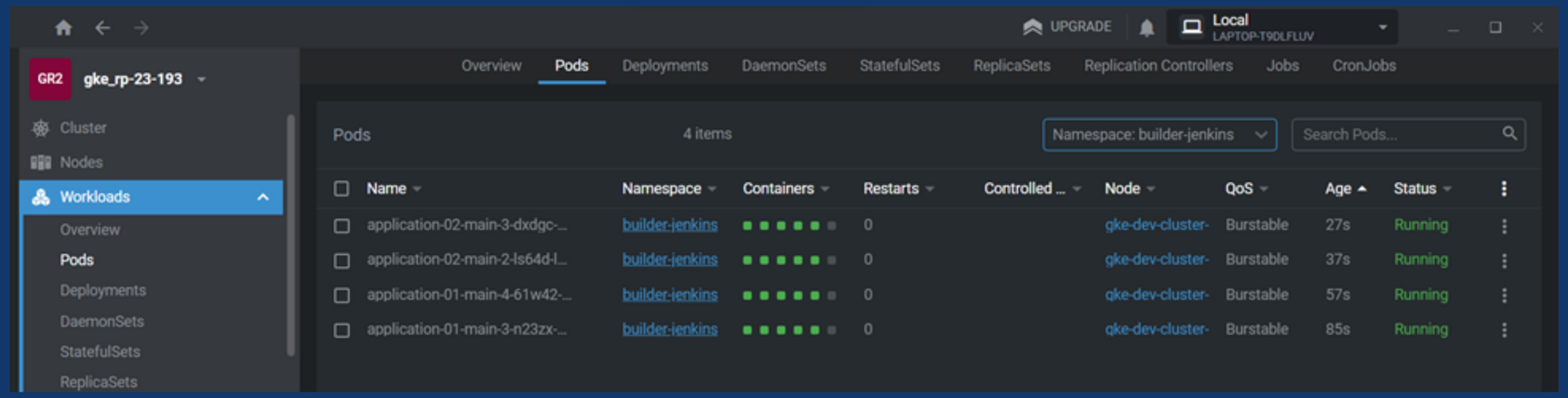
# Proposed Jira Issue Change

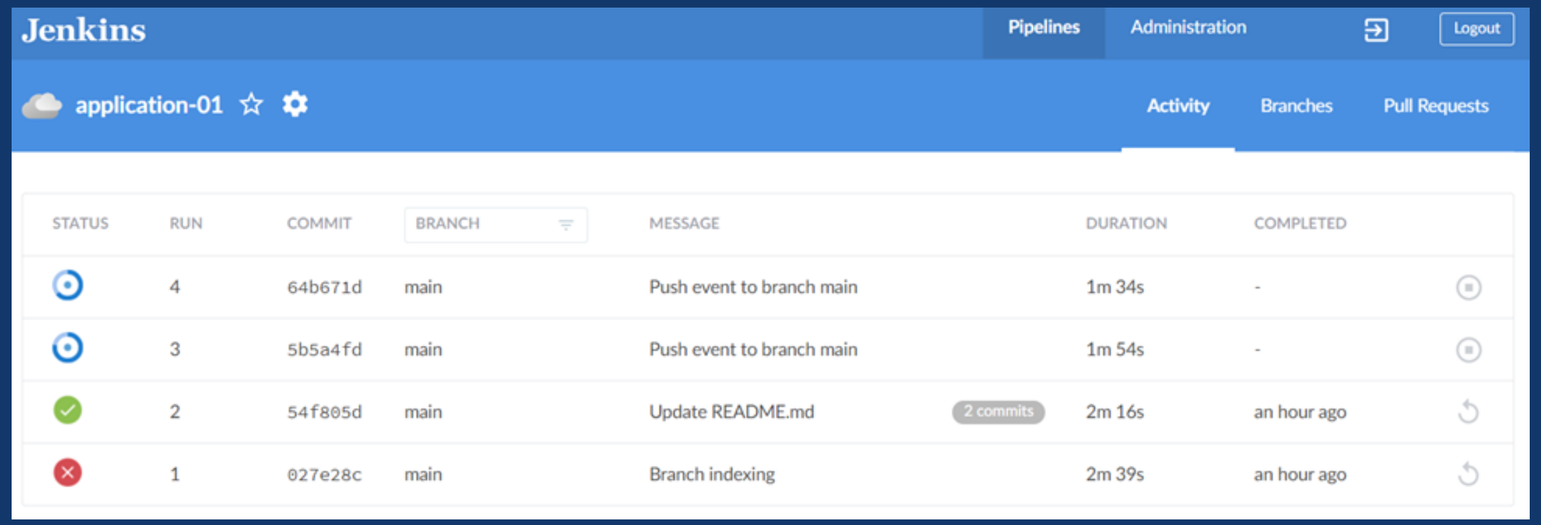


# Optimization of CI Pipeline Architecture

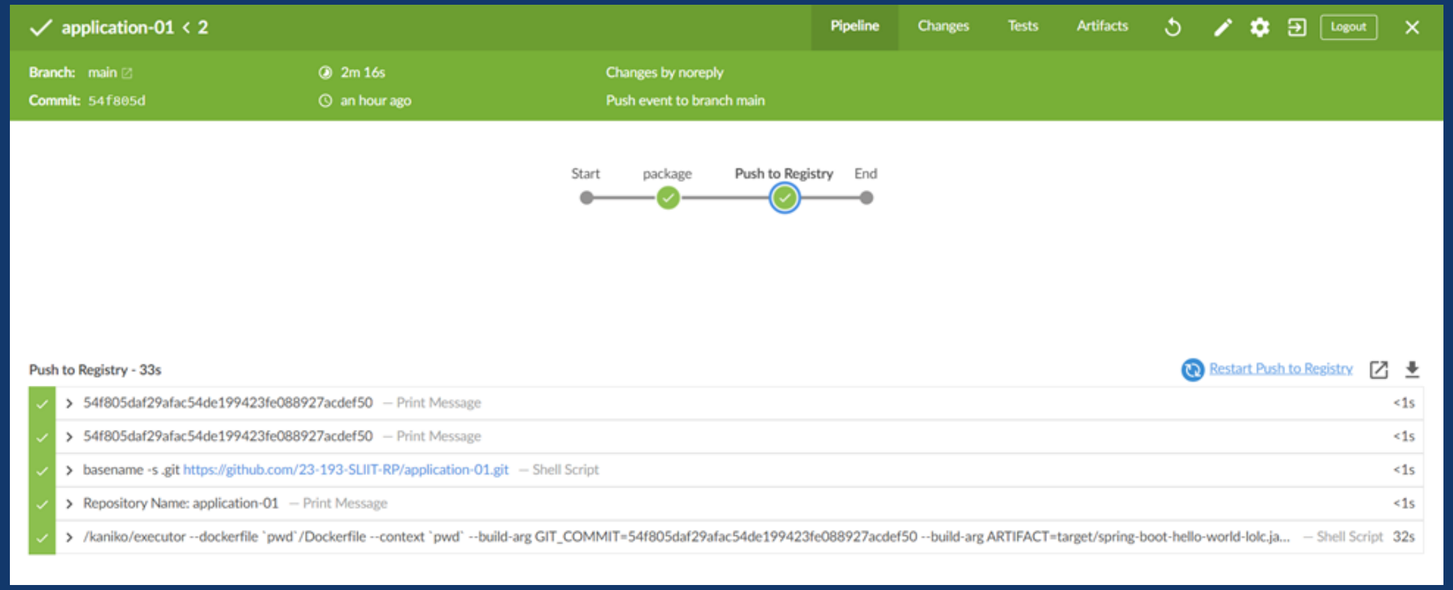


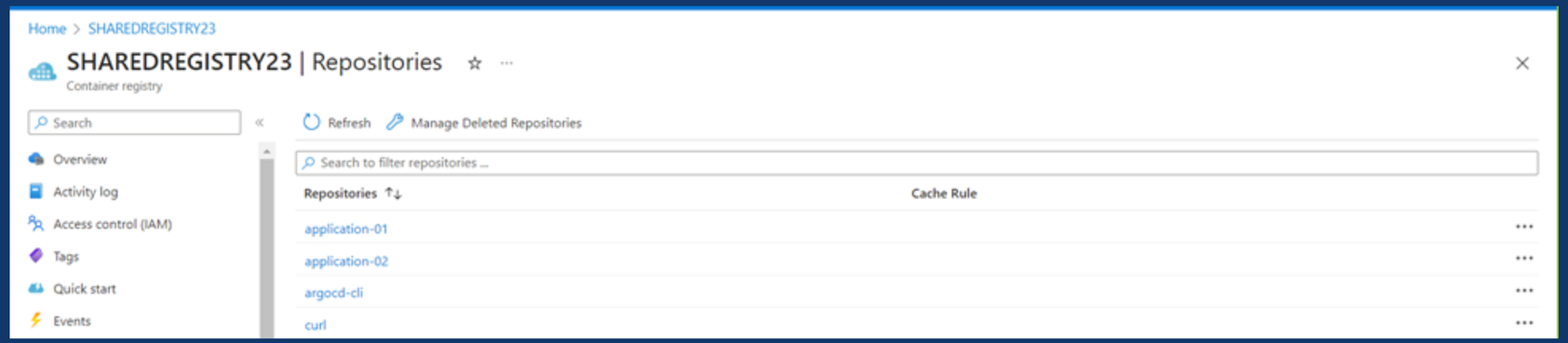
# Proof of Work



# Proof of Work



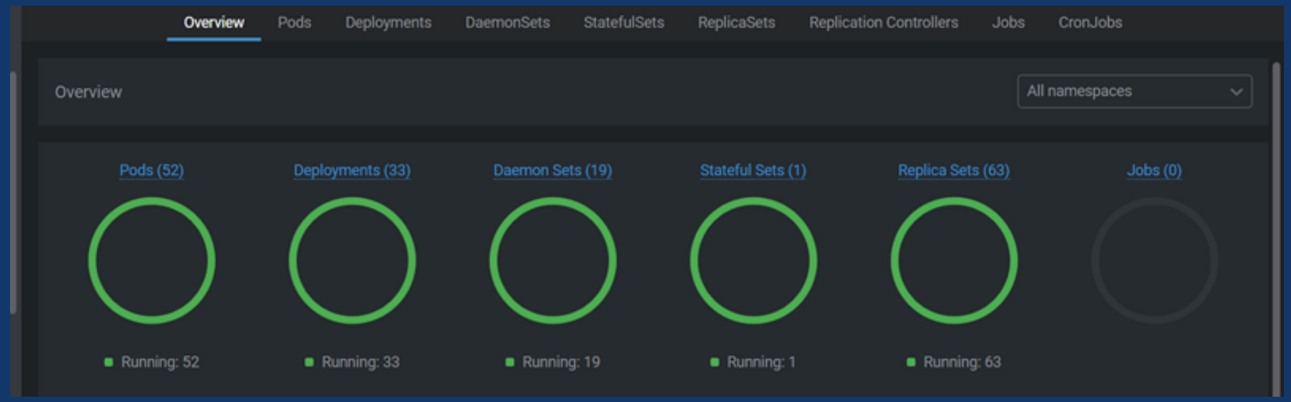



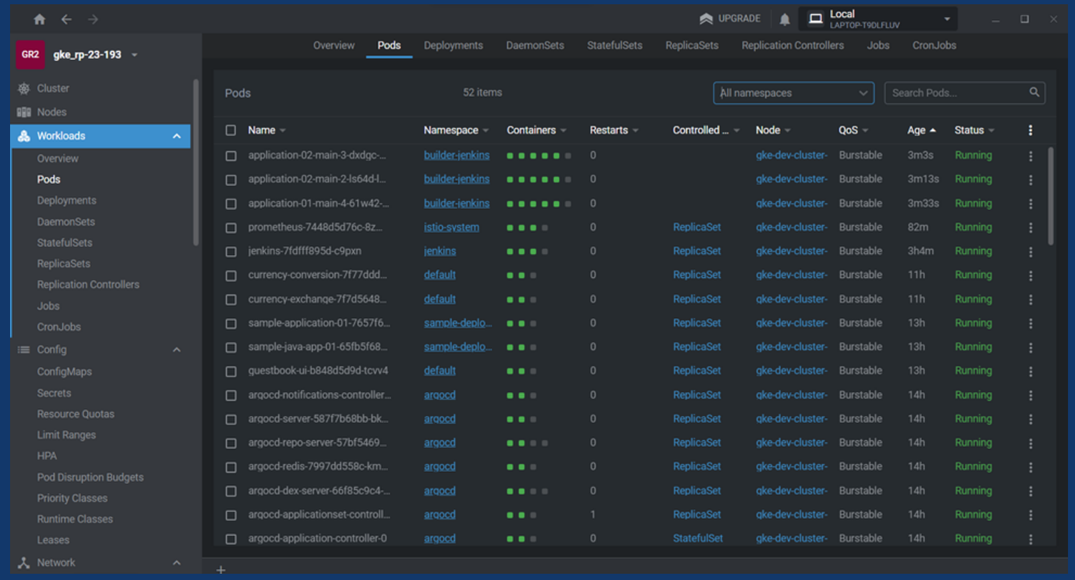
# Proof of Work

Pod application-02-main-3-dxdgc-3k9nr-sg2v4

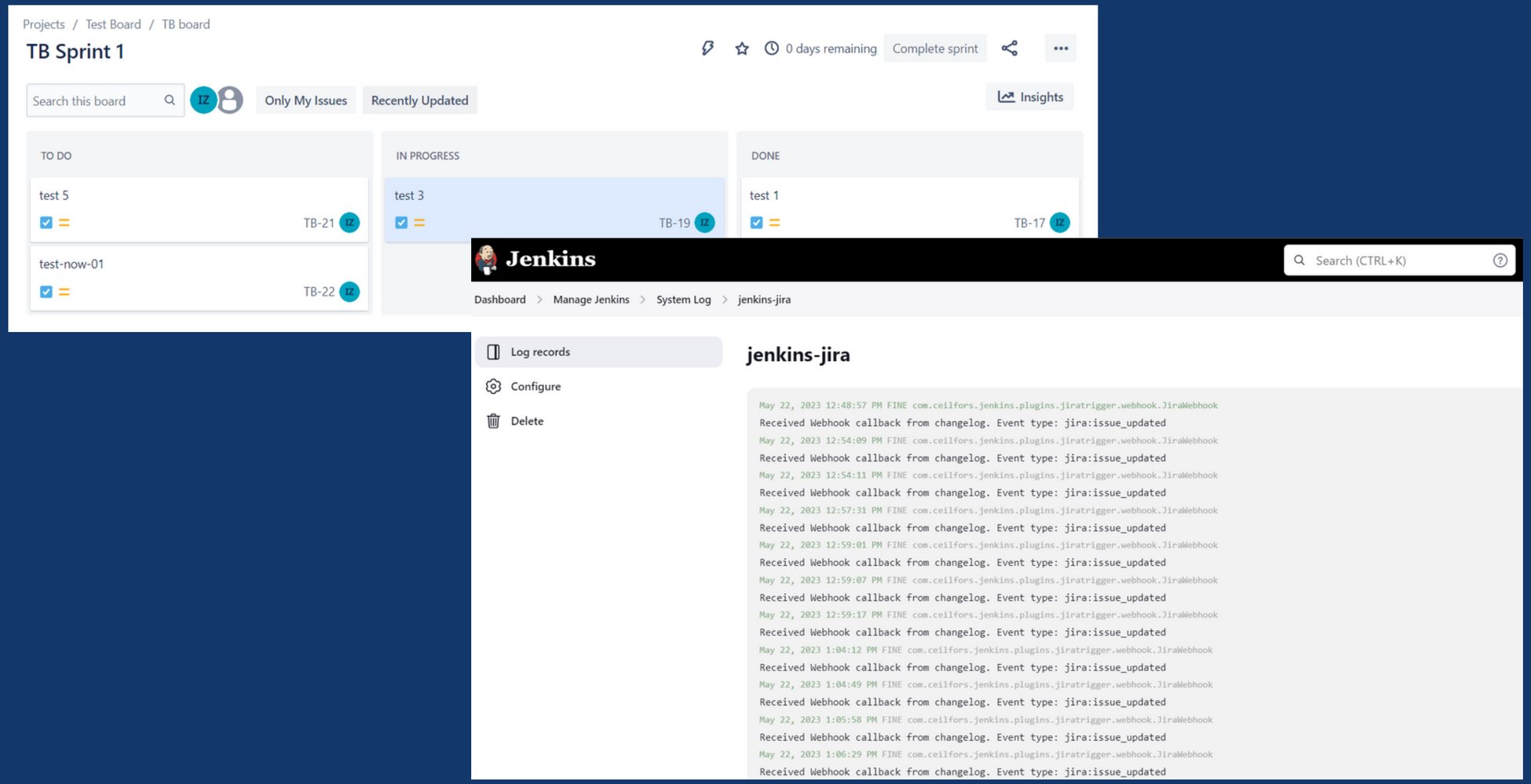
Namespace: builder-jenkins | Pod: application-02-main-3-dxdgc-3k9nr-sg2v4 | Container: jnlp

```
May 22, 2023 8:57:15 PM hudson.remoting.jnlp.Main createEngine
INFO: Setting up agent: application-02-main-3-dxdgc-3k9nr-sg2v4
May 22, 2023 8:57:16 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3107.v665000b_51092
May 22, 2023 8:57:16 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/jenkins/agent/remoting as a remoting work directory
May 22, 2023 8:57:16 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/jenkins/agent/remoting
May 22, 2023 8:57:17 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://jenkins.jenkins.svc.cluster.local:8080/jenkins/]
May 22, 2023 8:57:17 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
May 22, 2023 8:57:17 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting TCP connection tunneling is enabled. Skipping the TCP Agent Listener Port availability check
May 22, 2023 8:57:17 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: jenkins.jenkins.svc.cluster.local
  Agent port:    50000
  Identity:      38:de:91:49:4c:f2:f4:d9:d7:e8:20:f3:e3:24:00:45
May 22, 2023 8:57:17 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
May 22, 2023 8:57:17 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to jenkins.jenkins.svc.cluster.local:50000
May 22, 2023 8:57:17 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
May 22, 2023 8:57:18 PM org.jenkinsci.remoting.protocol.impl.BIONetworkLayer$Reader run
INFO: Waiting for ProtocolStack to start.
May 22, 2023 8:57:18 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 38:de:91:49:4c:f2:f4:d9:d7:e8:20:f3:e3:24:00:45
May 22, 2023 8:57:18 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

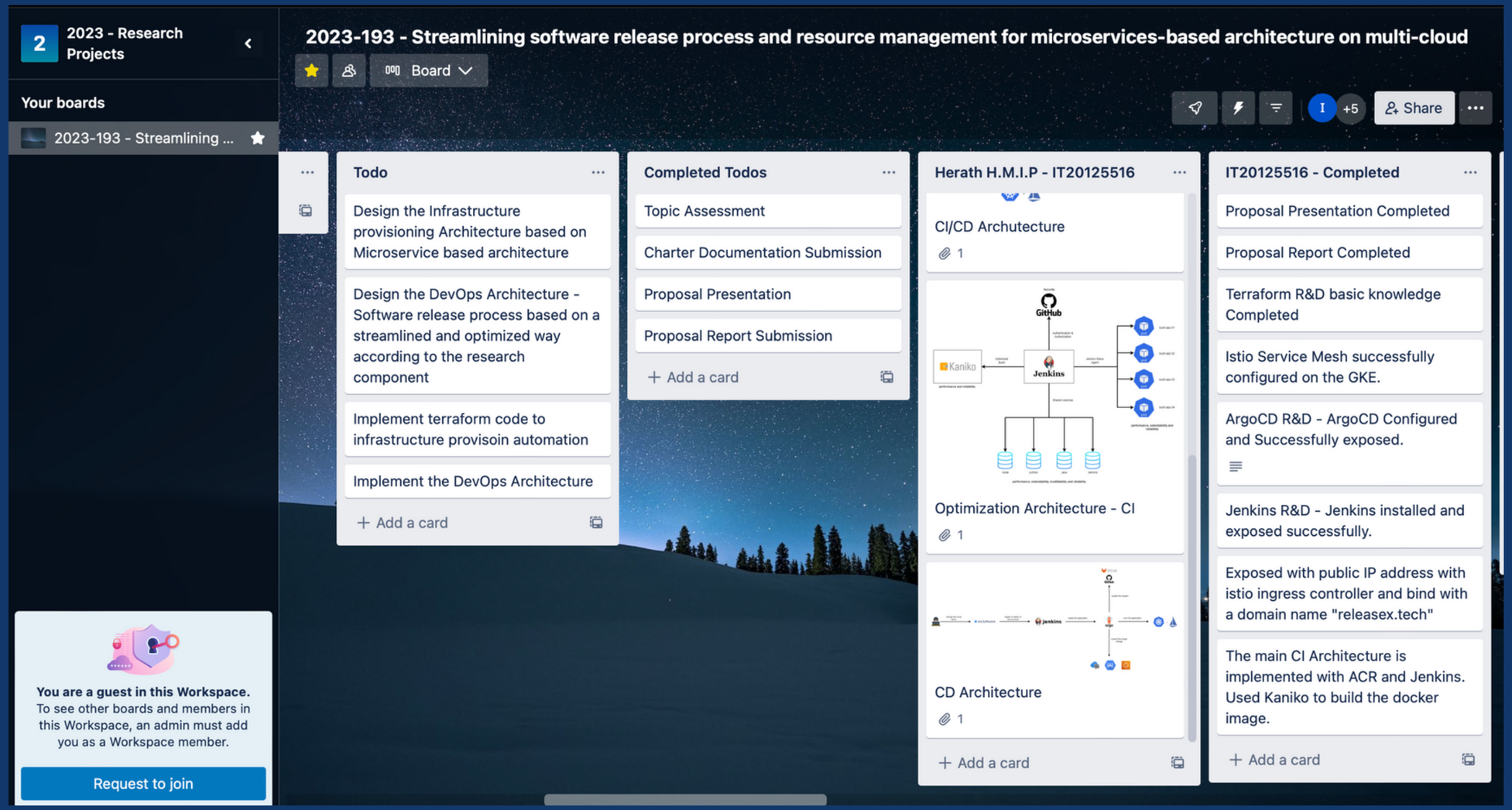


