



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

23-193

Our Team



Dr. Nuwan Kodagoda
Supervisor



Mr. Udara Samaratunge
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

Problems

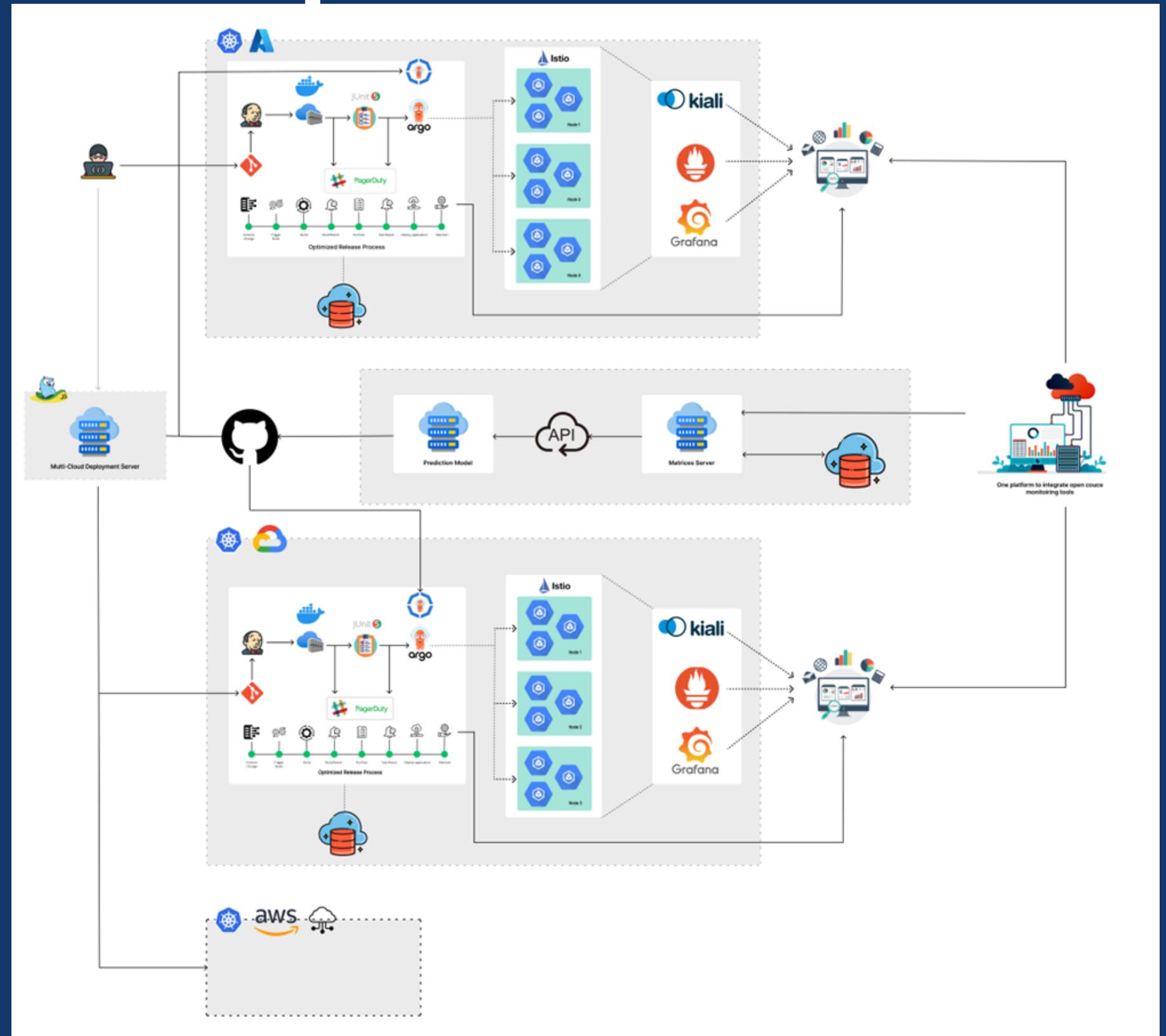


- Available software release process systems are not fully automated and optimized.**
- Available open-source monitoring tools' metrics need to be monitored in an one single platform.**
- Cannot predict the necessary resource request and resource limit for VPAs.**
- Available technologies are not focused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



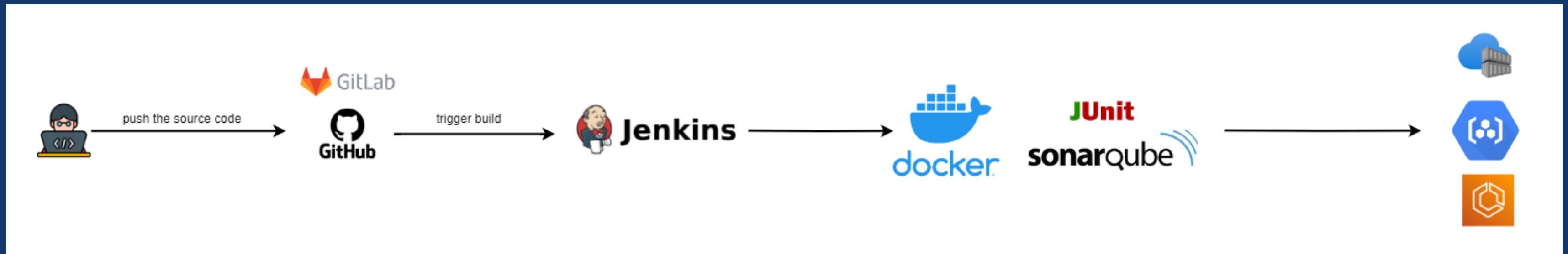
Herath H.M.I.P. - IT20125516
BSc. Software Engineering (SLIIT)

Target Problem

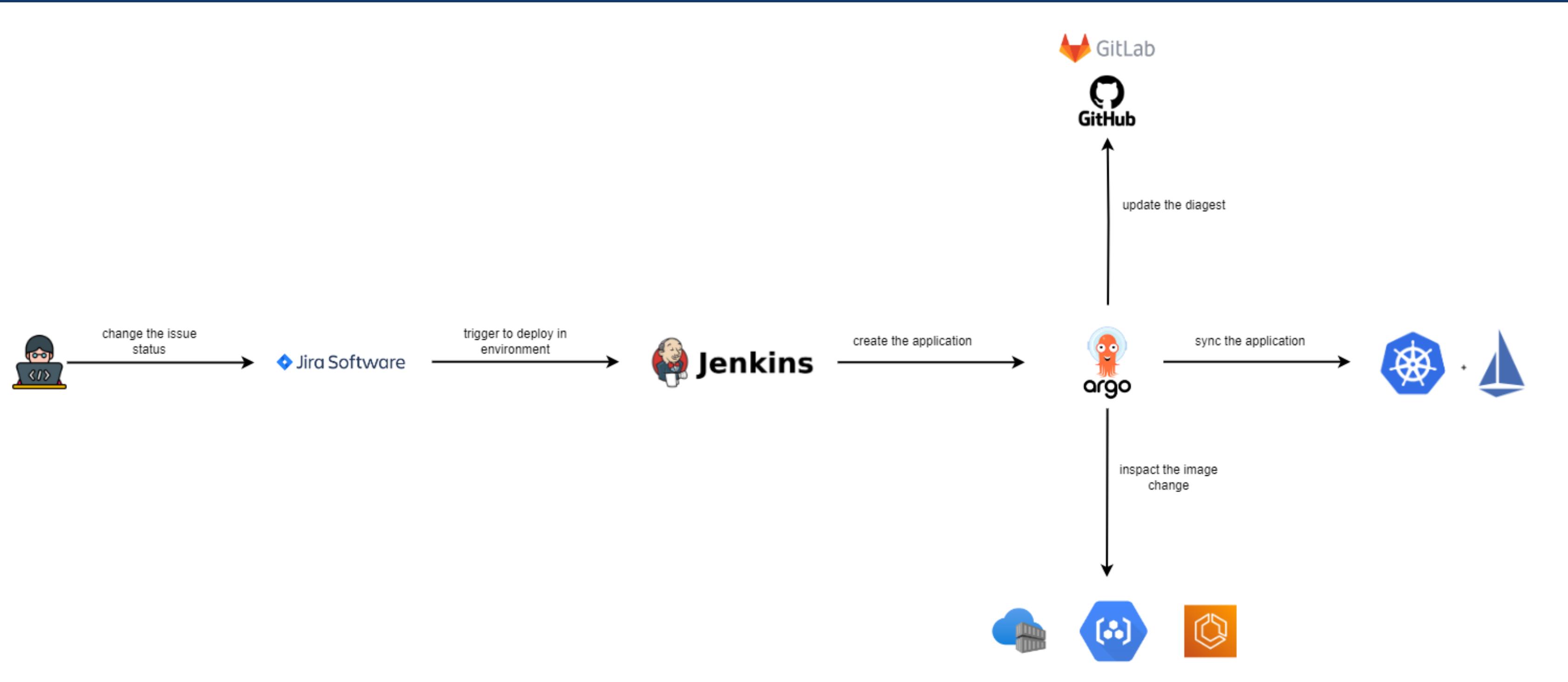
The existing methodologies for software release processes predominantly rely on manual execution, which poses several challenges such as suboptimal architecture, increased likelihood of human errors, excessive time consumption, and susceptibility to mistakes.

Proposed Solution

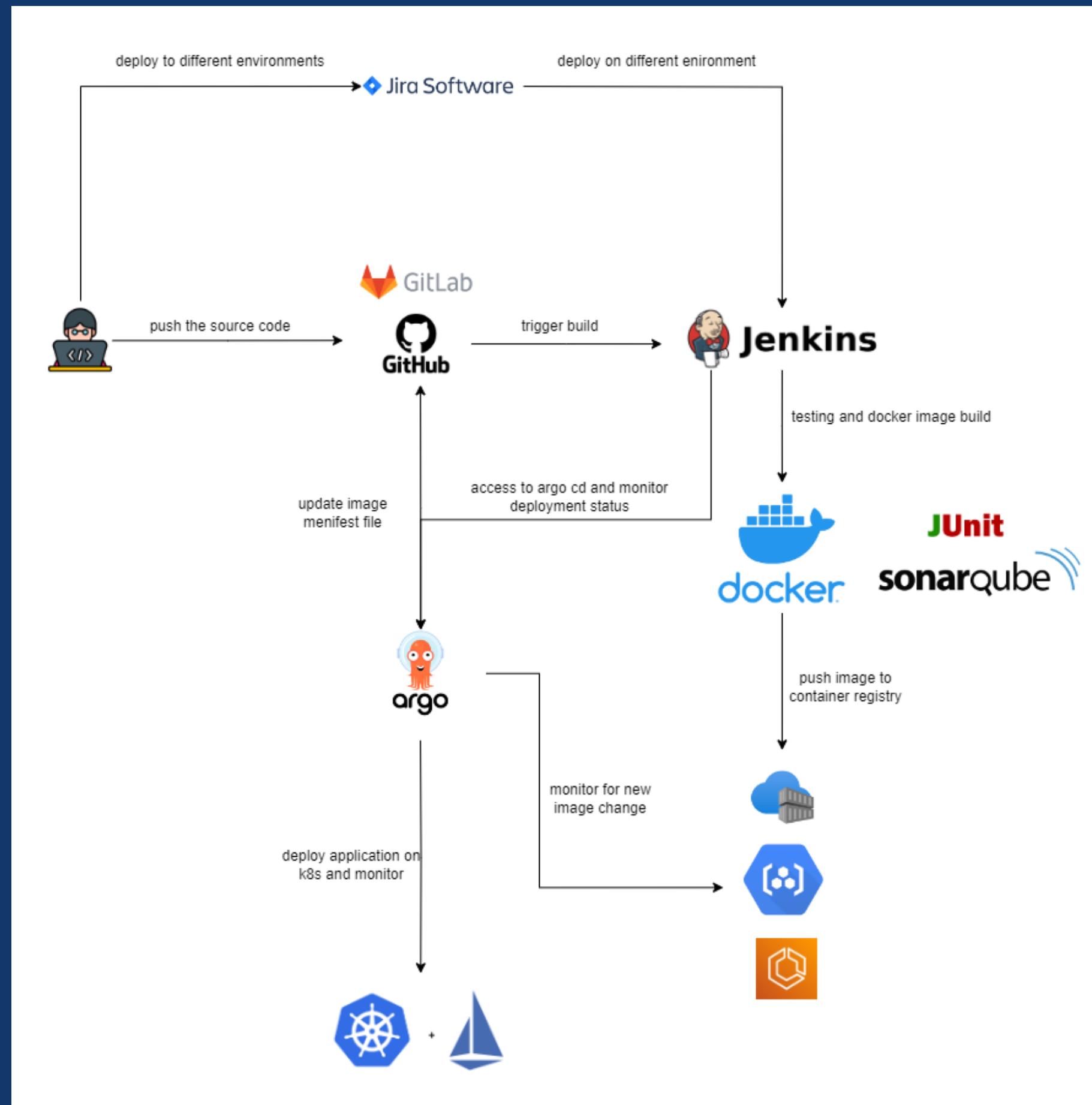
Rearchitect a software release process that can be applied commonly for most of the software release processes by optimizing the Software Release Process according to the architectural attributes such as performance, security, extendibility, reliability, and modifiability and fully automating the proposed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



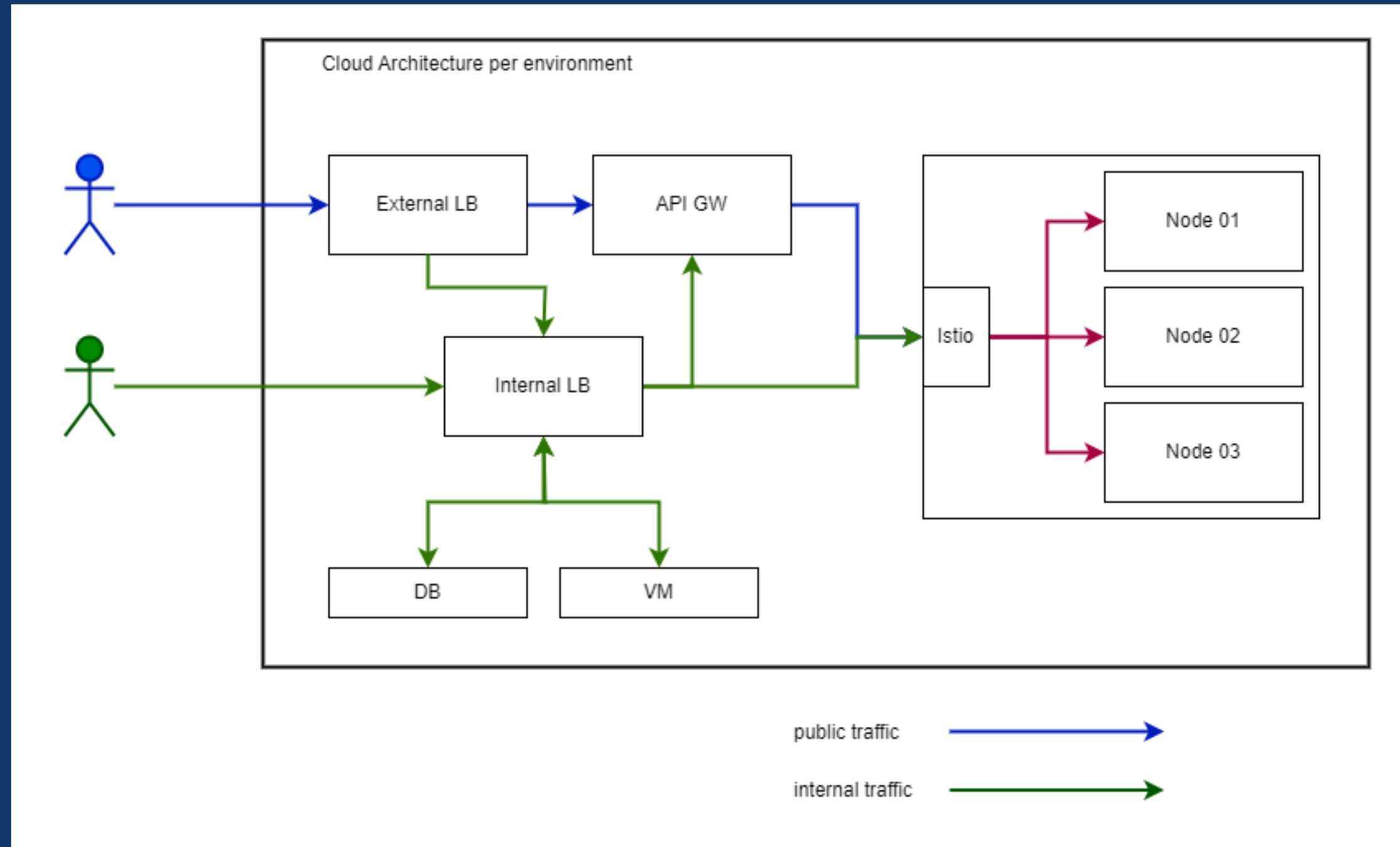
proposed CI Architecture



proposed CD Architecture



proposed CI/CD Architecture

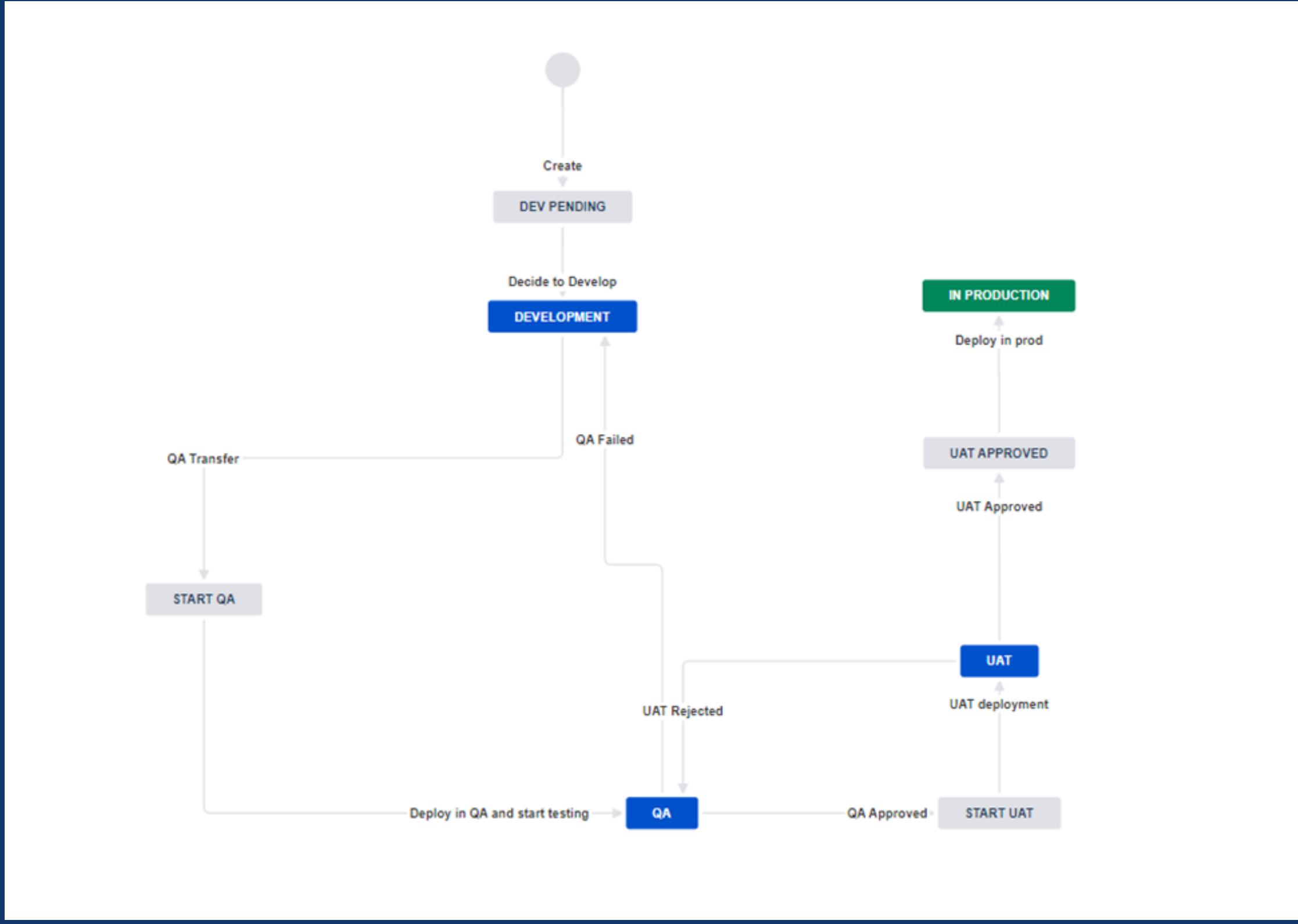


Cloud Environment Architecture

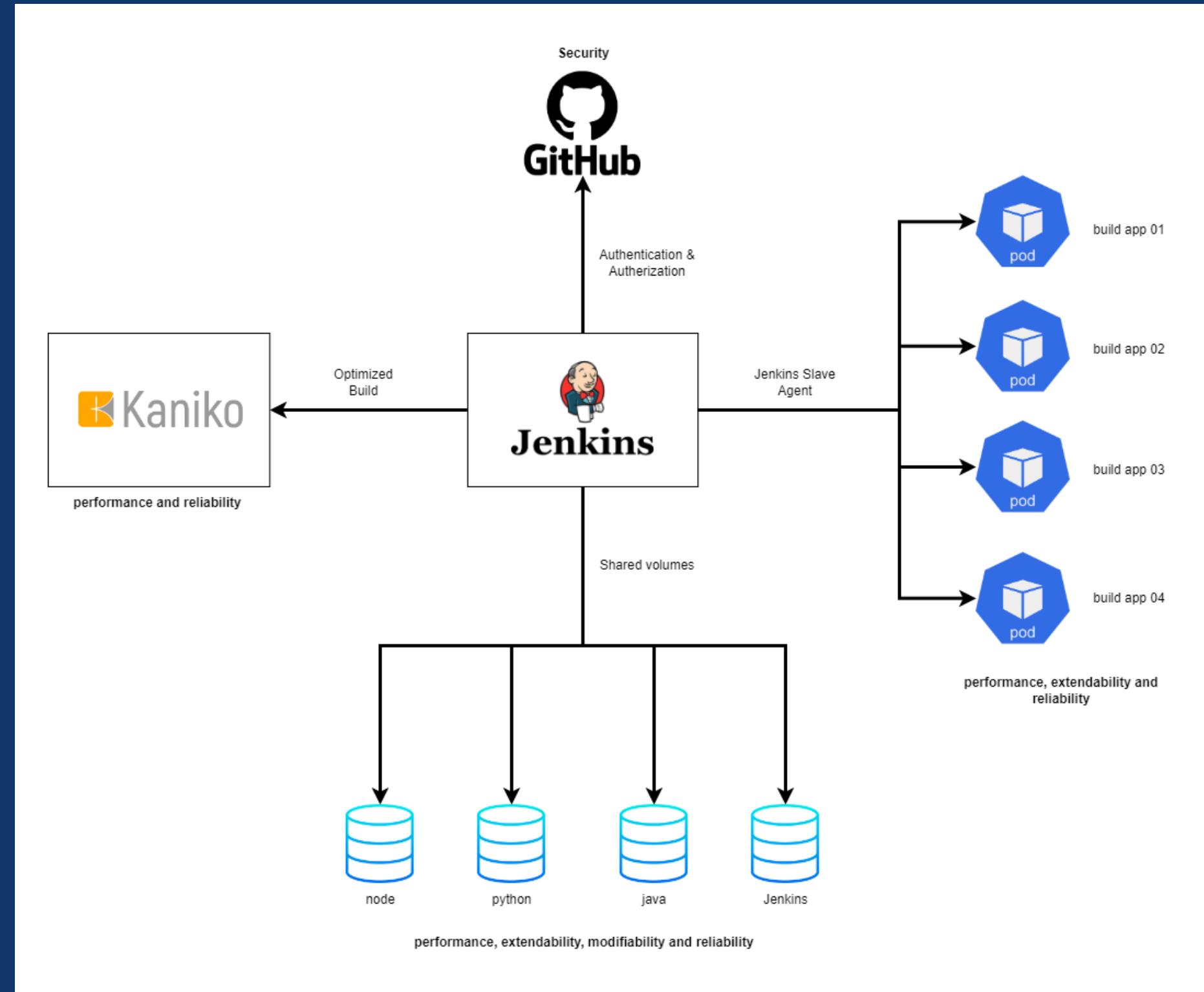
Current Progress (50%)