# Proof of Work



# Trello Board



# Next Expected Process (100%)

- Implement the optimizations of CI architecture
- 100% implementation of the CD architecture on Argo CD with Jira Issue updates,
- Improve optimizations on CI/CD process based on previously selected architectural attributes (performance, security, extendibility, reliability, and modifiability)
- Implement deployment in different cluster environments.
- Integrate the full system related to all four release components.

# Functional Requirement

- The primary servers must be available to access anytime by the development team
- The proposed software release process must be easy to use
- The docker image build must be fast and not affect the available Jenkins server
- The available servers and supportive applications must use less memory and storage
- Optimize the time to build and time to push
- Track the development easily with Jira

# Risk Mitigation

- The provided GitLab is not enough to store several source codes as the practical scenario
- The GitLab Organization is created and the source code of each application is purposed to push to the GitHub and the necessary configuration and deployment source code is pushed to the GitLab repository.
- The Argo CD server was working under HTTP(S) and the previous provisioning was based on Load Balancer IP Address. Because of the reason of having only HTTP, the server was not working well.
- The server was exposed under insecure mode with several research on Argo CD deployments.
- Jenkins and Jira could not be connected under the HTTP protocol of Jenkins.
- A domain name and SSL certification are configured on the load balancer.
- The newest version of the Jenkins & Jira configuration plugin is not triggering and continue R&D on another cluster.

# CENTRALIZED CLOUD VISUALIZATION TOOL USING OPEN-SOURCE MONITORING TOOLS.

**Jayawardena R.D.S.H - IT20074968**
**BSc Software Engineering (SLIIT)**

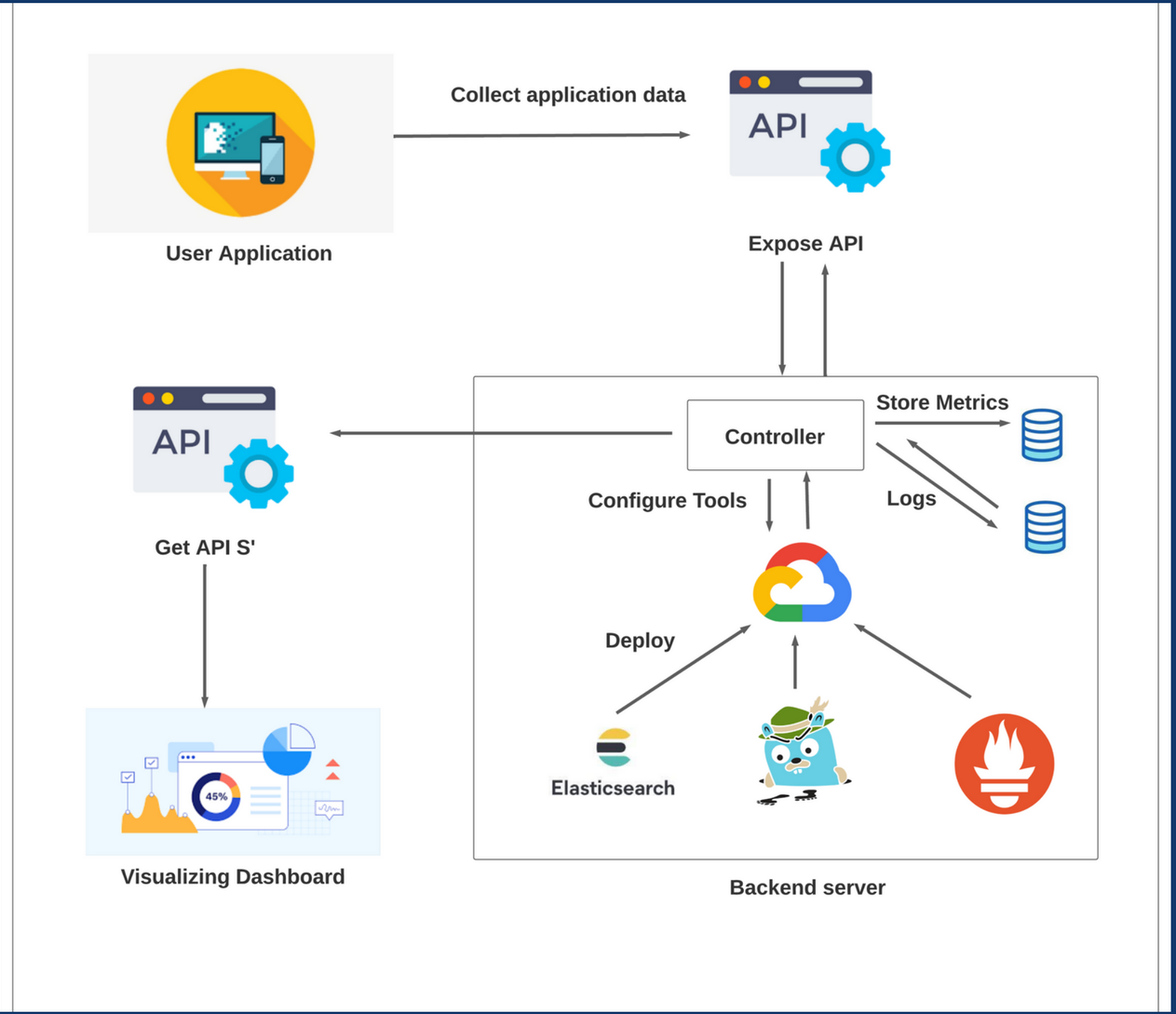
## Target Problem

Users switch between multiple monitoring tools for collecting and visualizing application data, such as metrics, logs, and time series. This leads to inefficiencies and fragmentation in data analysis.

## Proposed Solution

Develop an application that allows users to configure and collect various types of application data in a unified manner. Provide a consolidated visualization platform for comprehensive analysis and decision-making.

# SYSTEM ARCHITECTURE



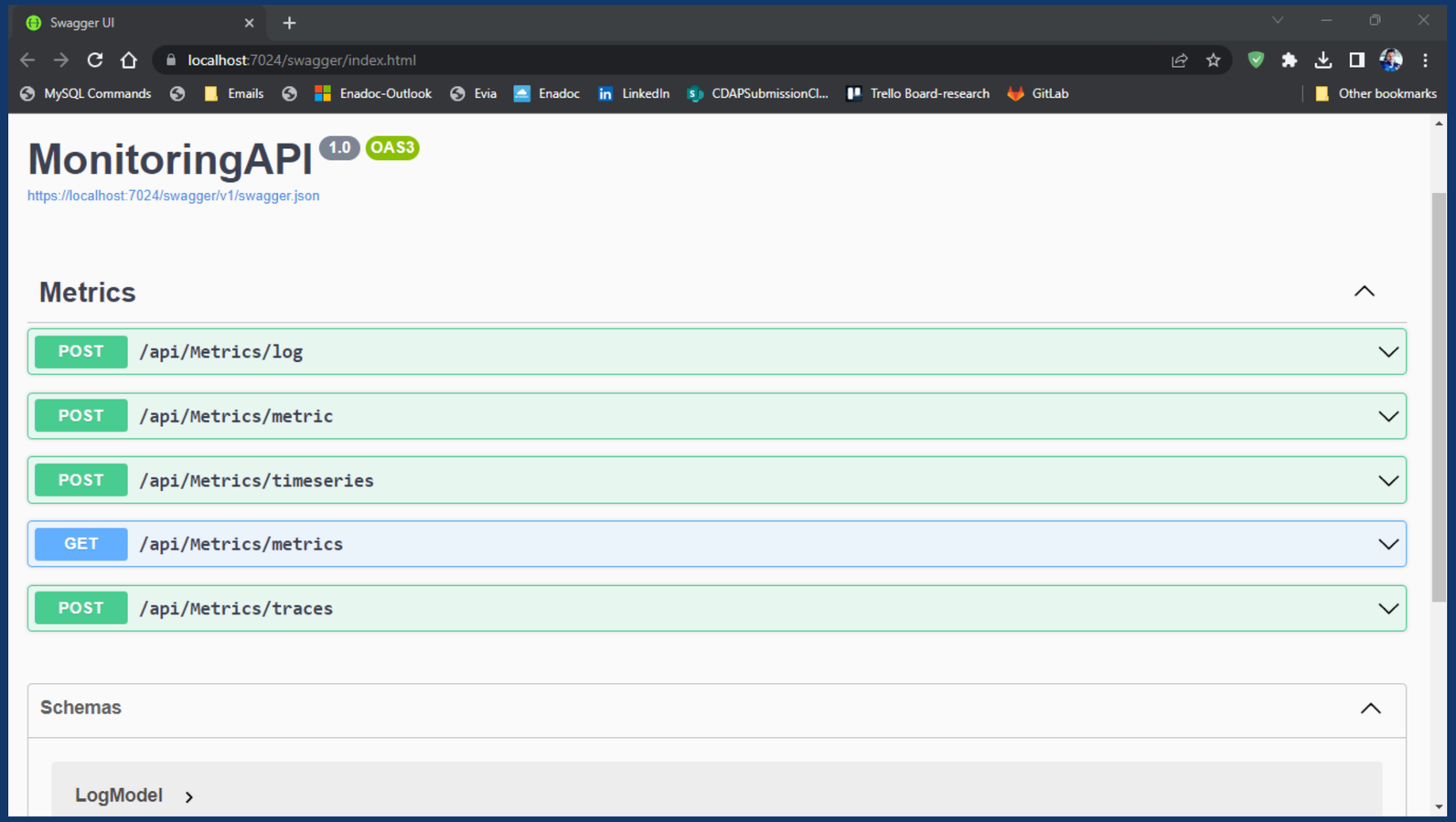
# CURRENT PROGRESS (50%)

- Define model classes for metrics and logs.
- Configure Prometheus, Jeager, Elasticsearch, and Serilogs with the backend.
- Implement API endpoints to handle incoming log and metric data.
- Create an API endpoint to retrieve application metrics and logs pass to the visualization tool.
- Format and return the metrics and logs data in a suitable format.
- Test and refine the functionality of API endpoints and visualization tool integration.

# NEXT EXPECTED PROGRESS ( 100% )

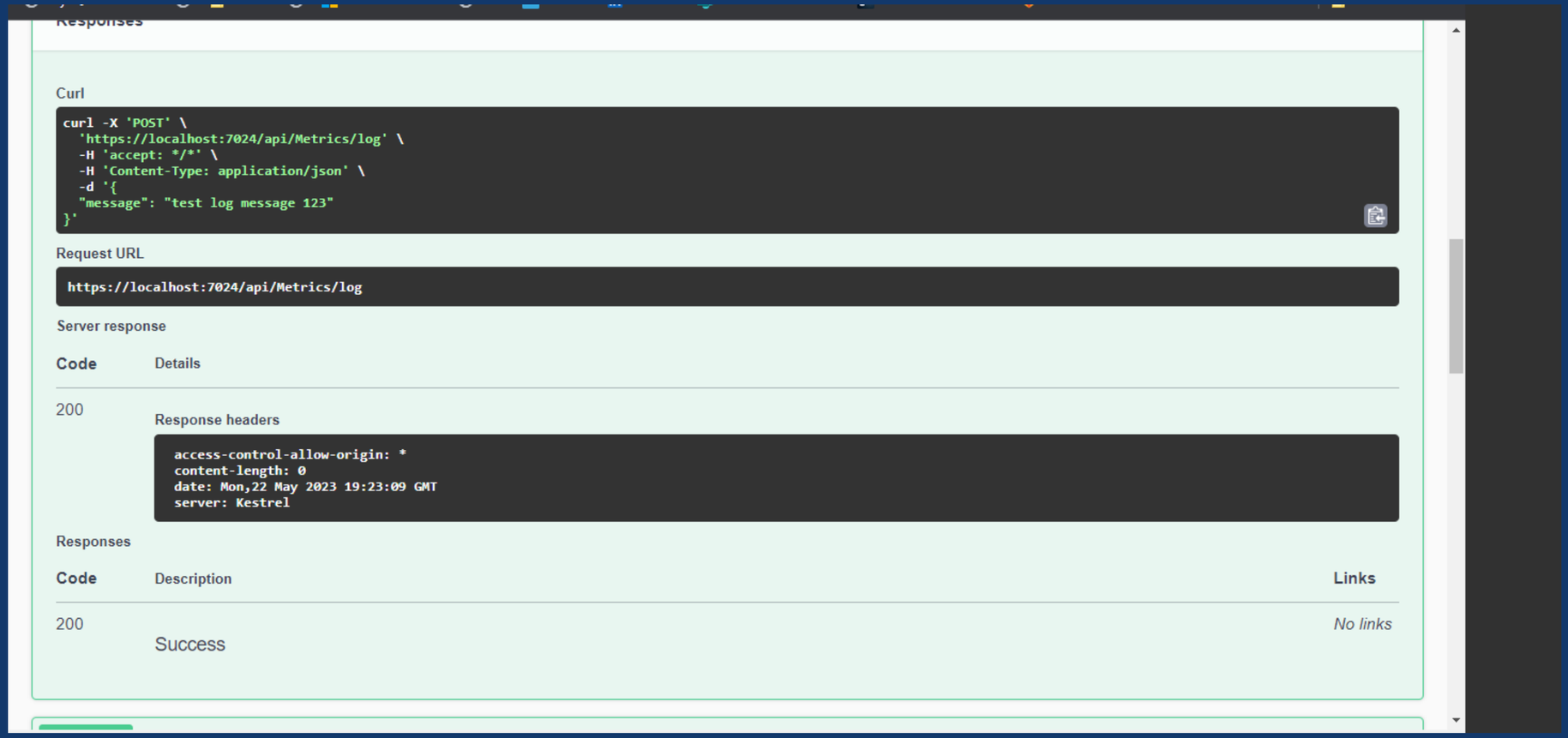
- Implement any additional backend functionalities required for processing and analyzing the collected data.
- Determine the visualization requirements and design a suitable user interface for displaying metrics and other data.
- Select appropriate frontend technologies and frameworks (e.g., JavaScript, React, Angular) to develop the visualization tool.
- Implement the necessary components and modules to fetch data from the backend API.
- Use data visualization libraries (e.g., D3.js, Chart.js) to create interactive and visually appealing charts, graphs, and dashboards.