- 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

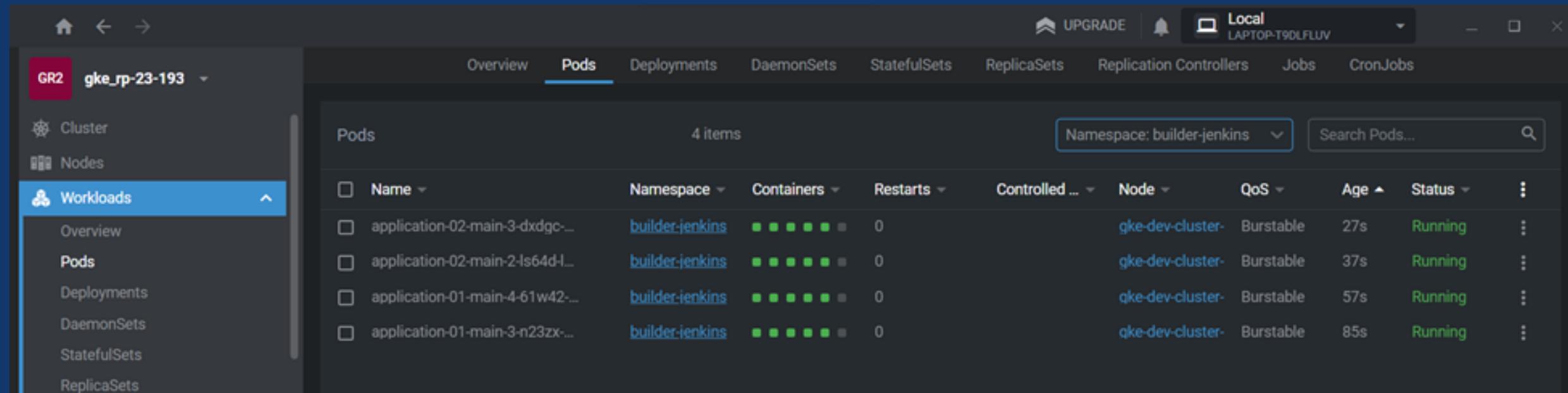
Proposed Jira Issue Change



Optimization of CI Pipeline Architecture

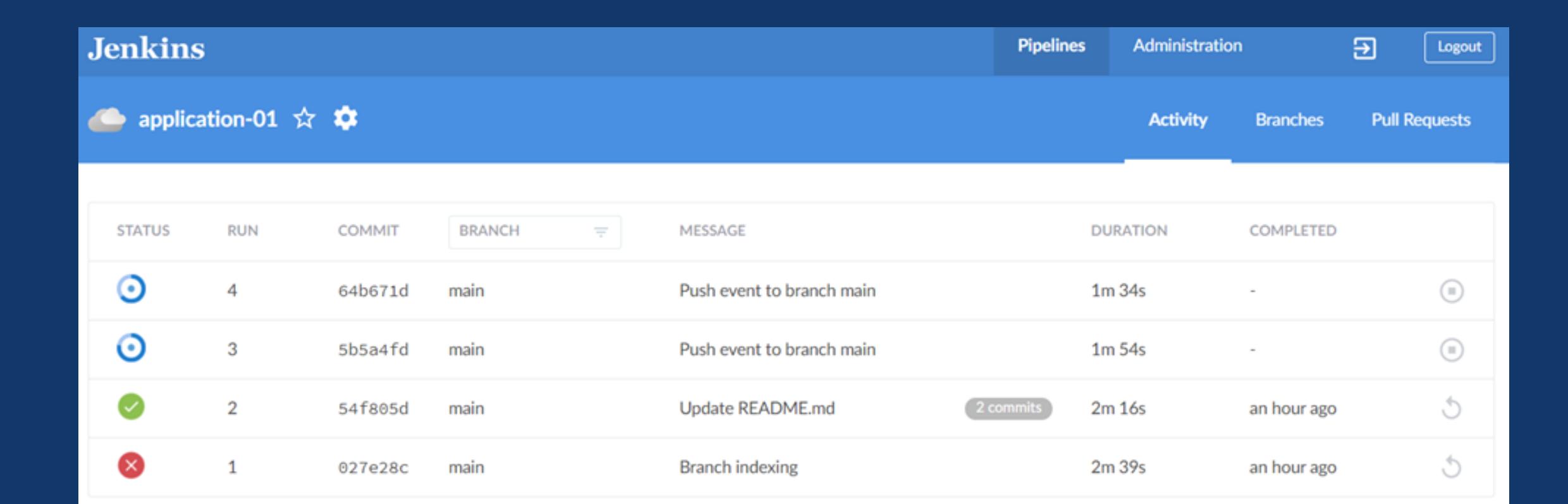


Proof of Work



The screenshot shows the Google Cloud Platform Kubernetes dashboard for cluster GR2. The 'Pods' tab is selected. There are four items listed:

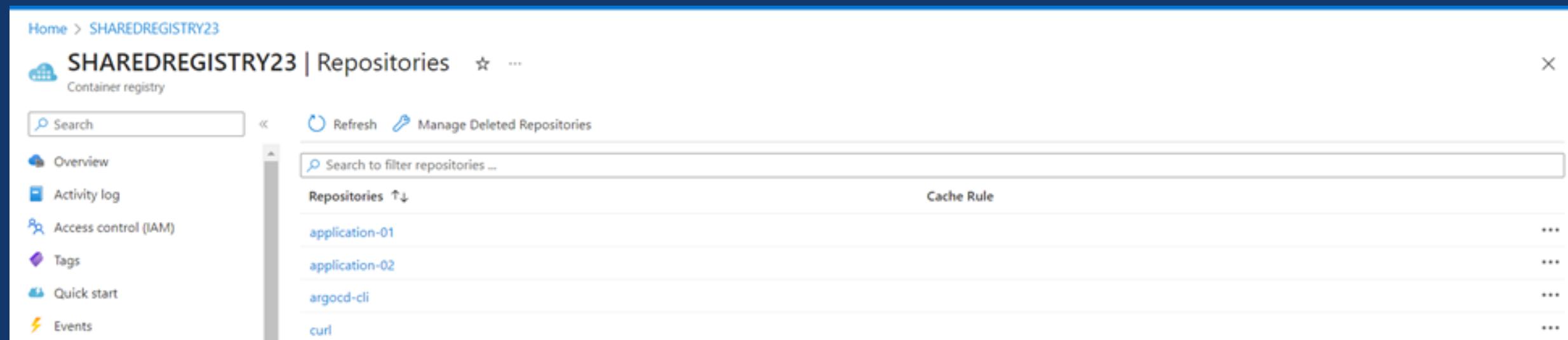
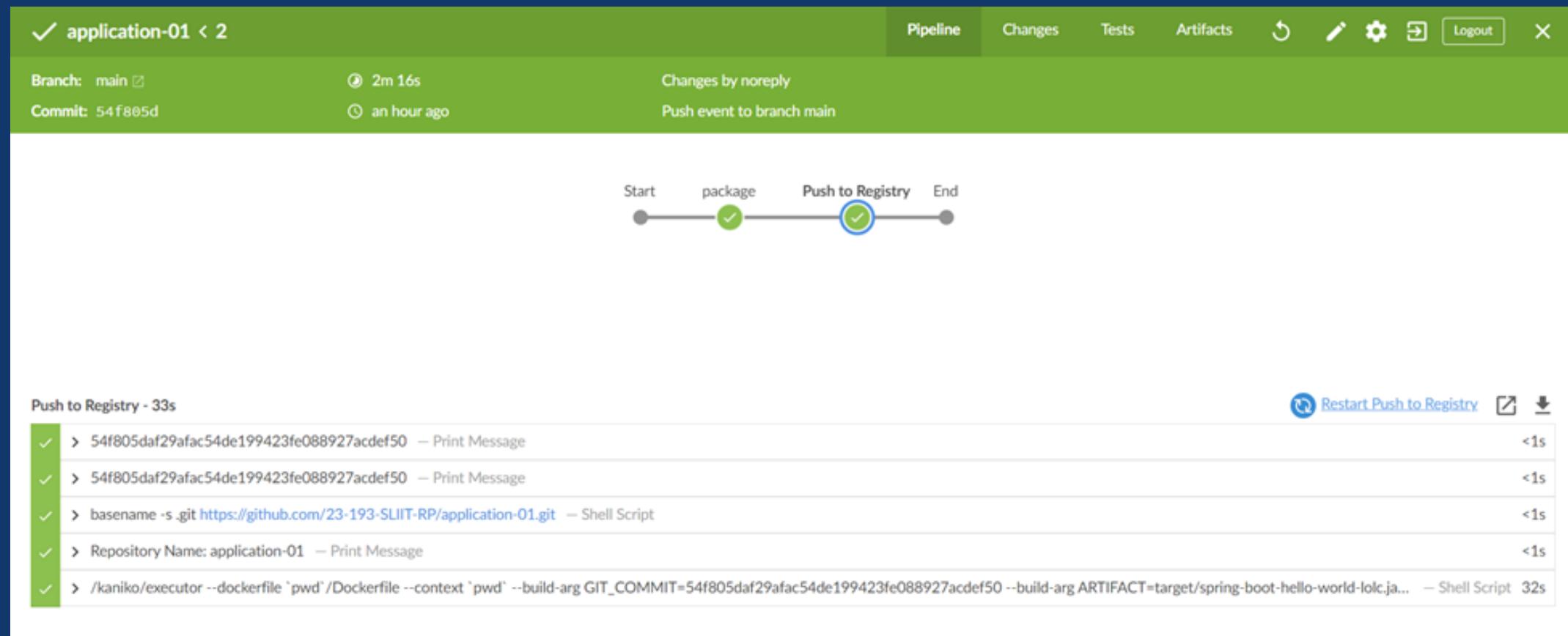
Name	Namespace	Containers	Restarts	Node	QoS	Age	Status
application-02-main-3-dxdgc-	builder-jenkins	4	0	gke-dev-cluster-	Burstable	27s	Running
application-02-main-2-ls64d-l...	builder-jenkins	4	0	gke-dev-cluster-	Burstable	37s	Running
application-01-main-4-61w42...	builder-jenkins	4	0	gke-dev-cluster-	Burstable	57s	Running
application-01-main-3-n23zx...	builder-jenkins	4	0	gke-dev-cluster-	Burstable	85s	Running



The screenshot shows the Jenkins pipeline interface for the 'application-01' project. The 'Activity' tab is selected. Four recent builds are listed:

Status	Run	Commit	Branch	Message	Duration	Completed
Success	4	64b671d	main	Push event to branch main	1m 34s	-
Success	3	5b5a4fd	main	Push event to branch main	1m 54s	-
Success	2	54f805d	main	Update README.md 2 commits	2m 16s	an hour ago
Failure	1	027e28c	main	Branch indexing	2m 39s	an hour ago

Proof of Work



Proof of Work

The image displays three screenshots of a Kubernetes user interface:

- Pod Log Viewer:** Shows the logs for a pod named "application-02-main-3-dxdgc-3k9nr-sg2v4" in the "builder-jenkins" namespace. The container selected is "jnlp". The logs show Jenkins remoting starting up, connecting to a server, and performing protocol handshakes.
- Cluster Overview Dashboard:** A summary page showing the status of various Kubernetes resources across all namespaces. It includes sections for Pods (52), Deployments (33), Daemon Sets (19), Stateful Sets (1), Replica Sets (63), and Jobs (0). Each section shows a green circle icon and the count of running pods.
- Detailed Pod List:** A table listing 52 items under the "Pods" tab. The columns include Name, Namespace, Containers, Restarts, Controlled By, Node, QoS, Age, and Status. Examples of listed pods include "application-02-main-3-dxdgc...", "application-02-main-4-61w42...", and "jenkins-7fdf895d-c9pxn".

Proof of Work

The image displays three screenshots illustrating a continuous integration pipeline between Jira and Jenkins.

- Jira Board:** Shows a sprint backlog for "TB Sprint 1". The board has columns: TO DO, IN PROGRESS, and DONE. Issues include "test 5" (status: In Progress), "test-now-01" (status: To Do), "test 3" (status: In Progress), and "test 1" (status: Done). Each issue has a checkmark icon and a "Jira" link.
- Jenkins Dashboard:** Shows the Jenkins logo and navigation links: Dashboard > Manage Jenkins > System Log > jenkins-jira. A search bar at the top right contains the text "Search (CTRL+K)".
- Jenkins Log Records:** Shows the "jenkins-jira" log records. The log output consists of multiple entries from May 22, 2023, at various times, all reporting "Received Webhook callback from changelog. Event type: jira:issue_updated".

Trello Board

2023-193 - Streamlining software release process and resource management for microservices-based architecture on multi-cloud

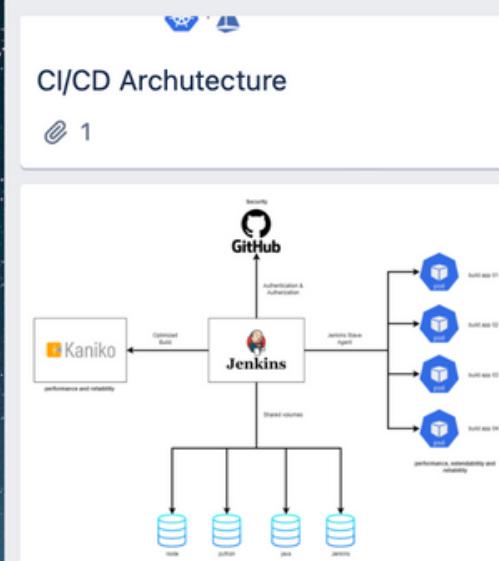
Todo

- Design the Infrastructure provisioning Architecture based on Microservice based architecture
- Design the DevOps Architecture - Software release process based on a streamlined and optimized way according to the research component
- Implement terraform code to infrastructure provision automation
- Implement the DevOps Architecture

Completed Todos

- Topic Assessment
- Charter Documentation Submission
- Proposal Presentation
- Proposal Report Submission

Herath H.M.I.P - IT20125516

- CI/CD Archutecture**
A diagram showing a Jenkins instance at the center. It receives inputs from GitHub (Authenticaion & Authorization) and Kaniko (Optimized build). Jenkins then performs shared volumes and builds for Node.js, Python, Java, and .NET. Each build step is labeled with its name and a small icon.
- Optimization Architecture - CI**
A diagram showing a Jenkins instance at the center. It receives inputs from GitHub and Kaniko. Jenkins then performs shared volumes and builds for Node.js, Python, Java, and .NET. Each build step is labeled with its name and a small icon.
- CD Architecture**
A diagram showing a Jenkins instance at the center. It receives inputs from GitHub and Kaniko. Jenkins then performs shared volumes and builds for Node.js, Python, Java, and .NET. Each build step is labeled with its name and a small icon.

IT20125516 - Completed

- Proposal Presentation Completed
- Proposal Report Completed
- Terraform R&D basic knowledge Completed
- Istio Service Mesh successfully configured on the GKE.
- ArgoCD R&D - ArgoCD Configured and Successfully exposed.
- Jenkins R&D - Jenkins installed and exposed successfully.
- Exposed with public IP address with istio ingress controller and bind with a domain name "releasex.tech"
- The main CI Architecture is implemented with ACR and Jenkins. Used Kaniko to build the docker image.

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)

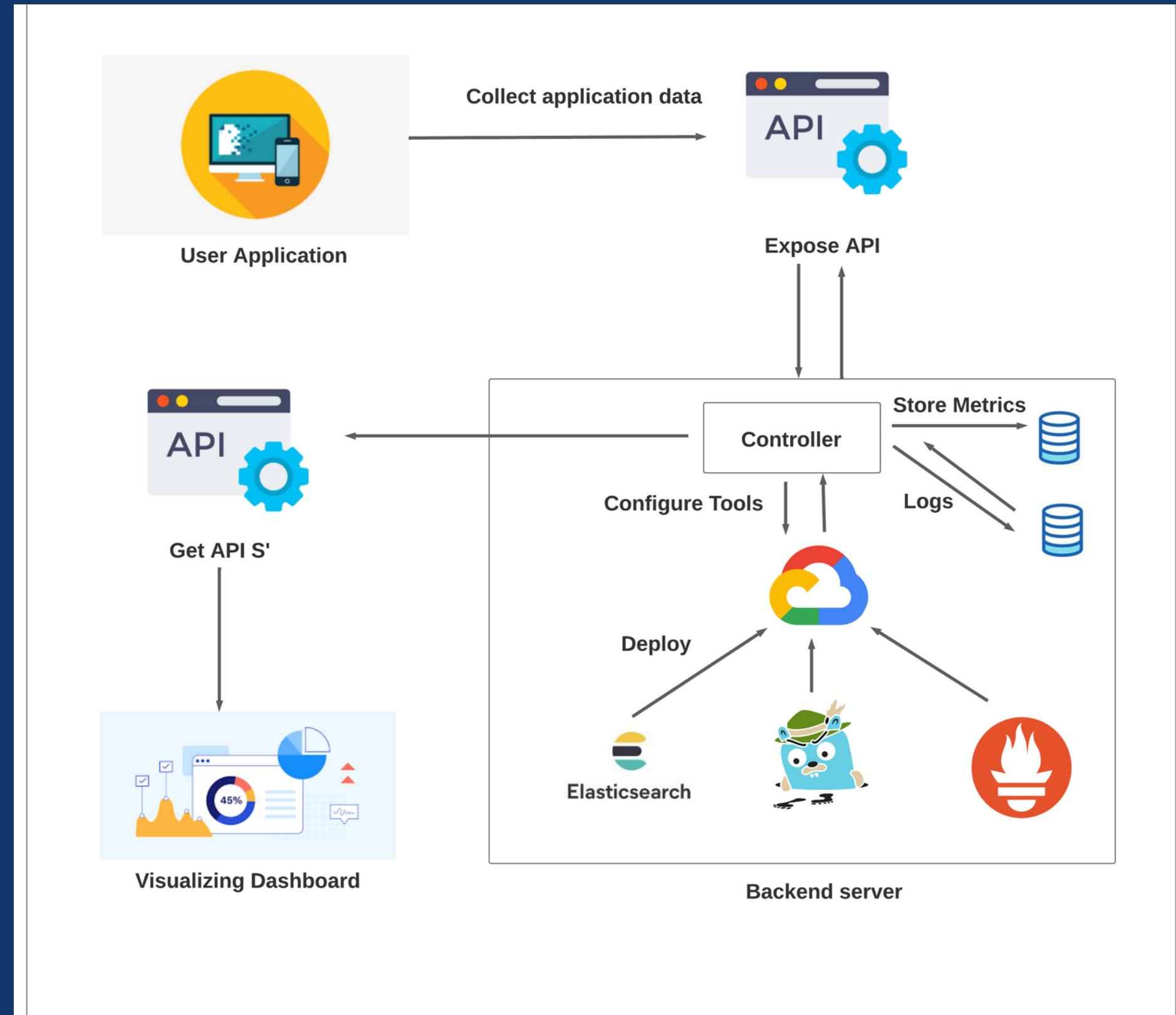
Target Problem

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 Serilog 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

The screenshot shows the Swagger UI interface for a 'MonitoringAPI'. The title bar indicates the URL is `localhost:7024/swagger/index.html`. The main content area is titled 'MonitoringAPI 1.0 OAS3' and shows the endpoint `https://localhost:7024/swagger/v1/swagger.json`. Below this, there is a section titled 'Metrics' containing five API endpoints:

- `POST /api/Metrics/log`
- `POST /api/Metrics/metric`
- `POST /api/Metrics/timeseries`
- `GET /api/Metrics/metrics`
- `POST /api/Metrics/traces`

Below the 'Metrics' section is a 'Schemas' section with a single entry: `LogModel >`.