# PROOF OF WORK




Swagger API dashboard

# PROOF OF WORK




API Response

# PROOF OF WORK

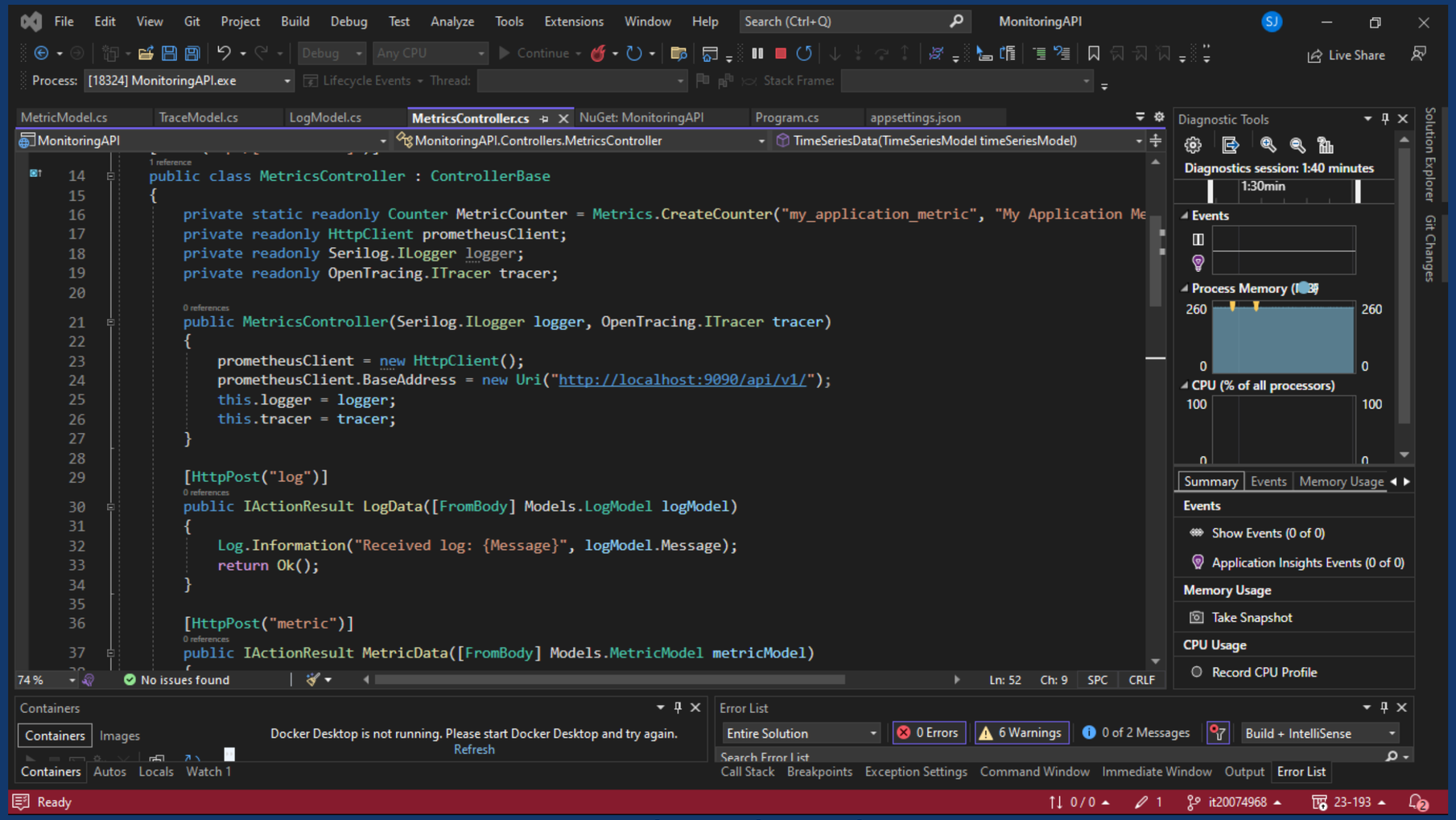
prometheus.yml | logfile.txt

File Edit View

```
2023-05-22 08:15:40.035 +05:30 [INF] Received log: test log message
2023-05-22 08:15:40.082 +05:30 [INF] HTTP POST /api/Metrics/log responded 200 in 194.7850 ms
2023-05-22 08:16:04.505 +05:30 [INF] Received log: test log message 888
2023-05-22 08:16:04.507 +05:30 [INF] HTTP POST /api/Metrics/log responded 200 in 3.7816 ms
2023-05-22 14:08:06.183 +05:30 [INF] Received log: log rrrrrrrr
2023-05-22 14:08:06.194 +05:30 [INF] HTTP POST /api/Metrics/log responded 200 in 20.5468 ms
2023-05-22 18:22:43.049 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 341.0455 ms
2023-05-22 18:27:45.588 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.6744 ms
2023-05-22 18:33:21.051 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 16.5774 ms
2023-05-22 18:33:25.014 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 0.7007 ms
2023-05-22 18:37:07.555 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 1.0056 ms
2023-05-22 19:01:15.528 +05:30 [ERR] HTTP POST /api/Metrics/metric responded 500 in 50.0337 ms
2023-05-22 19:30:19.783 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 179.8124 ms
2023-05-22 19:30:29.045 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.6738 ms
2023-05-22 19:36:17.742 +05:30 [INF] HTTP POST /api/Metrics/metric responded 200 in 7.1433 ms
2023-05-22 19:39:12.538 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.1402 ms
2023-05-22 19:39:14.342 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0045 ms
2023-05-22 19:39:14.917 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0100 ms
2023-05-22 19:39:15.118 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0097 ms
2023-05-22 19:39:15.298 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0126 ms
2023-05-22 19:39:15.670 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0074 ms
2023-05-22 19:39:15.867 +05:30 [INF] HTTP GET /api/Metrics/metric responded 405 in 0.0103 ms
2023-05-22 20:14:31.568 +05:30 [INF] HTTP GET /api/Metrics/metrics responded 200 in 30.3192 ms
2023-05-22 21:11:50.526 +05:30 [INF] HTTP GET /api/Metrics/metrics responded 200 in 222.9825 ms
2023-05-22 22:41:24.124 +05:30 [ERR] HTTP GET /api/Metrics/metrics responded 500 in 46.6823 ms
2023-05-22 22:50:48.597 +05:30 [INF] HTTP GET /favicon.ico responded 404 in 0.1661 ms
2023-05-22 22:57:28.176 +05:30 [INF] HTTP GET /api/Metrics/metrics responded 200 in 231.0656 ms
2023-05-22 22:58:12.189 +05:30 [INF] HTTP GET /api/Metrics/metrics responded 200 in 208.1392 ms
```

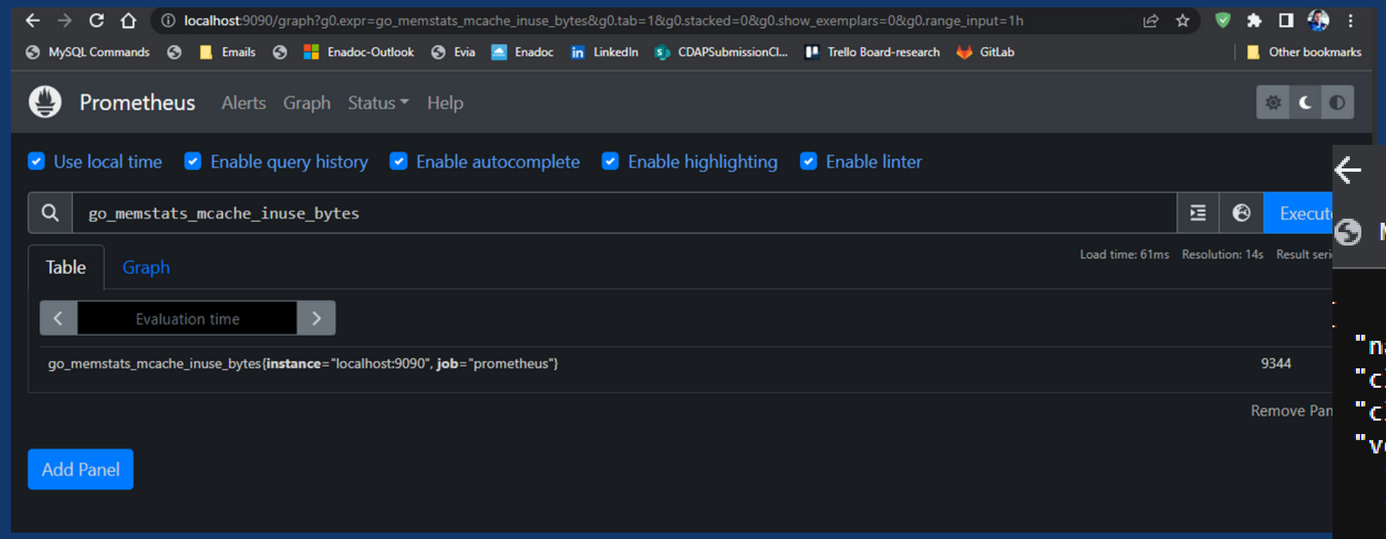
Log Data collection

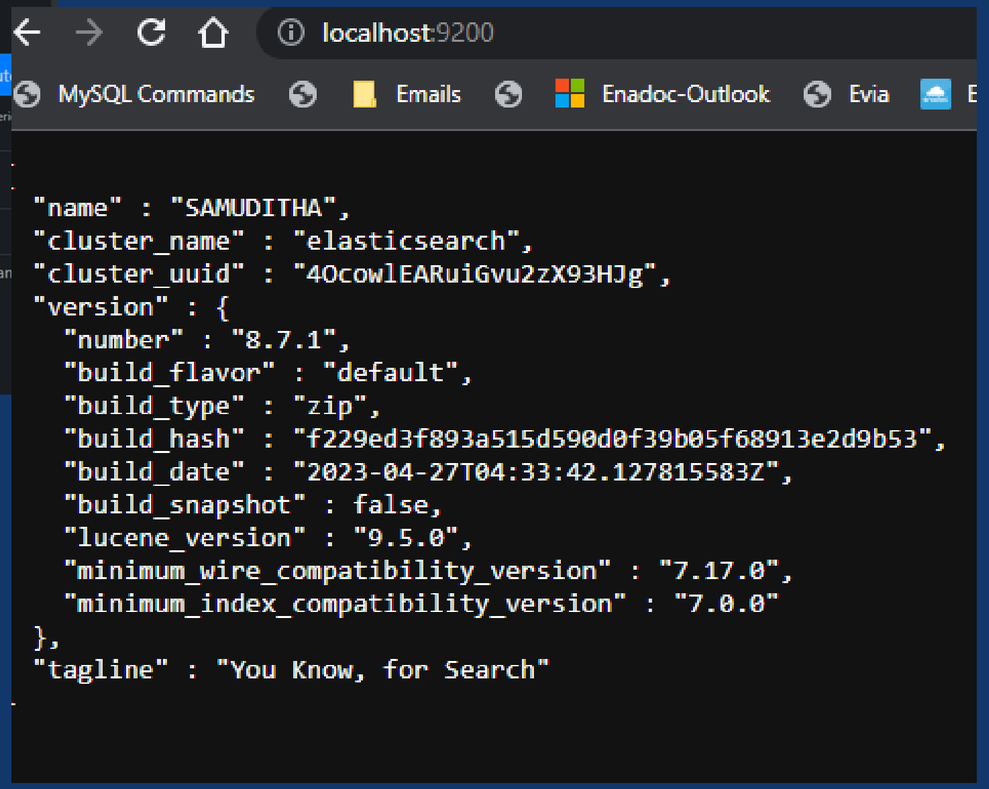
# PROOF OF WORK




Backend code

# PROOF OF WORK




Configuration of Prometheus/Elasticsearch

# FUNCTIONAL /NON-FUNCTIONAL REQUIREMENTS

- Define API endpoints for handling metrics, logs, and data.
- Store data in a suitable data store.
- Validate incoming data for integrity and format.
- Implement security measures like authentication and authorization.
- Handle errors and exceptions gracefully.
- Implement robust security measures for data protection.
- Maintainable codebase for easy updates and enhancements.
- Scalability options for handling increased loads.

# Risk Mitigation

- I faced the problem of how to configure the user's application with a centralized tool and how to collect metrics from the user's application.
- To resolve this, I needed to configure all monitoring tools with a centralized application and expose an API to collect data from the user's application.
- The second problem was how to store the collected data. During my research, I discovered various databases that support these monitoring tools.

# Machine Learning workflow for Optimization of Kubernetes Deployment Performance.

**Fadhil M.R.A. - IT20784720**
**BSc Software Engineering (SLIIT)**

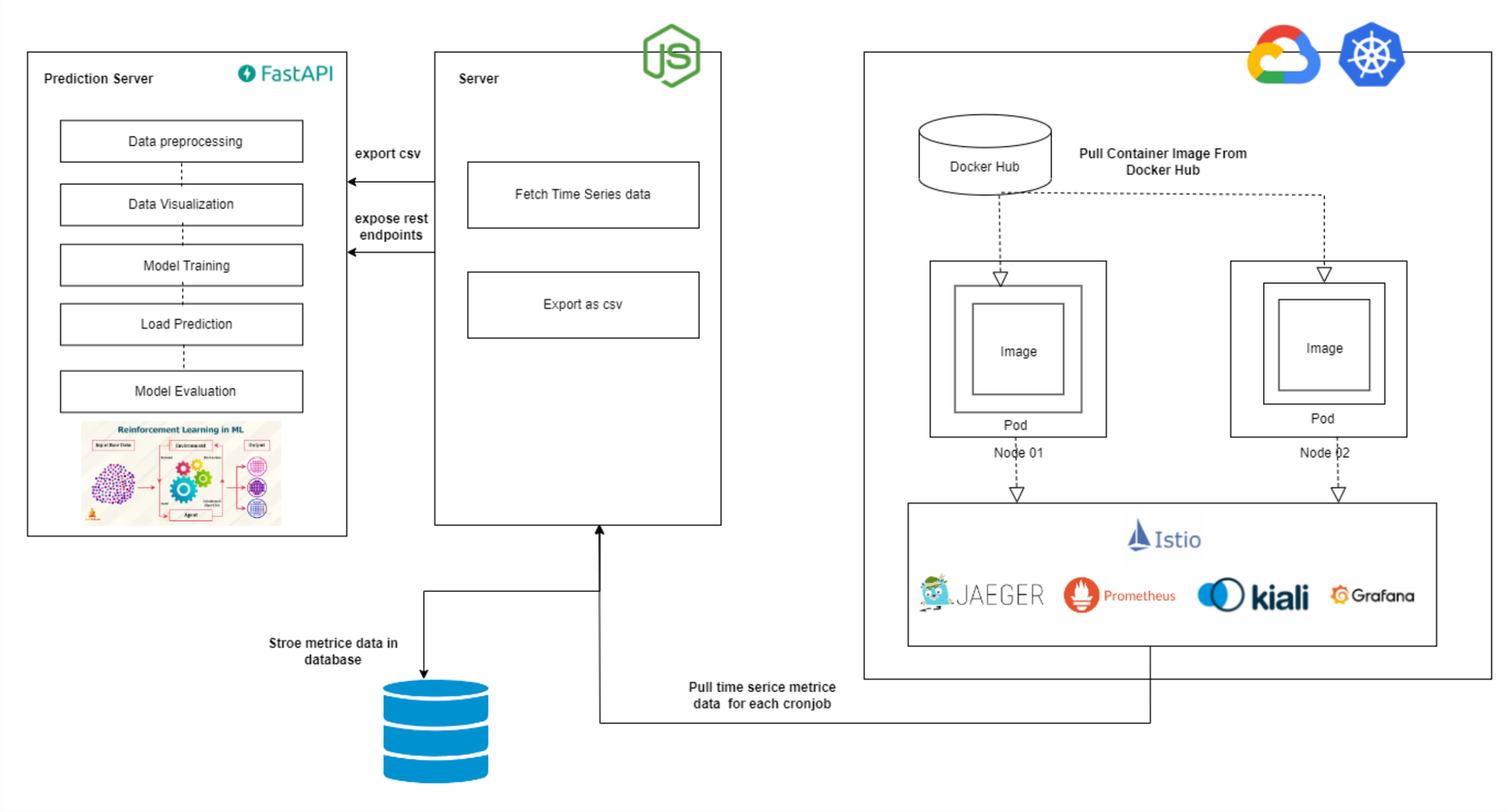
# Target Problem

there are few studies that focus on optimizing the resource workload of distributed applications using a machine learning model and both vertical and horizontal auto-scaling on Kubernetes, by gethering the  pod level time series  metrices using a Restapi and by creating an algorithm to scale the kubernets cluster according to the metrices workload.

# Proposed Solution

the proposed solution is to set up the Istio service mesh in Google Kubernetes Engine, and integrate prometheus , Graphana, Kiali , Jaeger and by deplying a sample microservice application on google kubernets engine and gether the time series metrice by creating a cron job , by using the reinforcement learning algorithm an optimized and fast prediction mechanism can be provide for scale the cluster.

# SYSTEM ARCHITECTURE



# Current Progress (50%)

- Created a GKE (Google Kubernetes Engine) Cluster as the primary cluster to deploy servers and applications
- Installed and configure Istio on Cluster to expose applications and servers from Kubernetes and integrated with prometheus , Graphana, Kiali and Jaeger.
- Exposed the istio-ingress gateway and prometheus under a public ip address.
- deployed a sample microservice application with Gateway and VirtualService yaml files.
- developed a nodejs application to fetch the time series matrices from prometheus and store in a database.
- generate csv files to different time series matrices.
- export the csv file to jupyter notebook and make some preprocessing and data visualizing functions.

# Next Expected Process (100%)

- select the reinforcement learning technique  to predict time series matrices
- Implementing the cpu load , memory usage , centrality prediction algorithm using reinforcement learning.
- create a cronjob to fetch time series metrices in a given time frame.
- create a mechanism to fetch time series metrices whenever the cronjob was executed.
- by using the vertical pod autoscaling , predict the workload that is required in  a service.
- deploy the reinforcement learning prediction model.

# Functional Requirement

- The metrices server need to be available.
- The metrice must be able to fetch metrices data according to the cronjob with less time consuming.
- The prediction model need  to be accurate.
- The prediction model must be able to dynamicaly predict the cpu workload.

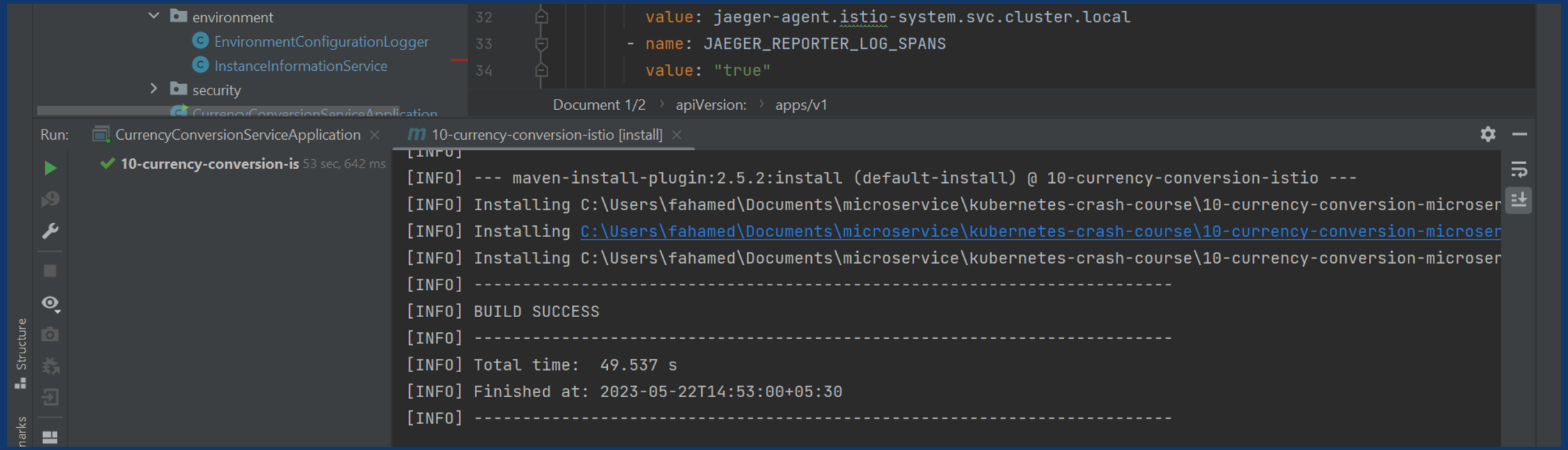
# Proof of Work

```
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
fadhilahamed07@cloudshell:~ (rp-23-193)$ gcloud container clusters get-credentials dev-cluster-01 --zone us-central1-a --project rp-23-193
Fetching cluster endpoint and auth data.
kubeconfig entry generated for dev-cluster-01.
fadhilahamed07@cloudshell:~ (rp-23-193)$ ls
get_helm.sh  istio-1.13.3  istio-1.17.2  istio-1.2.2  istio-series  istio.yaml  README-cloudshell.txt
fadhilahamed07@cloudshell:~ (rp-23-193)$ cd istio-1.17.2/
fadhilahamed07@cloudshell:~/istio-1.17.2 (rp-23-193)$ ls
bin  istio.yaml  LICENSE  manifests  manifest.yaml  README.md  samples  tools
fadhilahamed07@cloudshell:~/istio-1.17.2 (rp-23-193)$ kubectl get gateway
NAME               AGE
bookinfo-gateway   28m
main-gateway       11d
fadhilahamed07@cloudshell:~/istio-1.17.2 (rp-23-193)$ kubectl get svc istio-ingressgateway -n istio-system
NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)                                                                      AGE
istio-ingressgateway   LoadBalancer   10.96.4.4     34.68.11.25    15021:30106/TCP,80:30309/TCP,443:32645/TCP,31400:30415/TCP,15443:30416/TCP   12d
fadhilahamed07@cloudshell:~/istio-1.17.2 (rp-23-193)$ export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.
alancer.ingress[0].ip}')
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].port}')