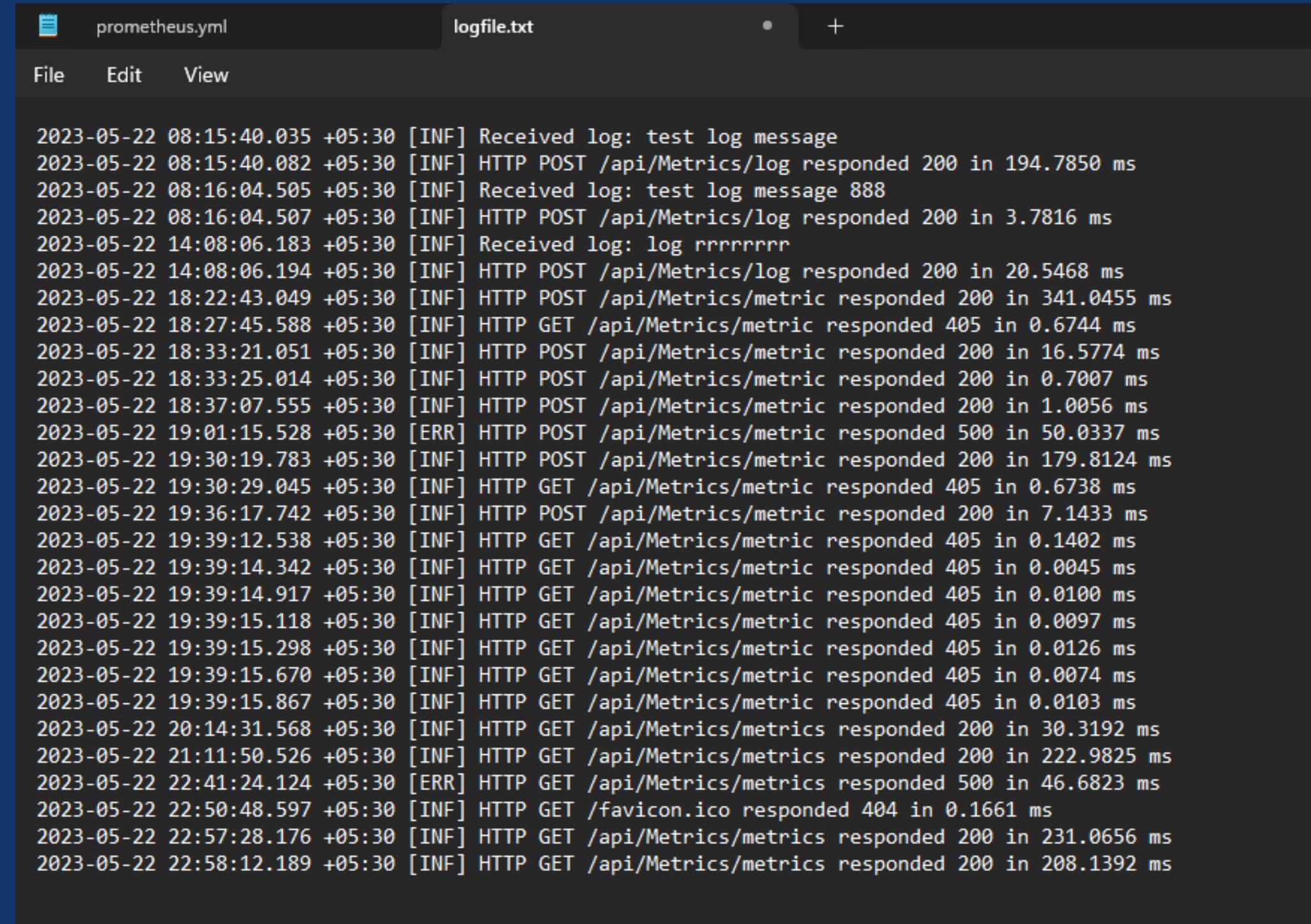
Swagger API dashboard

PROOF OF WORK

The screenshot shows a Swagger UI interface for an API. At the top, there's a "Responses" section. Below it, under "Curl", is a code block for a POST request to `curl -X 'POST' \ 'https://localhost:7024/api/Metrics/log' \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{\n "message": "test log message 123"\n}'`. Under "Request URL" is `https://localhost:7024/api/Metrics/log`. The "Server response" section shows a 200 status code. The "Details" tab for 200 shows "Response headers" with `access-control-allow-origin: *\ncontent-length: 0\ndate: Mon,22 May 2023 19:23:09 GMT\nserver: Kestrel`. The "Responses" section shows a 200 status code with "Description" "Success" and "Links" "No links".

API Response

PROOF OF WORK



A screenshot of a terminal window titled "logfile.txt". The window shows a list of log entries from May 22, 2023. The logs include various HTTP requests and responses, some with errors, and some test log messages. The log entries are timestamped and show response times in milliseconds.

```
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

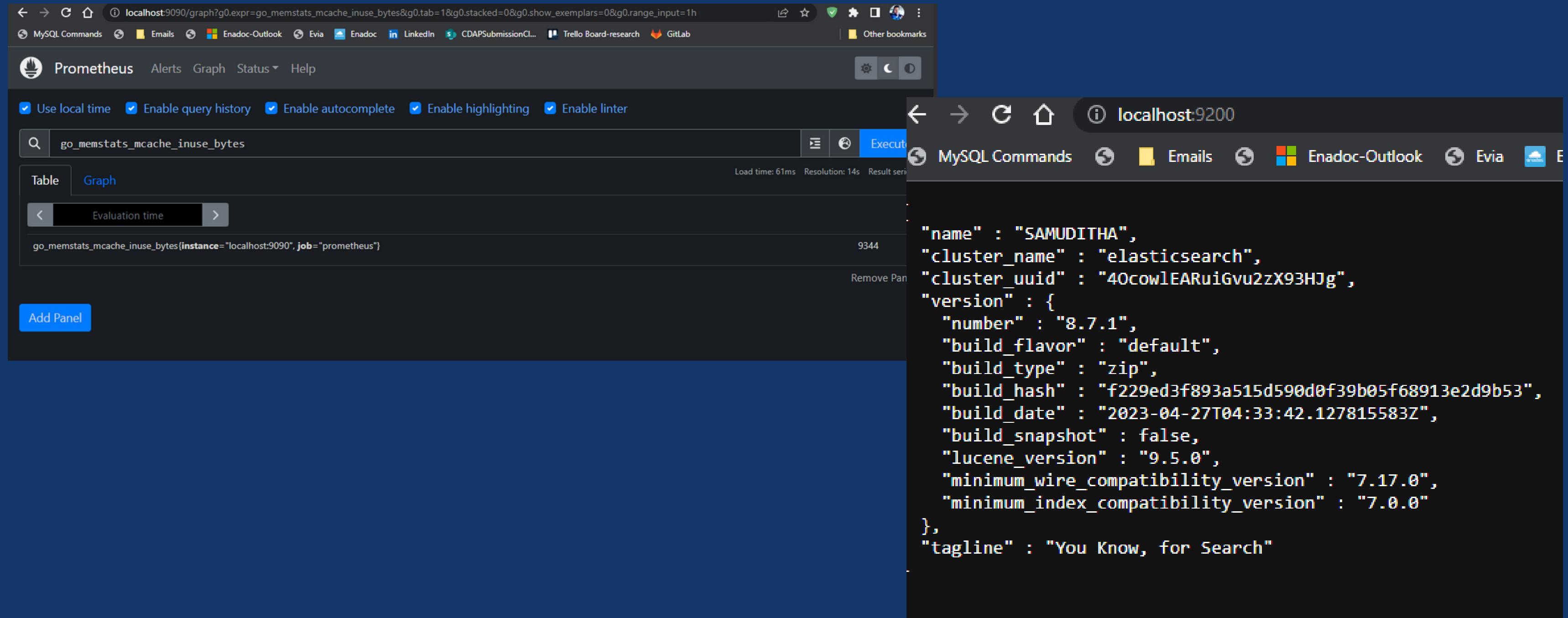
PROOF OF WORK

The screenshot shows the Microsoft Visual Studio IDE interface. The main area displays the `MetricsController.cs` file, which contains C# code for a controller that handles log and metric data. The code includes constructor injection for `HttpClient`, `Serilog.ILogger`, and `OpenTracing.ITracer`. It features two `[HttpPost]` methods: `LogData` and `MetricData`. The `LogData` method logs a message to the application's log, while the `MetricData` method returns a success response. The `No issues found` status bar at the bottom indicates no compilation errors.

The right side of the screen features the **Diagnostic Tools** window, which provides real-time monitoring of the application's performance. The **Diagnostics session** has been running for 1:40 minutes. The **Process Memory** tab shows memory usage with a current value of 260 MB. The **CPU (%) of all processors** tab shows CPU usage at 0%. Below these are sections for **Events**, **Memory Usage**, and **CPU Usage**.

Backend code

PROOF OF WORK



The image shows two browser windows side-by-side. The left window is the Prometheus interface at localhost:9090, displaying a query for 'go_memstats_mcache_inuse_bytes' which returns a single result: 9344. The right window is the Elasticsearch interface at localhost:9200, displaying the cluster's metadata in JSON format.

```
localhost:9090/graph?g0.expr=go_memstats_mcache_inuse_bytes&g0.tab=1&g0.stacked=0&g0.show_exemplars=0&g0.range_input=1h
{
  "name": "SAMUDITHA",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "40cowlEARuiGvu2zX93HJg",
  "version": {
    "number": "8.7.1",
    "build_flavor": "default",
    "build_type": "zip",
    "build_hash": "f229ed3f893a515d590d0f39b05f68913e2d9b53",
    "build_date": "2023-04-27T04:33:42.127815583Z",
    "build_snapshot": false,
    "lucene_version": "9.5.0",
    "minimum_wire_compatibility_version": "7.17.0",
    "minimum_index_compatibility_version": "7.0.0"
  },
  "tagline": "You Know, for Search"
}
```

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)

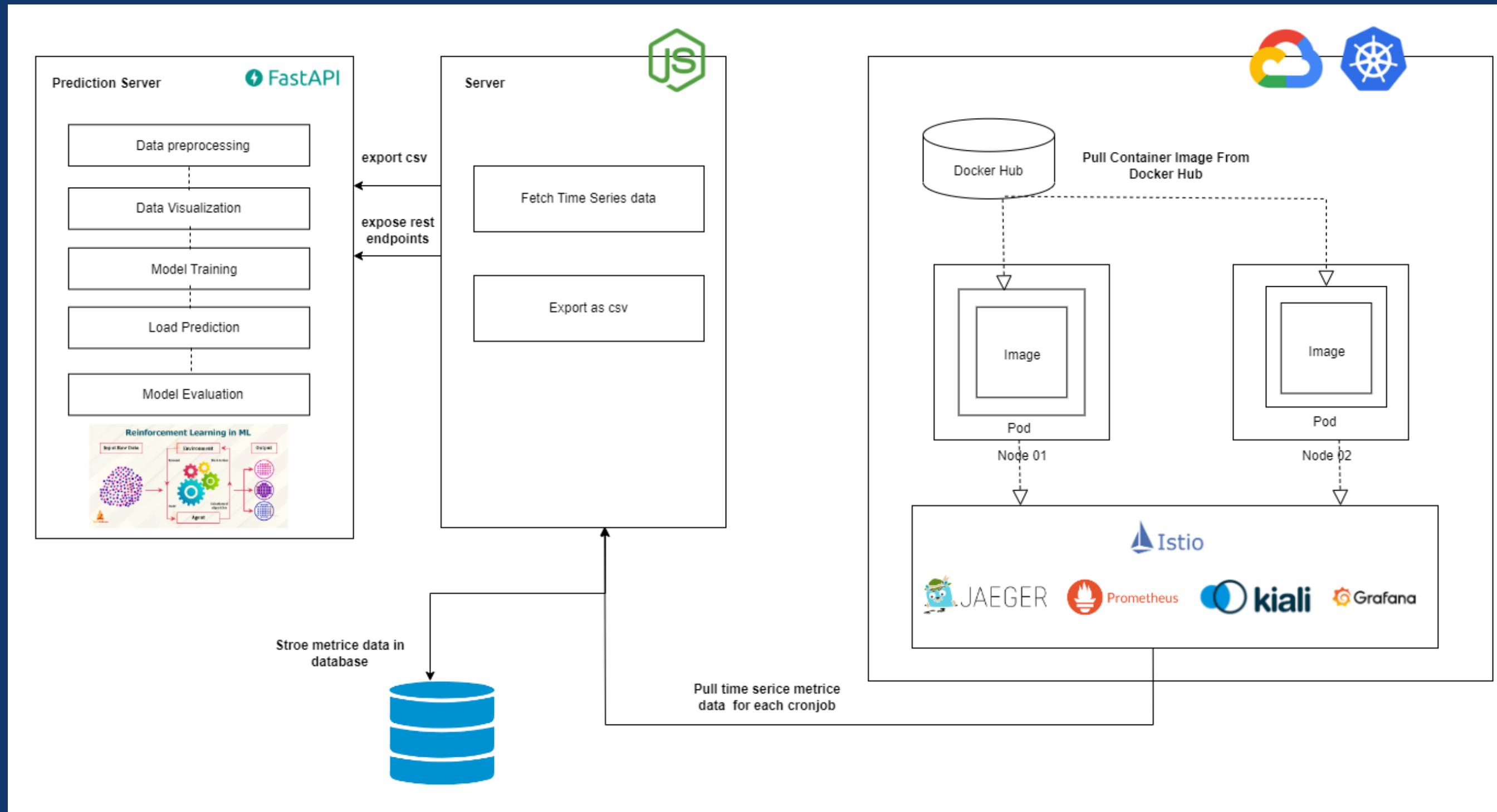
Target Problem

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 gathering 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.

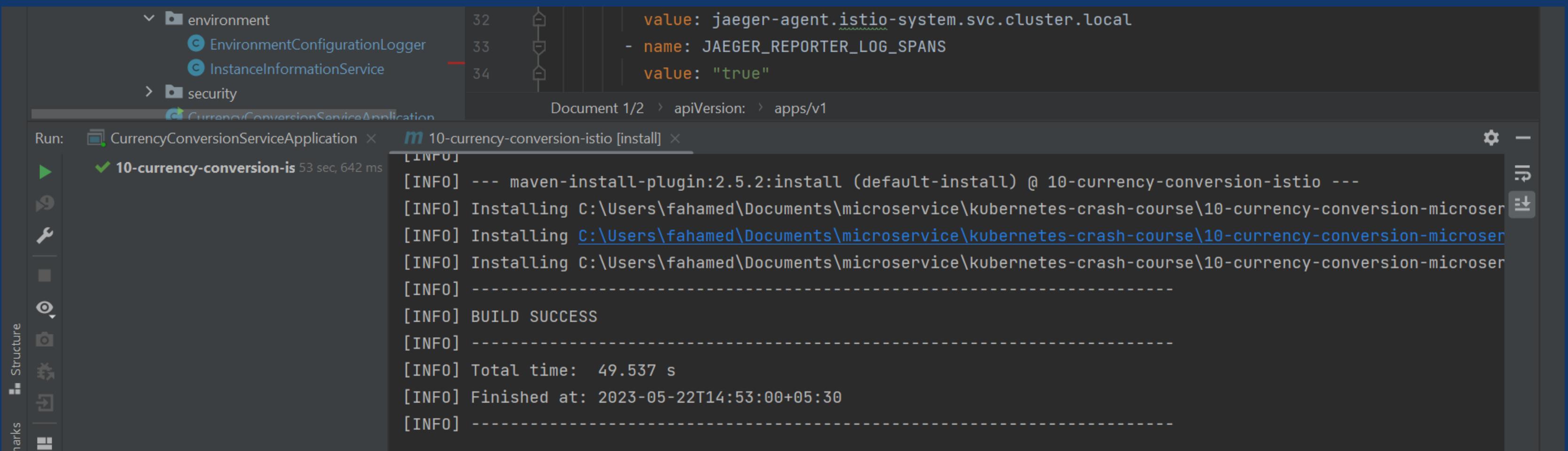
Proof of Work