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].port}')
fadhilahamed07@cloudshell:~/istio-1.17.2 (rp-23-193)$ echo $SECURE_INGRESS_PORT
```

```
configmap/istio-services-grafana-dashboards created
deployment.apps/jaeger created
service/tracing created
service/zipkin created
service/jaeger-collector created
serviceaccount/kiali created
configmap/kiali created
clusterrole.rbac.authorization.k8s.io/kiali-viewer created
clusterrole.rbac.authorization.k8s.io/kiali created
clusterrolebinding.rbac.authorization.k8s.io/kiali created
role.rbac.authorization.k8s.io/kiali-controlplane created
rolebinding.rbac.authorization.k8s.io/kiali-controlplane created
```

# Proof of Work

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ☐ | istio-egressgateway | OK | Cluster IP | 10.96.4.154 | 1/1 | istio-system | dev-clu... |
| ☐ | istio-ingressgateway | OK | External load balancer | 34.68.11.25:15021 | 1/1 | istio-system | dev-clu... |
| ☐ | istiod | OK | Cluster IP | 10.96.6.70 | 1/1 | istio-system | dev-clu... |

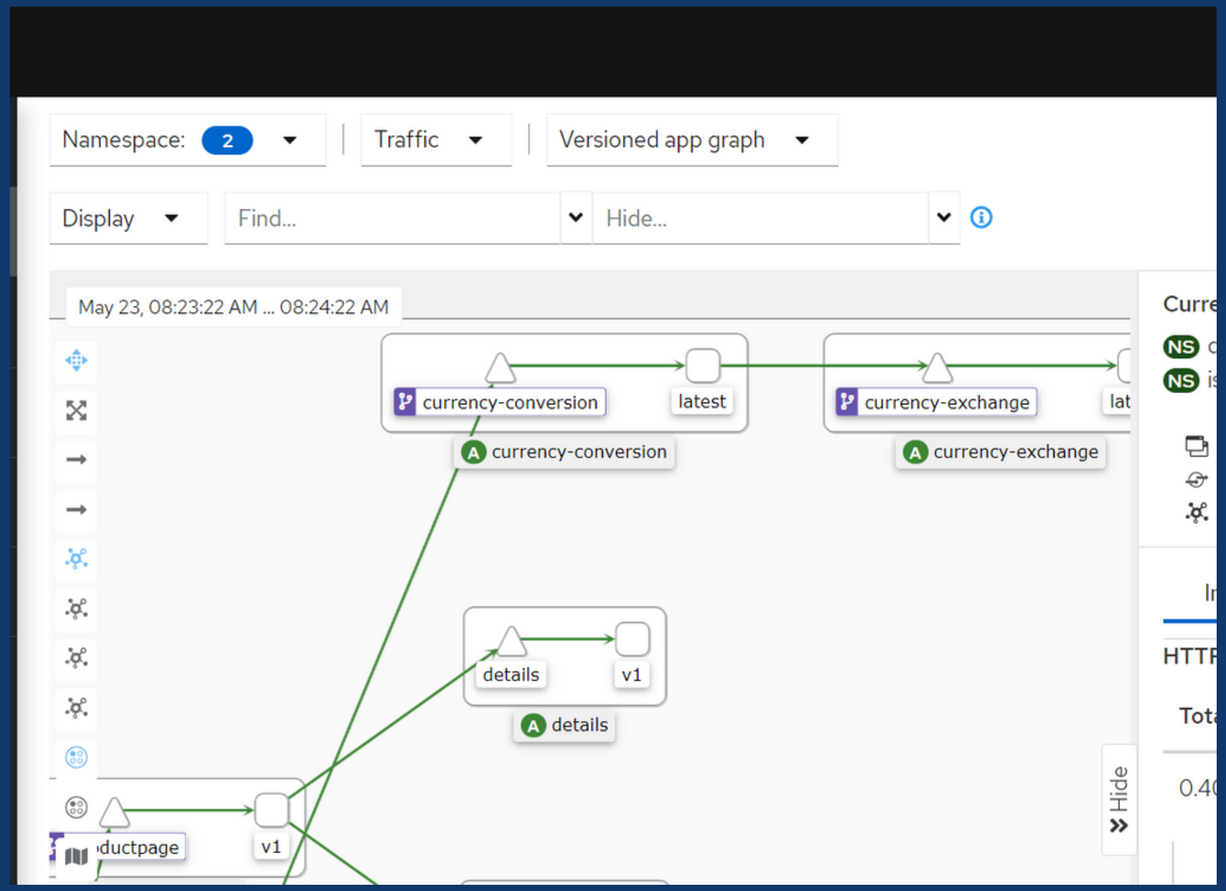


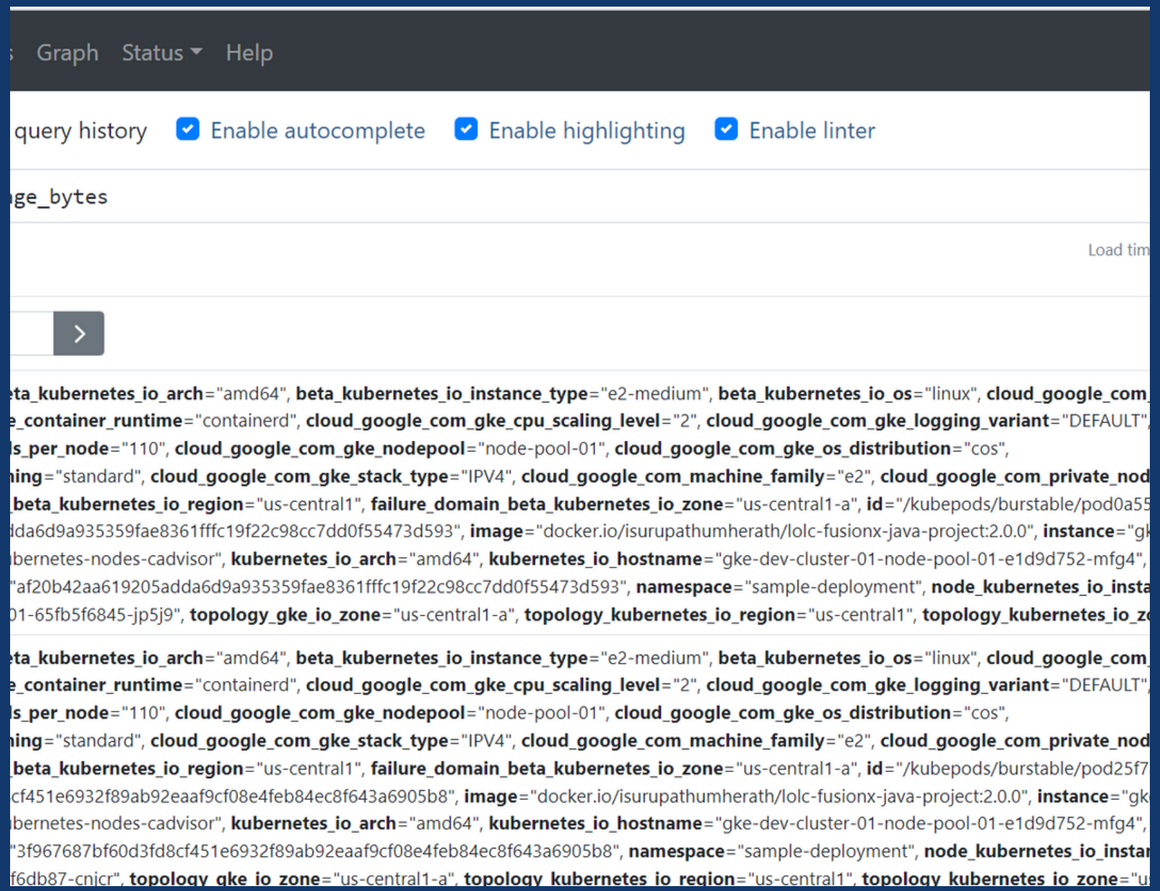
# Proof of Work

```
▼ 0:
  ▼ metric:
      __name__:                                  "container_memory_usage_bytes"
      beta_kubernetes_io_arch:                   "amd64"
      beta_kubernetes_io_instance_type:          "e2-medium"
      beta_kubernetes_io_os:                     "linux"
      cloud_google_com_gke_boot_disk:            "pd-balanced"
      cloud_google_com_gke_container_runtime:    "containerd"
      cloud_google_com_gke_cpu_scaling_level:    "2"
      cloud_google_com_gke_logging_variant:      "DEFAULT"
      cloud_google_com_gke_max_pods_per_node:    "110"
      cloud_google_com_gke_nodepool:             "node-pool-01"
      cloud_google_com_gke_os_distribution:      "cos"
      cloud_google_com_gke_provisioning:         "standard"
      cloud_google_com_gke_stack_type:           "IPV4"
      cloud_google_com_machine_family:           "e2"
      cloud_google_com_private_node:             "false"
      container:                                 "app"
      failure_domain_beta_kubernetes_io_region:  "us-central1"
      failure_domain_beta_kubernetes_io_zone:    "us-central1-a"
    ▶ id:                                        "/kubepods/burstable/pod0…9f22c98cc7dd0f55473d593"
    ▶ image:                                     "docker.io/isurupathumher…ionx-java-project:2.0.0"
      instance:                                  "gke-dev-cluster-01-node-pool-01-e1d9d752-mfg4"
      job:                                       "kubernetes-nodes-cadvisor"
      kubernetes_io_arch:                        "amd64"
      kubernetes_io_hostname:                    "gke-dev-cluster-01-node-pool-01-e1d9d752-mfg4"
```

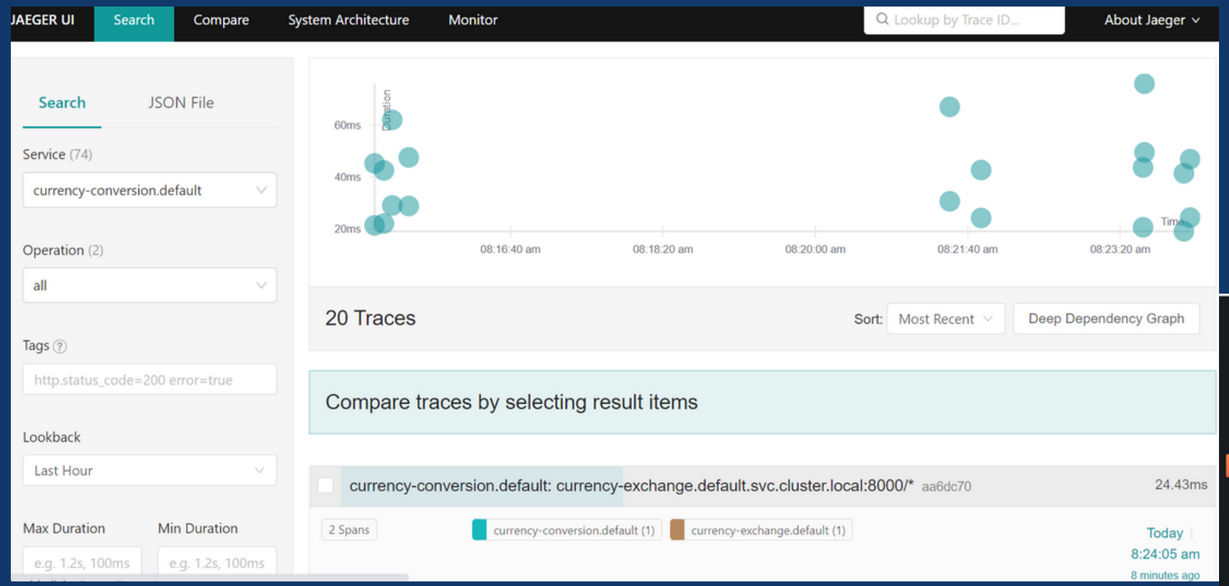
| ubernete | kubernete | kubernete | name | namespac | node_kub | pod | topology_ | topolog | topolog |
|---|---|---|---|---|---|---|---|---|---|
| md64 | gke-dev-c | linux | 13a877173 | sample-de | e2-mediu | sample-ja | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | f64085414 | argocd | e2-mediu | argocd-se | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | ea5fa5545 | argocd | e2-mediu | argocd-re | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | d37ad61ac | sample-de | e2-mediu | sample-ap | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 12bd0d77 | default | e2-mediu | reviews-v | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 7ccd92e9b | istio-syste | e2-mediu | istio-ingre | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | eaca45fd7 | jenkins | e2-mediu | jenkins-7f | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 2f2ac071d | argocd | e2-mediu | argocd-ap | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 3bca1b06( | default | e2-mediu | reviews-v | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 68253a52c | default | e2-mediu | ratings-v1 | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | b5d6302ac | default | e2-mediu | currency-( | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 1b32b738l | default | e2-mediu | guestbool | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 1a817da54 | argocd | e2-mediu | argocd-no | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 2264eac1c | default | e2-mediu | reviews-v | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | b92ec05ac | default | e2-mediu | currency-( | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | b627a61f4 | istio-syste | e2-mediu | promethe | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 7895a110f | argocd | e2-mediu | argocd-ap | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 4365d1b47 | istio-syste | e2-mediu | istio-egre | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 68d614fb1 | argocd | e2-mediu | argocd-re | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | a538a840l | default | e2-mediu | productpa | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 6c332c2a9 | argocd | e2-mediu | argocd-de | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 6474e30de | default | e2-mediu | details-v1 | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | c7f9941b4 | jenkins | e2-mediu | jenkins-7f | us-central | us-centı | us-cent |
| md64 | gke-dev-c | linux | 5bd08068l | kube-syst | e2-mediu | metrics-se | us-central | us-centı | us-cent |

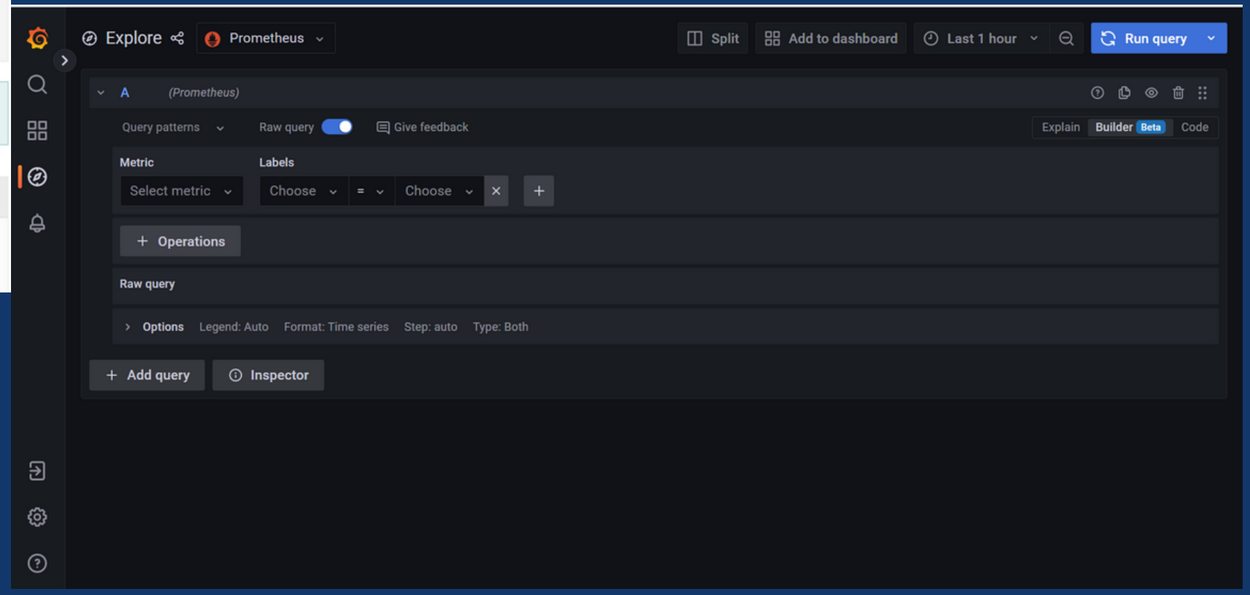
# Proof of Work



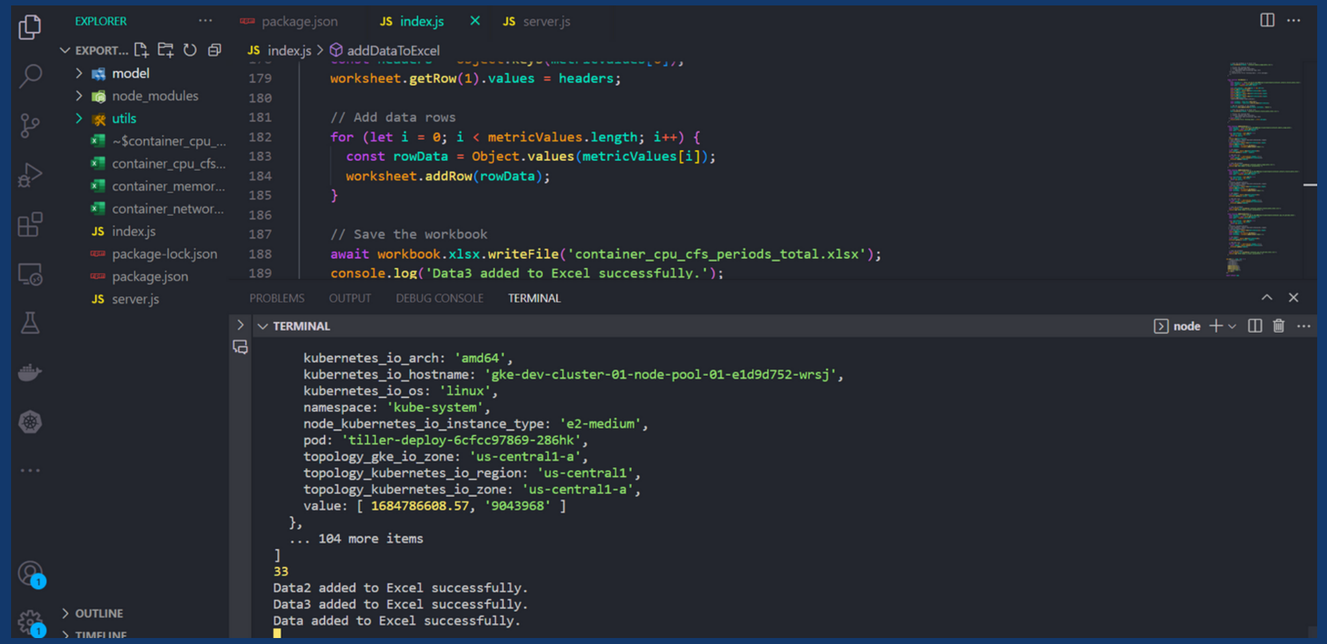



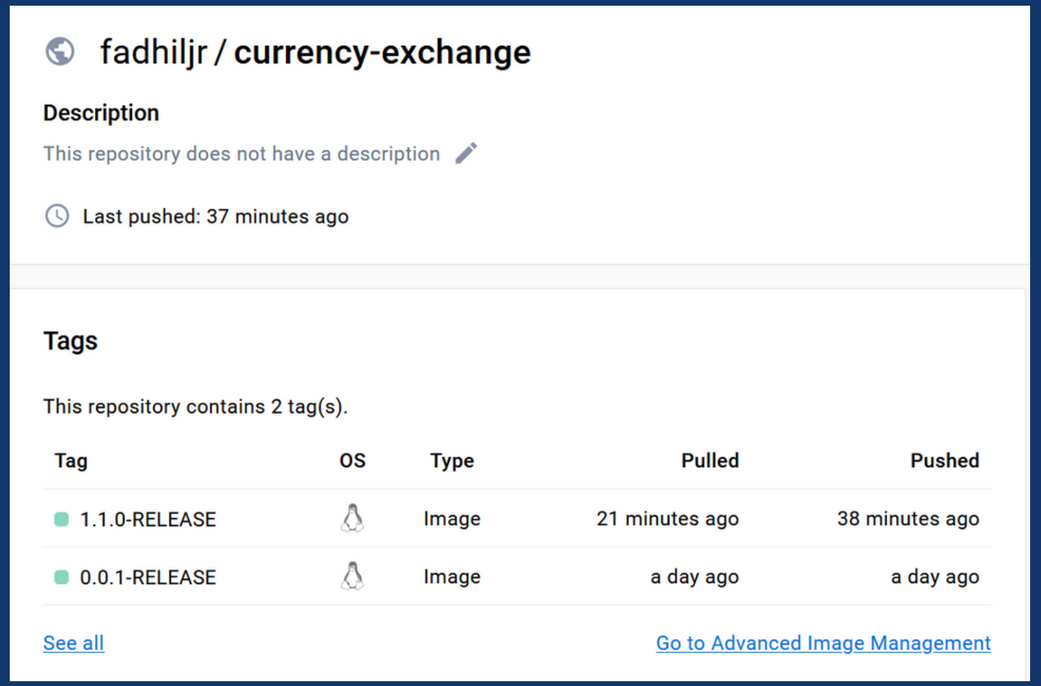
# Proof of Work



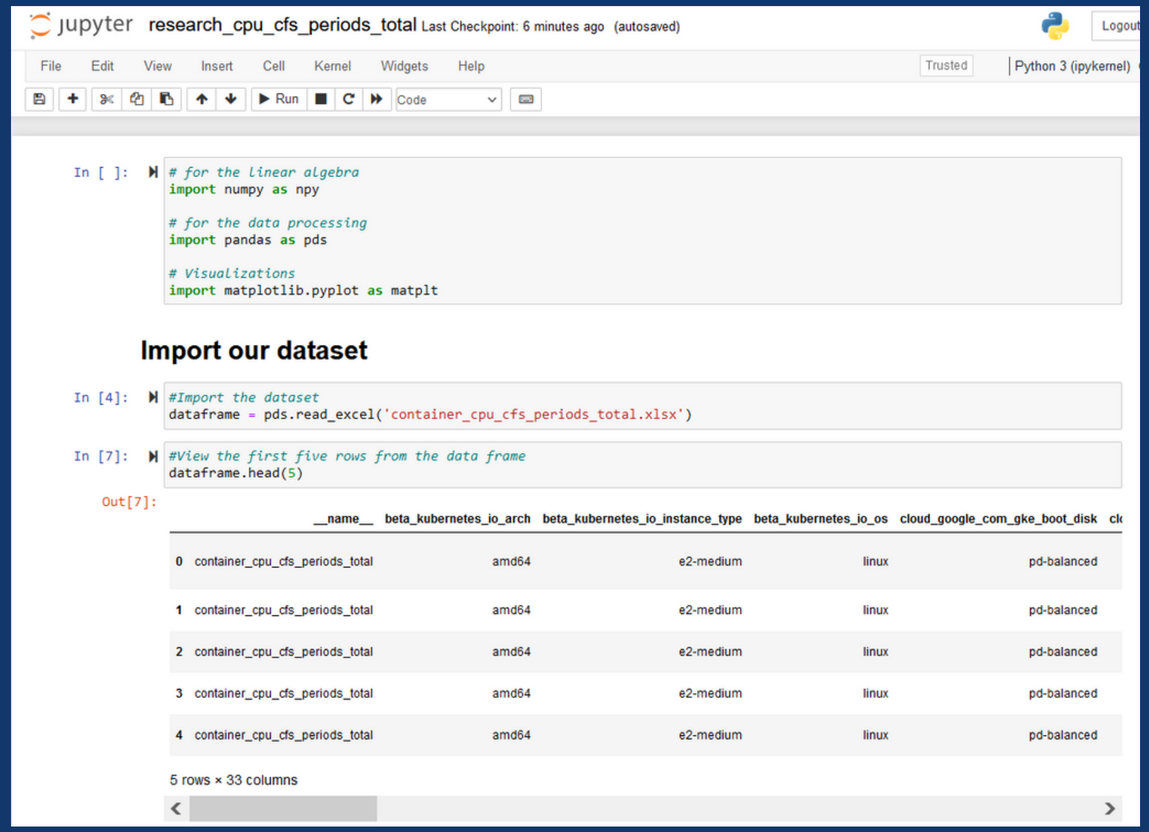



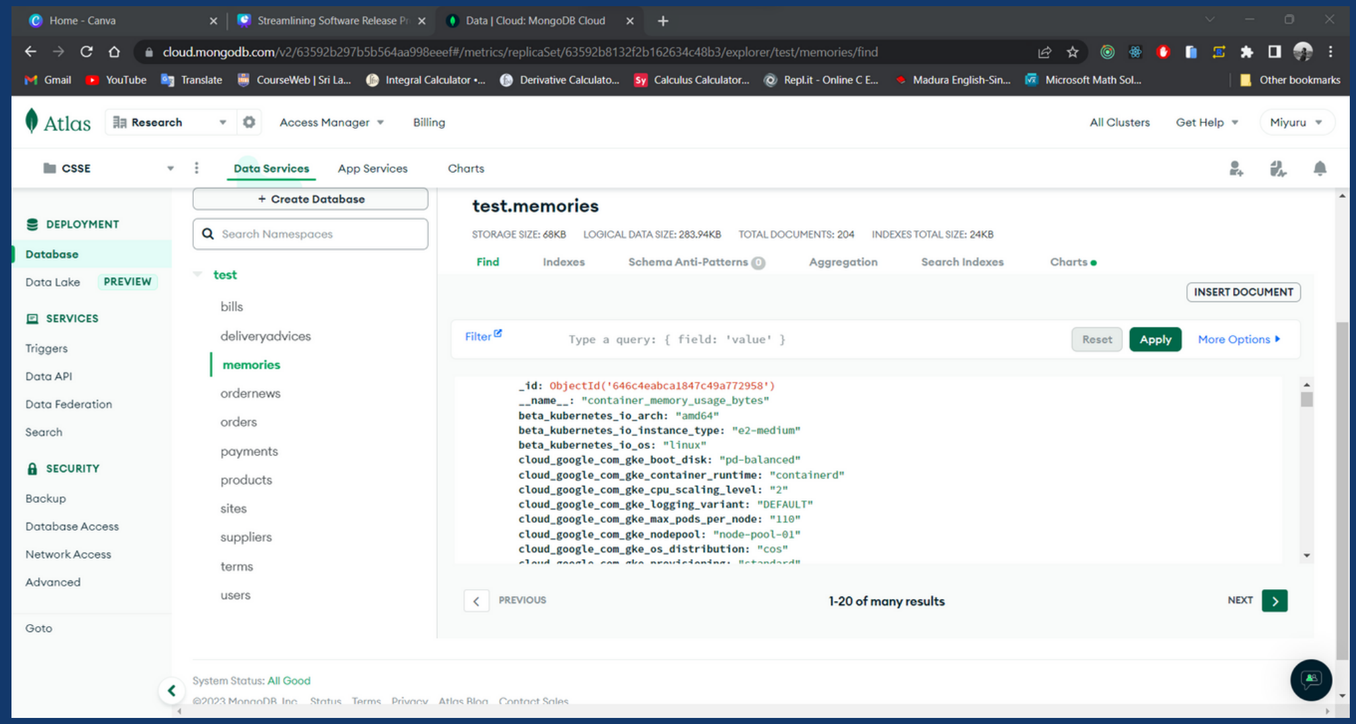
# Proof of Work



# Proof of Work



# Risk Mitigation

- when using the Reinforcement algorithm , there are several  techniques are available in reinforcement algorithm , to identify which technique will give the best accuracy is identifying bit hard.

- identify how to redeploy , once identify the prediction according to the time series metrice .

# Disaster recoverability and multi-cloud application deployment

**Gunarathna M. V. M. P. - IT19963402**
**BSc Software Engineering (SLIIT)**

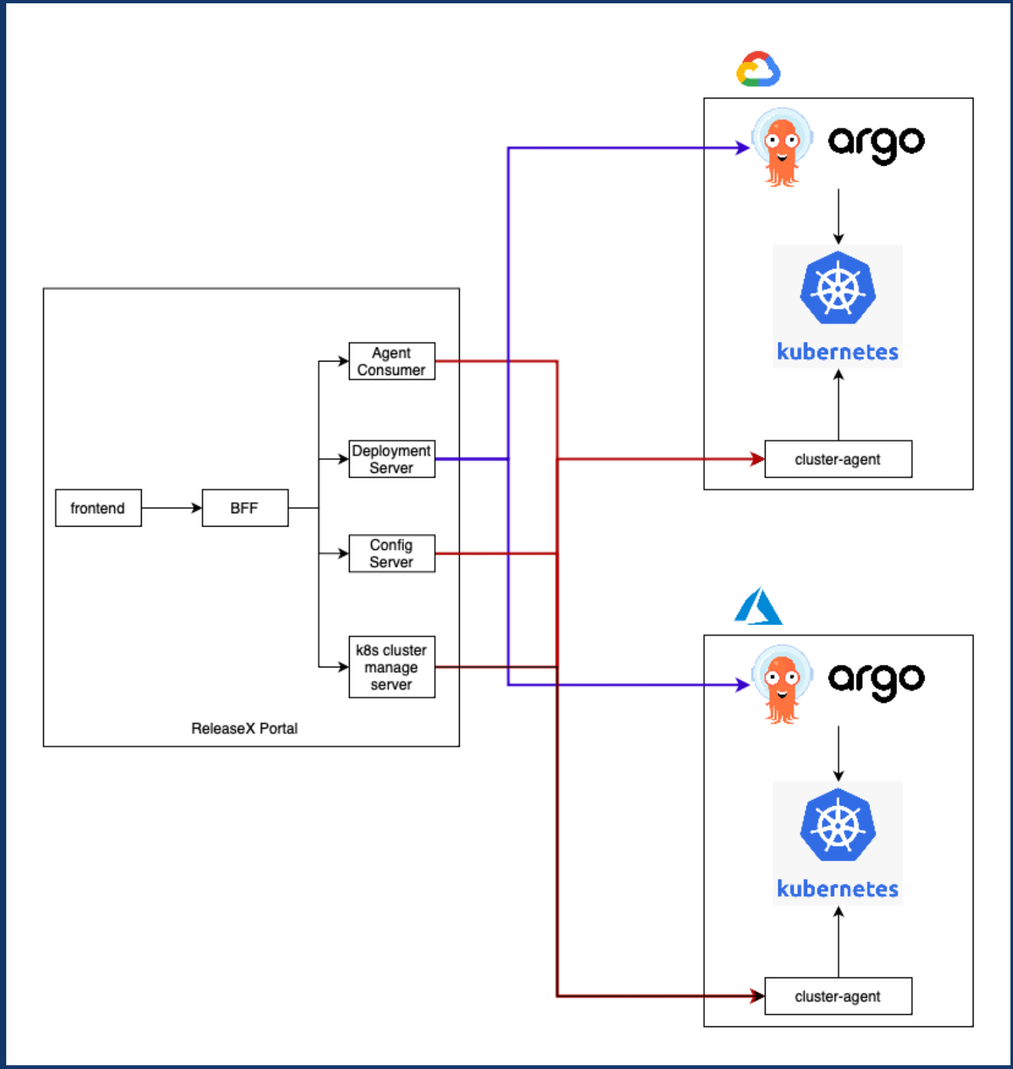
# Target Problem

Most of the existing solutions are only providing infrastructure provisioning. Therefor handling multi-cloud deployments and disaster recovery need human interaction to mitigate at least for once. And the complexity added when manually managing multi-cloud paradigm with disaster recovery throughout those multi-clouds.

# Proposed solution

Develop a single solution which automate the process of infrastruction provisioning and service level deployments on multi-cloud that accept user configuration customization and identifing incedents of clusters and redeploying on available cloud region or a different cloud vender while minimizing downtime and cost.