```
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-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=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
export INGRESS_PORT=$(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-grafana-dashboards created
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

<input type="checkbox"/>	istio-egressgateway	✓ OK	Cluster IP	10.96.4.154	1/1	istio-system	dev-clu...	
<input type="checkbox"/>	istio-ingressgateway	✓ OK	External load balancer	34.68.11.25:15021	1/1	istio-system	dev-clu...	
<input type="checkbox"/>	istiod	✓ OK	Cluster IP	10.96.6.70	1/1	istio-system	dev-clu...	



The screenshot shows a code editor interface with two main panes. The left pane displays a file structure for a 'CurrencyConversionServiceApplication' under '10-currency-conversion-is'. It includes sections for 'environment' (with 'EnvironmentConfigurationLogger' and 'InstanceInformationService') and 'security'. The right pane shows a YAML configuration snippet and a terminal window.

YAML Configuration (Right Pane):

```
32   value: jaeger-agent.istio-system.svc.cluster.local
33   - name: JAEGER_REPORTER_LOG_SPANS
34   value: "true"
```

Terminal Log (Bottom):

```
m 10-currency-conversion-istio [install]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ 10-currency-conversion-istio ---
[INFO] Installing C:\Users\fahamed\Documents\microservice\kubernetes-crash-course\10-currency-conversion-microser
[INFO] Installing C:\Users\fahamed\Documents\microservice\kubernetes-crash-course\10-currency-conversion-microser
[INFO] Installing C:\Users\fahamed\Documents\microservice\kubernetes-crash-course\10-currency-conversion-microser
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 49.537 s
[INFO] Finished at: 2023-05-22T14:53:00+05:30
[INFO] -----
```

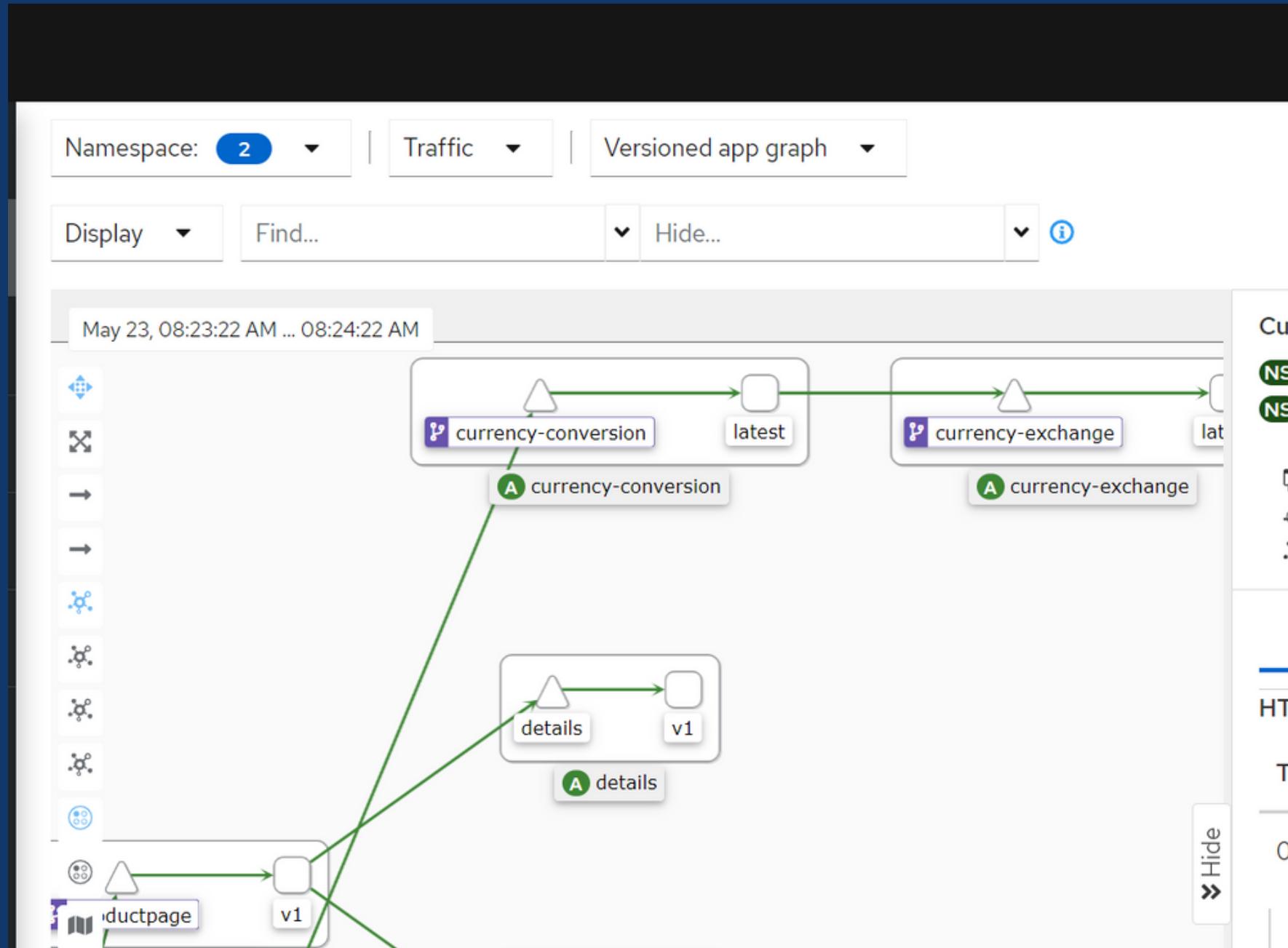
Proof of Work

```

  ▼ 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:
    ▶ image:
      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"
  
```

ubernetes	kubernetes	kubernetes	name	namespac	node_kub	pod	topology_	topolog	topolog
md64	gke-dev-c	linux	13a877173	sample-de	e2-mediu	sample-j	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	f64085414	argocd	e2-mediu	argocd-se	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	ea5fa5545	argocd	e2-mediu	argocd-re	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	d37ad61ac	sample-de	e2-mediu	sample-ap	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	12bd0d77	default	e2-mediu	reviews-v	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	7cccd92e9b	istio-syste	e2-mediu	istio-ingre	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	eaca45fd7	jenkins	e2-mediu	jenkins-7f	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	2f2ac071d	argocd	e2-mediu	argocd-ap	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	3bca1b060	default	e2-mediu	reviews-v	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	68253a52c	default	e2-mediu	ratings-v1	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	b5d6302a0	default	e2-mediu	currency-c	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	1b32b738b	default	e2-mediu	guestbo	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	1a817da54	argocd	e2-mediu	argocd-no	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	2264eac1c	default	e2-mediu	reviews-v	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	b92ec05ac	default	e2-mediu	currency-c	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	b627a61f4	istio-syste	e2-mediu	promethe	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	7895a110f	argocd	e2-mediu	argocd-ap	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	4365d1b4	istio-syste	e2-mediu	istio-egre	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	68d614fb1	argocd	e2-mediu	argocd-re	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	a538a840b	default	e2-mediu	productpa	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	6c332c2a9	argocd	e2-mediu	argocd-de	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	6474e30de	default	e2-mediu	details-v1	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	c7f9941b4	jenkins	e2-mediu	jenkins-7f	us-centr	us-centr	us-centr
md64	gke-dev-c	linux	5bd08068b	kube-syst	e2-mediu	metrics-se	us-centr	us-centr	us-centr

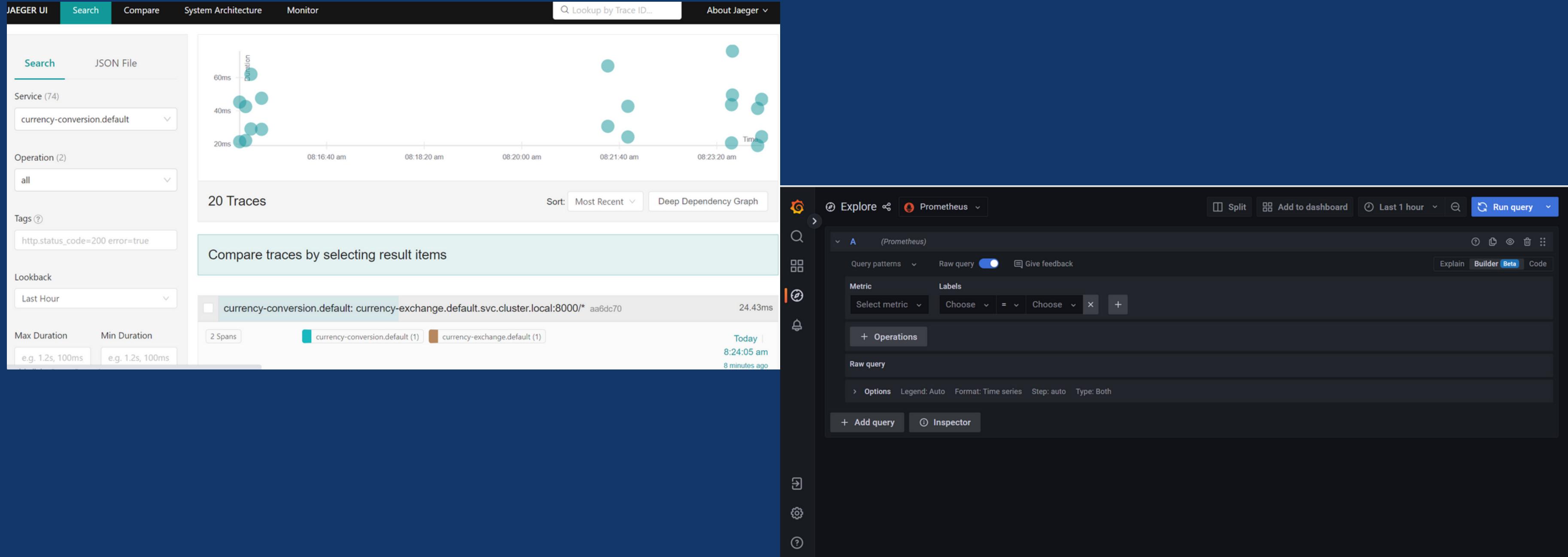
Proof of Work



The screenshot shows a log viewer with the following details:

- Graph Status Help**
- query history:** range_bytes
- Enable autocomplete, Enable highlighting, Enable linter:** checked
- Logs:** Current (with dropdown icons), Ingress, HTTP, Total: 0.40s
- Log Content:** A long JSON log entry for a Kubernetes pod. It includes fields like:
 - beta_kubernetes_io_arch="amd64", beta_kubernetes_io_instance_type="e2-medium", beta_kubernetes_io_os="linux", cloud_google_com_container_runtime="containerd", cloud_google_com_gke_cpu_scaling_level="2", cloud_google_com_gke_logging_variant="DEFAULT",
 - cpu_sps_per_node="110", cloud_google_com_gke_nodepool="node-pool-01", cloud_google_com_gke_os_distribution="cos",
 - logging="standard", cloud_google_com_gke_stack_type="IPV4", cloud_google_com_machine_family="e2", cloud_google_com_private_node_beta_kubernetes_io_region="us-central1", failure_domain_beta_kubernetes_io_zone="us-central1-a", id="/kubepods/burstable/pod0a55",
 - ddda6d9a935359fae8361fffc19f22c98cc7dd0f55473d593", image="docker.io/isurupathumherath/lolc-fusionx-java-project:2.0.0", instance="gke-dev-cluster-01-node-pool-01-e1d9d752-mfg4",
 - bernetes-nodes-cadvisor", kubernetes_io_arch="amd64", kubernetes_io_hostname="gke-dev-cluster-01-node-pool-01-e1d9d752-mfg4",
 - "af20b42aa619205adda6d9a935359fae8361fffc19f22c98cc7dd0f55473d593", namespace="sample-deployment", node_kubernetes_io_insta
 - 01-65fb5f6845-jp5j9", topology_gke_io_zone="us-central1-a", topology_kubernetes_io_region="us-central1", topology_kubernetes_io_zo

Proof of Work



Proof of Work

The screenshot shows a VS Code interface. The Explorer sidebar on the left lists files: package.json, JS index.js, and JS server.js. The index.js file is open in the editor, showing code related to adding data to an Excel sheet. The terminal below shows the execution of the code, resulting in three items being added to an Excel file: Data2, Data3, and Data.

```
worksheet.getRow(1).values = headers;
// Add data rows
for (let i = 0; i < metricValues.length; i++) {
  const rowData = Object.values(metricValues[i]);
  worksheet.addRow(rowData);
}
// Save the workbook
await workbook.xlsx.writeFile('container_cpu_cfs_periods_total.xlsx');
console.log('Data3 added to Excel successfully.');

kubernetes_io_arch: 'amd64',
kubernetes_io_hostname: 'gke-dev-cluster-01-node-pool-01-e1d9d752-wrsj',
kubernetes_io_os: 'linux',
namespace: 'kube-system',
node_kubernetes_io_instance_type: 'e2-medium',
pod: 'tiller-deploy-6fcfc97869-286hk',
topology_gke_io_zone: 'us-central1-a',
topology_kubernetes_io_region: 'us-central',
topology_kubernetes_io_zone: 'us-central1-a',
value: [ 1684786608.57, '9043968' ]
},
... 104 more items
]
33
Data2 added to Excel successfully.
Data3 added to Excel successfully.
Data added to Excel successfully.
```

The screenshot shows a Docker Hub repository page for `fadhljr/currency-exchange`. The repository has no description. It was last pushed 37 minutes ago. It contains two tags: `1.1.0-RELEASE` and `0.0.1-RELEASE`, both of which are images. The `1.1.0-RELEASE` tag was pulled 21 minutes ago and pushed 38 minutes ago. The `0.0.1-RELEASE` tag was pulled a day ago and pushed a day ago.

Tag	OS	Type	Pulled	Pushed
1.1.0-RELEASE		Image	21 minutes ago	38 minutes ago
0.0.1-RELEASE		Image	a day ago	a day ago

[See all](#) [Go to Advanced Image Management](#)

Proof of Work

jupyter research_cpu_cfs_periods_total Last Checkpoint: 6 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) Logout

In []: In []: # for the Linear algebra
import numpy as npy

for the data processing
import pandas as pd

Visualizations
import matplotlib.pyplot as plt

Import our dataset

In [4]: In [4]: #Import the dataset
dataframe = pd.read_excel('container_cpu_cfs_periods_total.xlsx')

In [7]: In [7]: #View the first five rows from the data frame
dataframe.head(5)

Out[7]:

	_name__	beta_kubernetes_io_arch	beta_kubernetes_io_instance_type	beta_kubernetes_io_os	cloud_google_com_gke_boot_disk	cl
0	container_cpu_cfs_periods_total	amd64	e2-medium	linux	pd-balanced	
1	container_cpu_cfs_periods_total	amd64	e2-medium	linux	pd-balanced	
2	container_cpu_cfs_periods_total	amd64	e2-medium	linux	pd-balanced	
3	container_cpu_cfs_periods_total	amd64	e2-medium	linux	pd-balanced	
4	container_cpu_cfs_periods_total	amd64	e2-medium	linux	pd-balanced	

5 rows × 33 columns

Home - Canva Streamlining Software Release Pr Data | Cloud: MongoDB Cloud

Atlas Research Access Manager Billing

CSSE Data Services App Services Charts

+ Create Database

DEPLOYMENT Database PREVIEW test

Database bills

SEARCH deliveryadvices

SERVICES triggers memories

Data API ordernews

Data Federation orders

Search payments

SECURITY products

Backup sites

Database Access suppliers

Network Access terms

Advanced users

memories

test.memories

STORAGE SIZE: 68KB LOGICAL DATA SIZE: 283.94KB TOTAL DOCUMENTS: 204 INDEXES TOTAL SIZE: 24KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes Charts

INSERT DOCUMENT

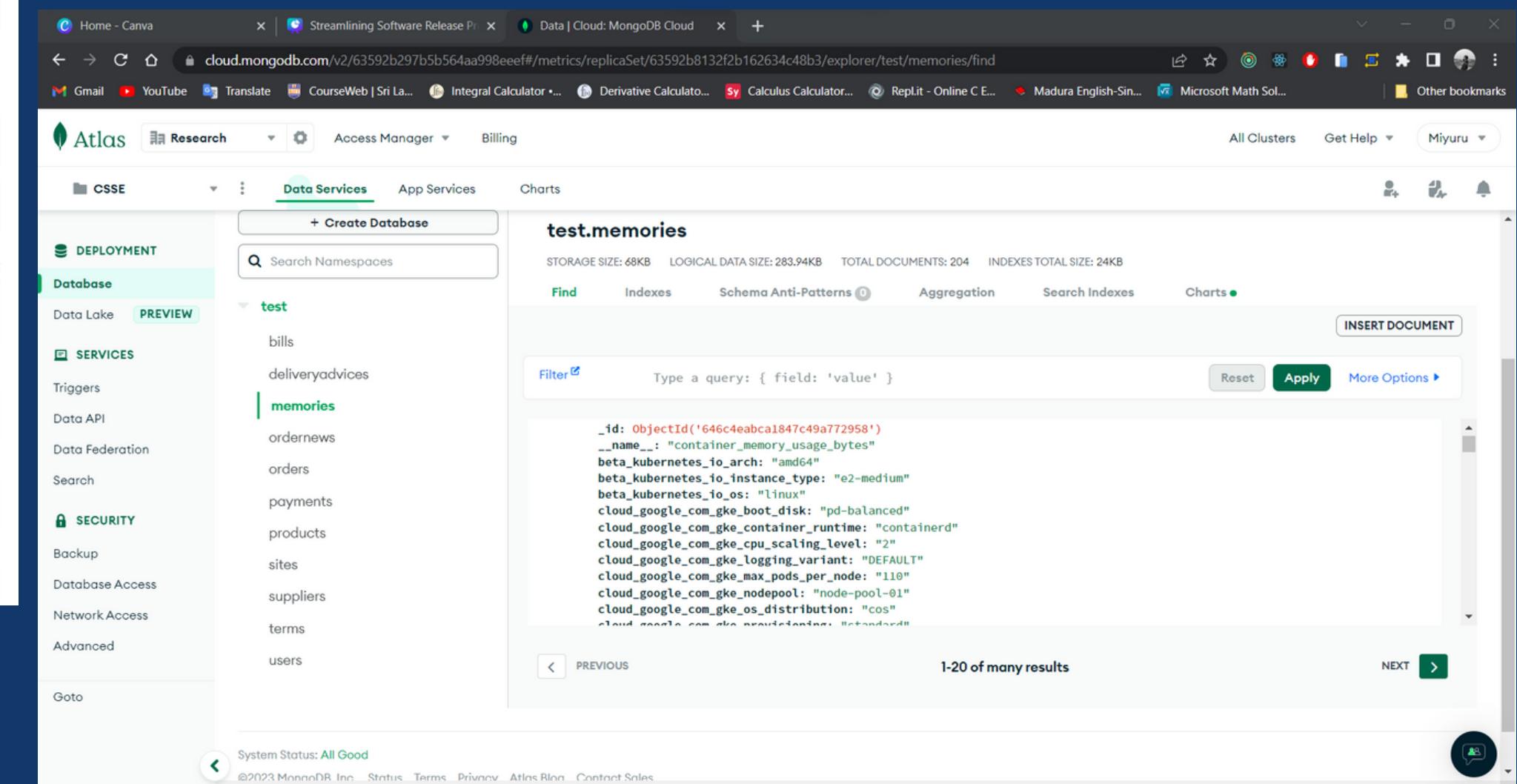
Type a query: { field: 'value' }

Reset Apply More Options

1-20 of many results

PREVIOUS NEXT

System Status: All Good ©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales



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)

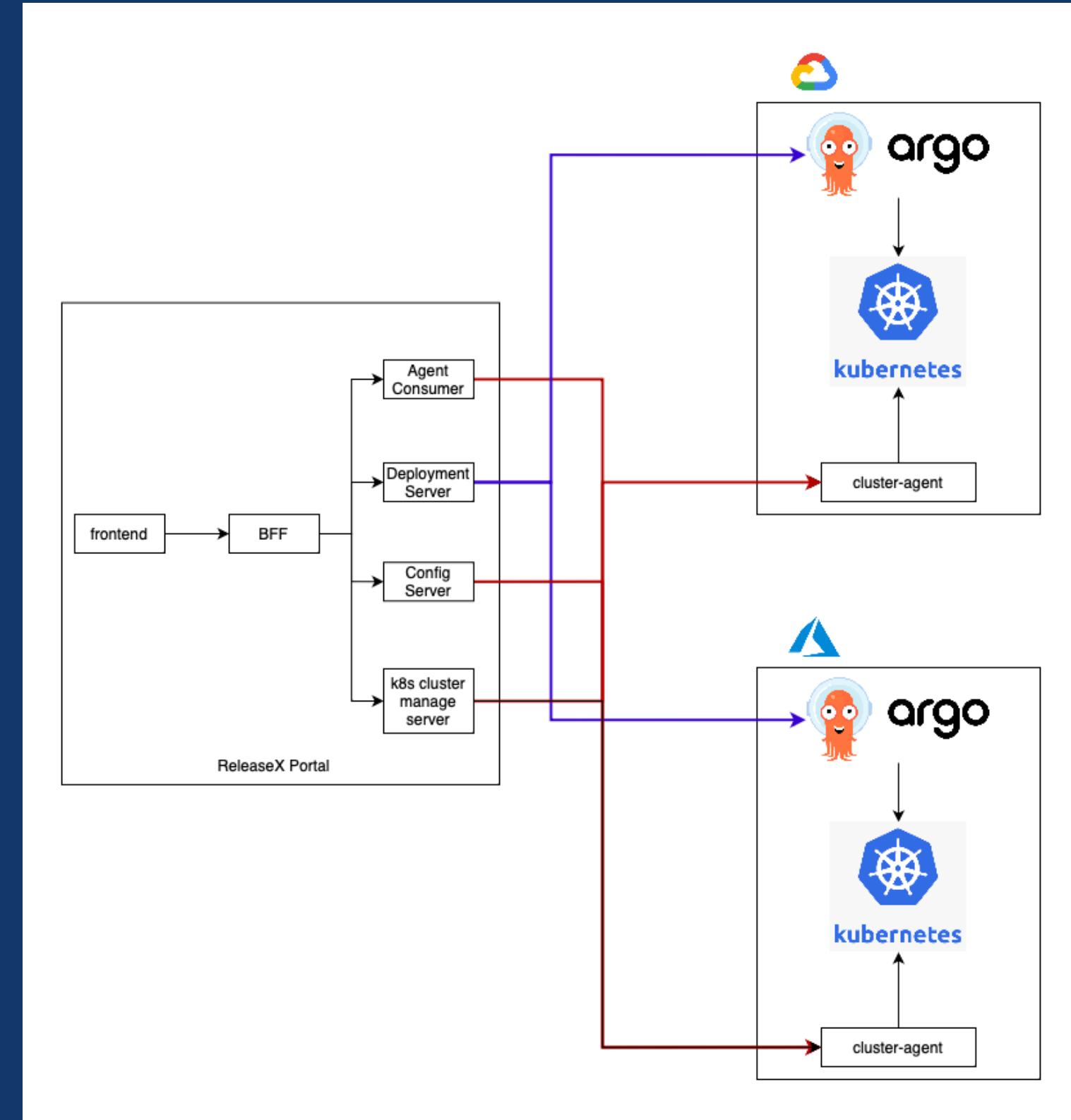
Target Problem

Most of the existing solutions are only providing infrastructure provisioning. Therefore 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 infrastructure provisioning and service level deployments on multi-cloud that accept user configuration customization and identifying incidents of clusters and redeploying on available cloud region or a different cloud vendor 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 assurance of the disaster recovery.