# System Architecture



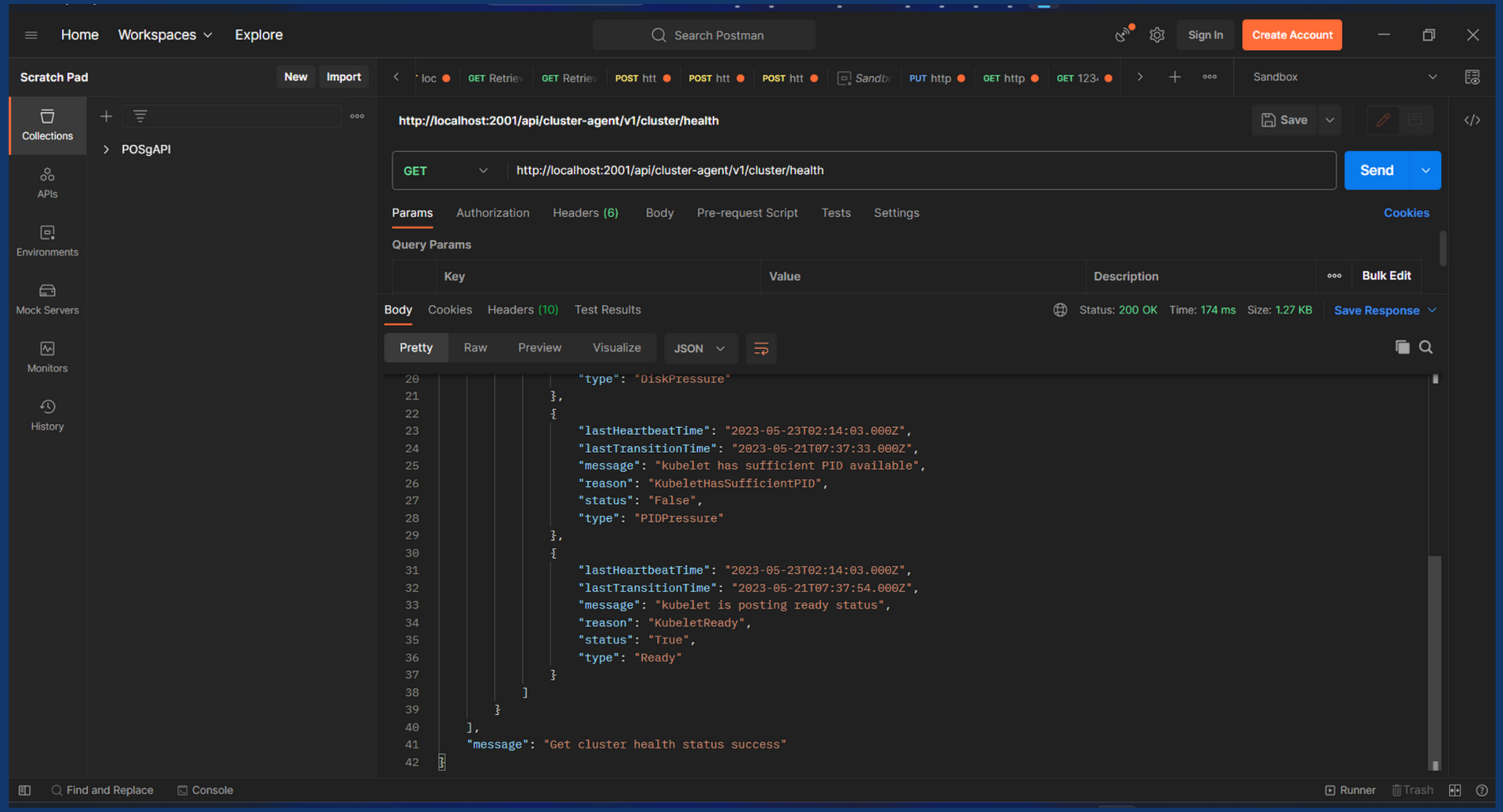
# Current Progress (35%)

- Setup a local kubernetes cluster to deploy and test the services.
- Implemented an agent to extract cluster status and expose it outside to the cluster.
- Wrote yml files to deploy cluster agent inside the cluster.
- Deployed cluster agent and test it through Postman.

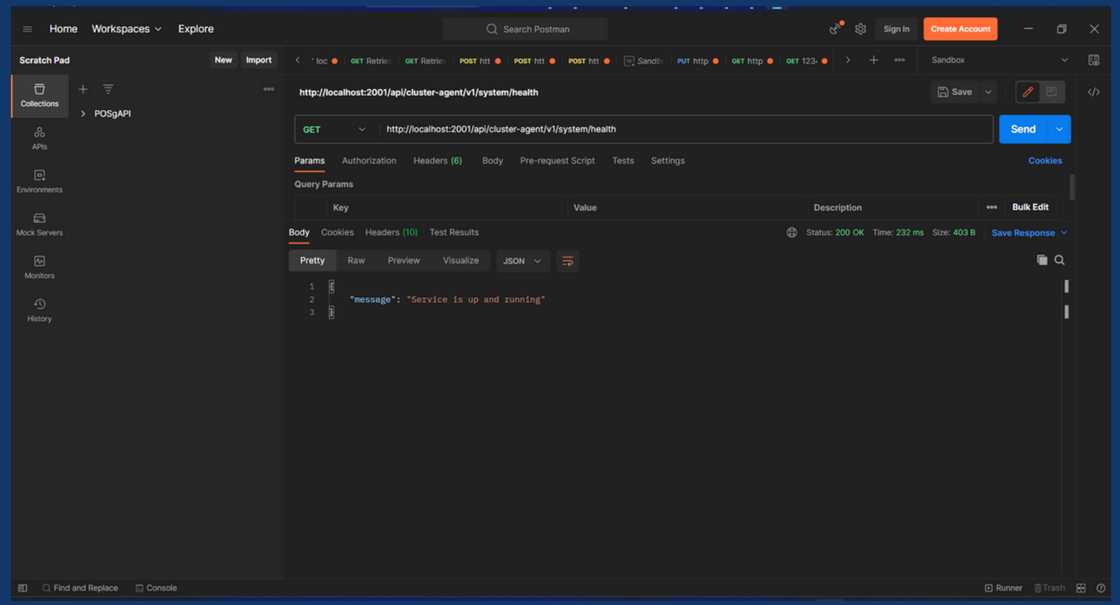
# Next Expected Progress (100%)

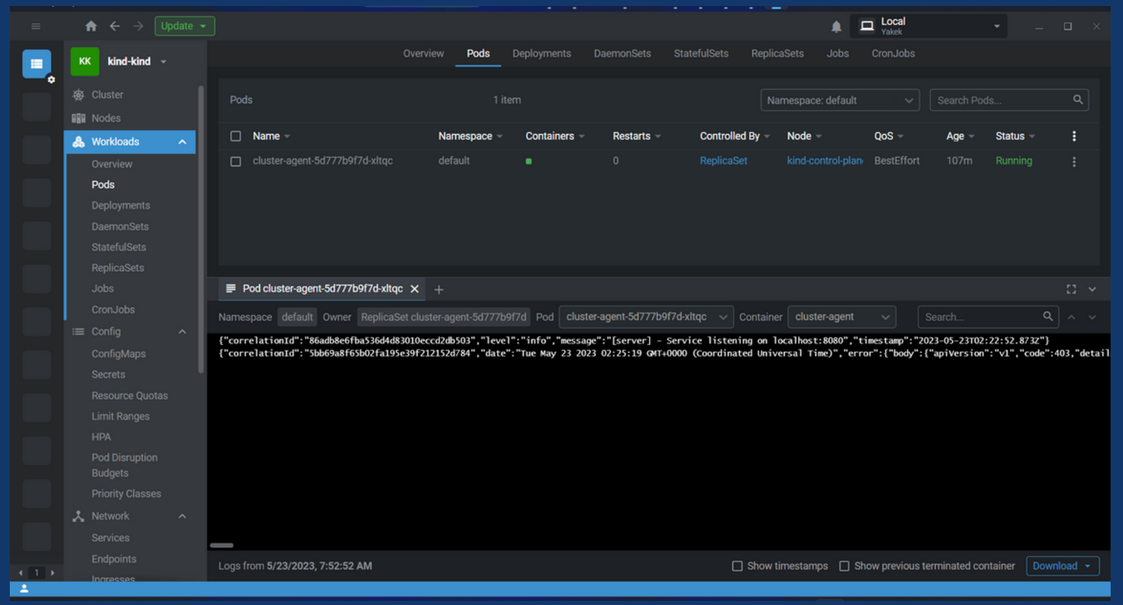
- Develop a application to configure clouds and customization need to do and visualize cluster status.
- Extend it's backend to communicate with the cluster agents inside k8 clusters and develop a websocket to update those data in real-time.
- Deploy ArgoCD on all clusters and develop a micro-service to consume it's API to automate the multi-cloud deployments and disaster recovery by the data got from the agent.
- Develop another micro-service to scale up and down the cluster when it necessary to main the cost and adapt to smooth cluster transition.
- Integrate all those micro-services to the dashboard.
- Deploy the dashboard services into itself to consume our service and get the benifits and assurence of the disaster recovery.

# Proof of Work

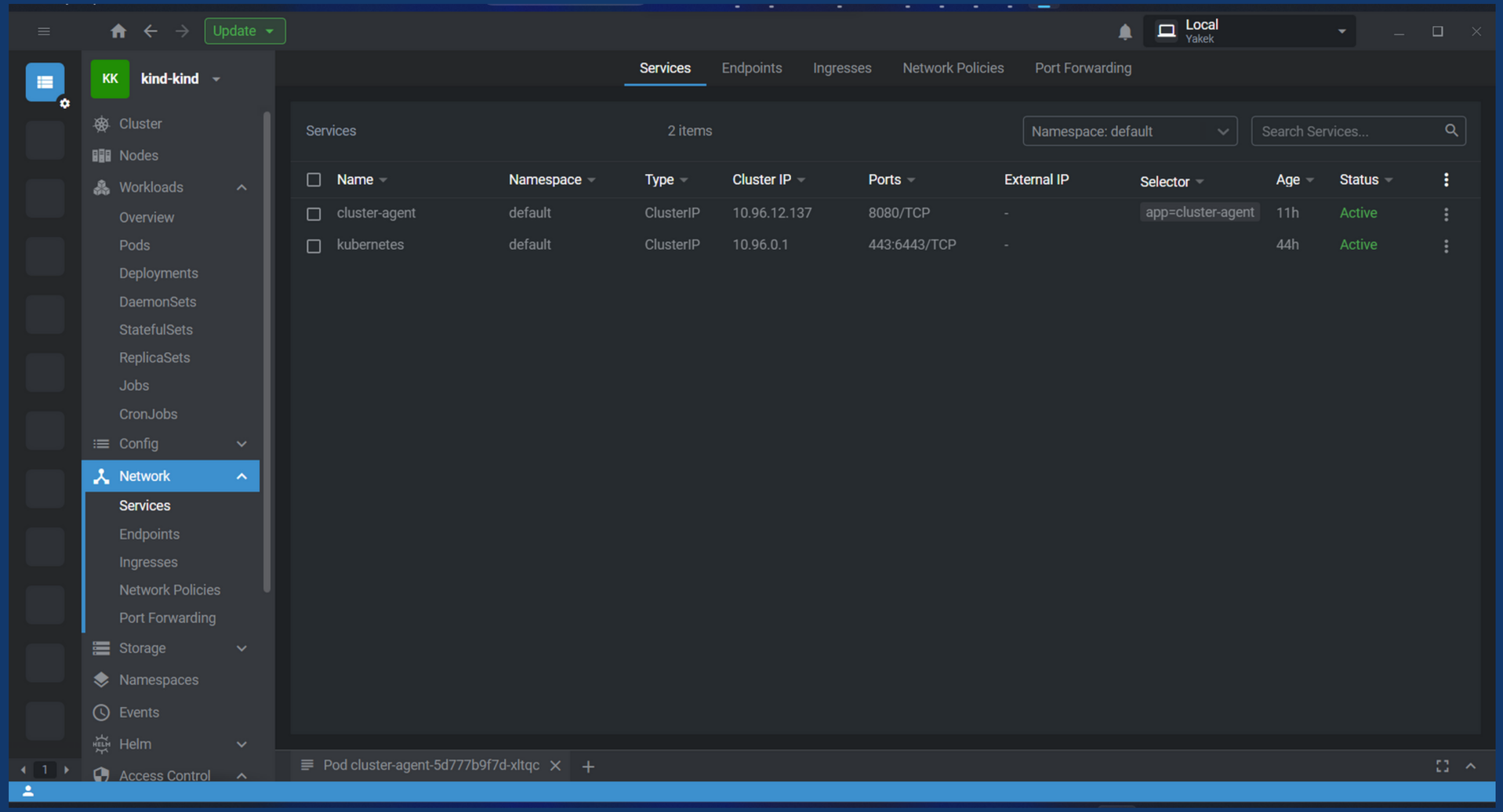


# Proof of Work



# Proof of Work



# Risk Mitigation

- In the initial configurations on cluster health checkingin the server the cluster endpoint could not be accessed due to a network autherization issue.
- Started the configuration as checking for node health checking
- During doing the node checking implementation on the health checking server, the cluster health configuration could be settled correctly.
- Finally started back with cluster configurations for cluster health check in the server.

# Current Cost Estimation

| Service | Price |
|---|---|
| Google Kubernetes Cluster | USD 97.84 / month |
| Load Balancing | USD 10.00 / month |
| Jira Cloud | USD 7.75 / month |
| **Total** | **USD 115.59 / month** |

# Marketing Plan

# THANK YOU

23-193