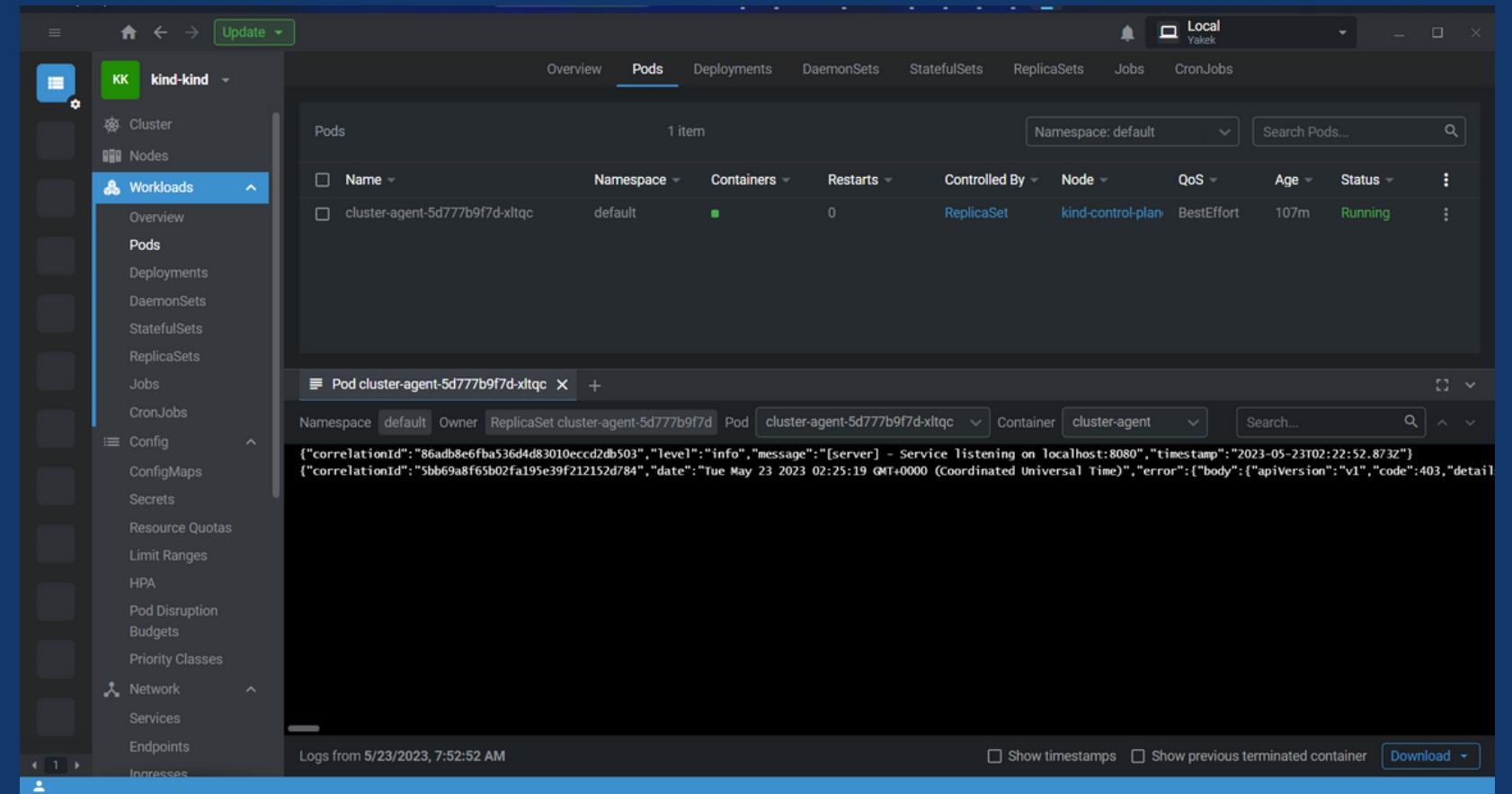
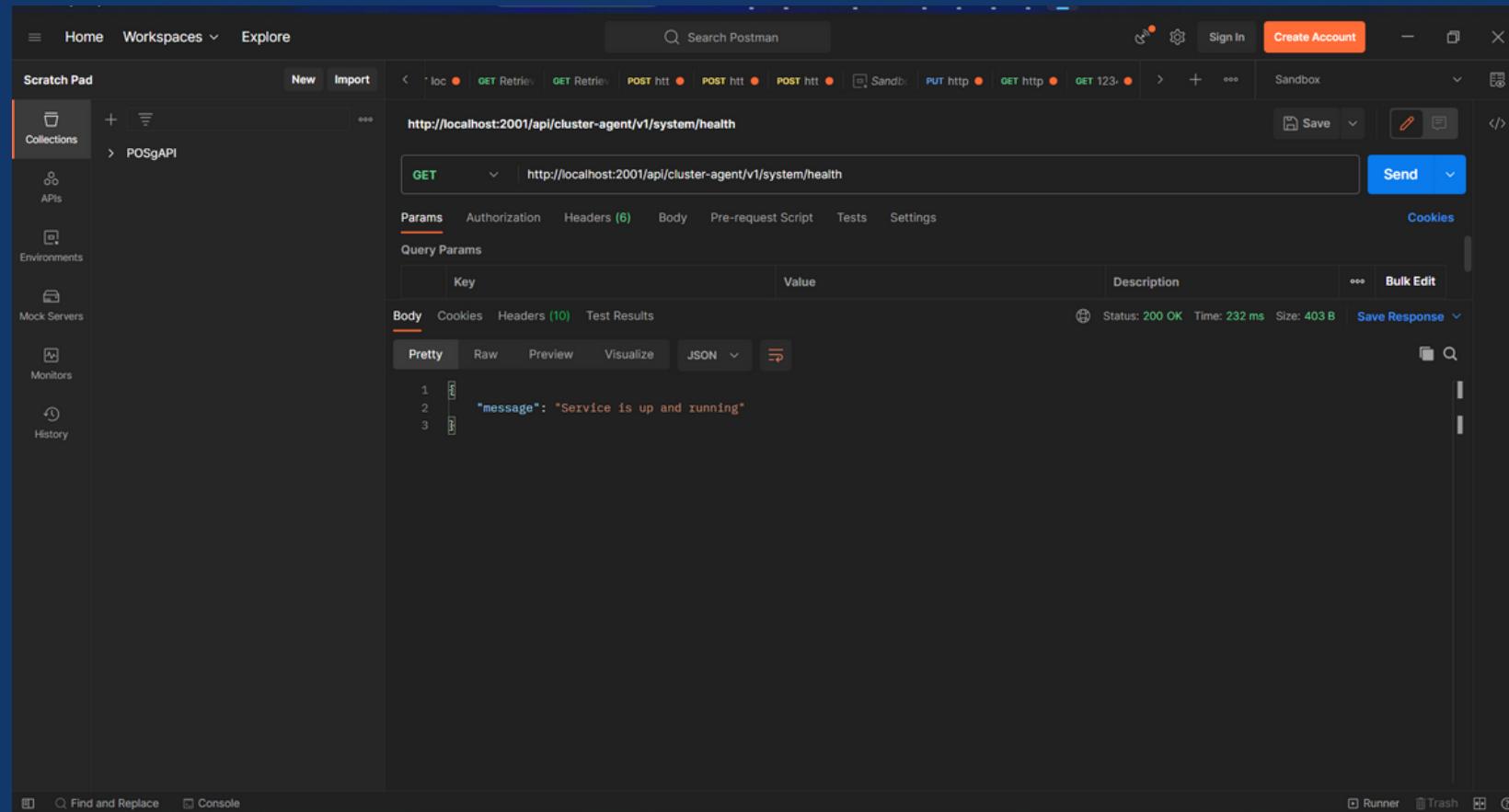
Proof of Work

The screenshot shows the Postman application interface. The left sidebar contains navigation links: Home, Workspaces, Explore, Scratch Pad, Collections, APIs, Environments, Mock Servers, Monitors, and History. The main workspace displays a collection named 'POSGAPI'. A GET request is being made to the URL `http://localhost:2001/api/cluster-agent/v1/cluster/health`. The 'Headers' tab shows six headers. The 'Body' tab displays the JSON response:

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
        "type": "DiskPressure"
    },
    {
        "lastHeartbeatTime": "2023-05-23T02:14:03.000Z",
        "lastTransitionTime": "2023-05-21T07:37:33.000Z",
        "message": "kubelet has sufficient PID available",
        "reason": "KubeletHasSufficientPID",
        "status": "False",
        "type": "PIDPressure"
    },
    {
        "lastHeartbeatTime": "2023-05-23T02:14:03.000Z",
        "lastTransitionTime": "2023-05-21T07:37:54.000Z",
        "message": "kubelet is posting ready status",
        "reason": "KubeletReady",
        "status": "True",
        "type": "Ready"
    }
],
"message": "Get cluster health status success"
```

The status bar at the bottom indicates: Status: 200 OK, Time: 174 ms, Size: 1.27 KB, and Save Response.

Proof of Work



Proof of Work

The screenshot shows the Kind Kubernetes dashboard interface. The left sidebar has a 'Network' section selected, which includes 'Services', 'Endpoints', 'Ingresses', 'Network Policies', and 'Port Forwarding'. The main content area displays a table of services in the 'default' namespace. There are two items listed:

Name	Namespace	Type	Cluster IP	Ports	External IP	Selector	Age	Status
cluster-agent	default	ClusterIP	10.96.12.137	8080/TCP	-	app=cluster-agent	11h	Active
kubernetes	default	ClusterIP	10.96.0.1	443:6443/TCP	-		44h	Active

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

Pricing Table		Free	Silver	Gold	Platinum	Custom
		\$0	\$9.99	\$15.99	\$34.99	TBD
Fully Automated CI Architecture		✓	✓	✓	✓	
Integrated Monitoring Tool		✗	✓	✓	✓	
Automate Resource Allocation		✗	✗	✓	✓	
Multi Cloud Deployment		✗	✗	✗	✓	
Fully Automated and Optimised CD/CD Architecture		✗	✗	✗	✓	Contact to Discuss with ReleaseX

THANK YOU

23-193