

Base of SRE:

In the context of Site Reliability Engineering (SRE), "toil" refers to the repetitive, manual, and automatable tasks that SREs perform, which don't add significant value but consume valuable time and resources, hindering focus on more strategic work.

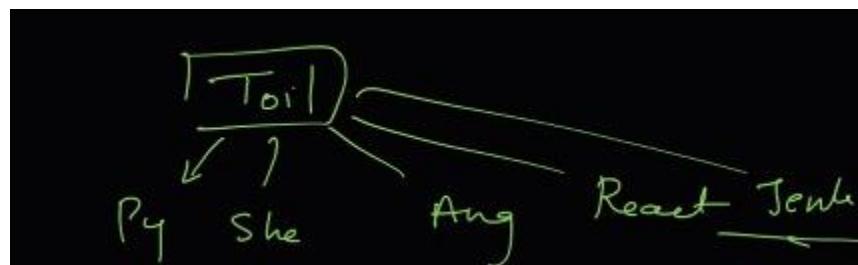
Here's a more detailed explanation:

Definition:

Toil is defined as tasks that are:

- **Manual:** Requiring human intervention rather than automation.
- **Repetitive:** Occurring frequently and following a predictable pattern.
- **Automatable:** Could be easily automated with the right tools and processes.
- **Tactical:** Focused on immediate problems rather than long-term solutions.
- **Devoid of Enduring Value:** Not contributing to the long-term goals of the service or organization.
- **Scales Linearly:** The amount of toil increases proportionally with the growth of the service or system.

Toil can be defined as a work, which performs some work, which is manual, repetitive and can be automated and something which is tactical, scalable, automating using Jenkins ,deployment of code on any machine rather than manual can be done using docker .rather then taking manual input from business we can use react pages to take input.



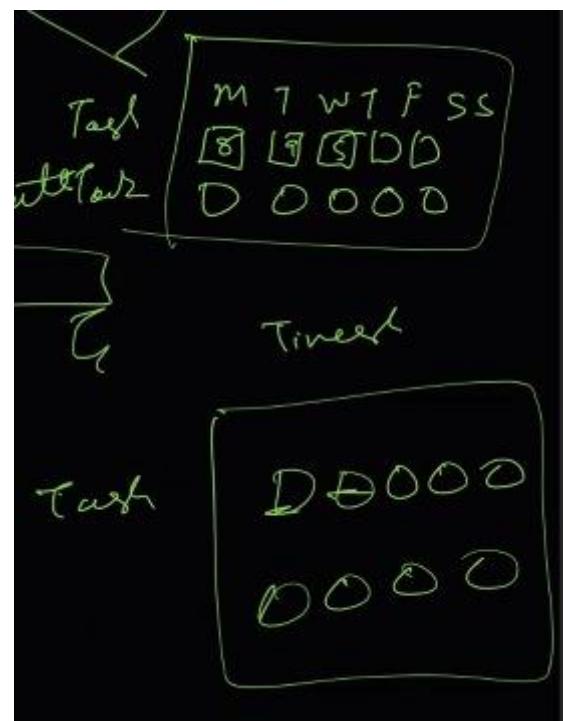
Toil is problematic as:

- it consumes time, which can be used on engineering .
- it scales linearly as service increase .
- decrease job satisfaction and there is a burnout .
- eliminates strategic thinking as daily repetitive work.

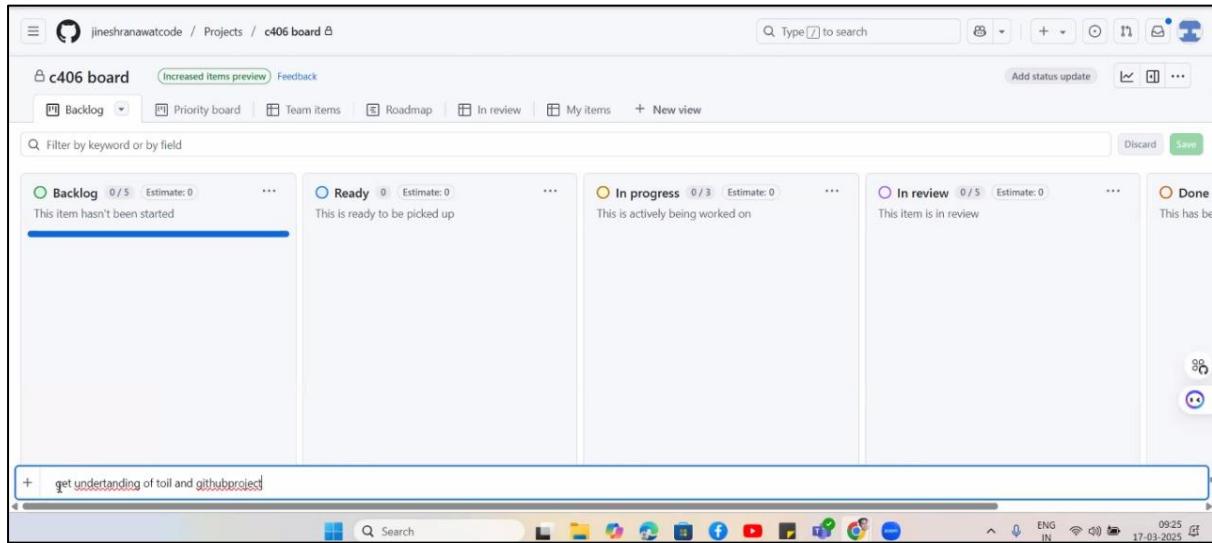
How to eliminate toil:

This is done with help of ticket . when person faces error in website or development they use interface or application named service now and then this ticket is assigned to the SRE guy and we respond via our comments.

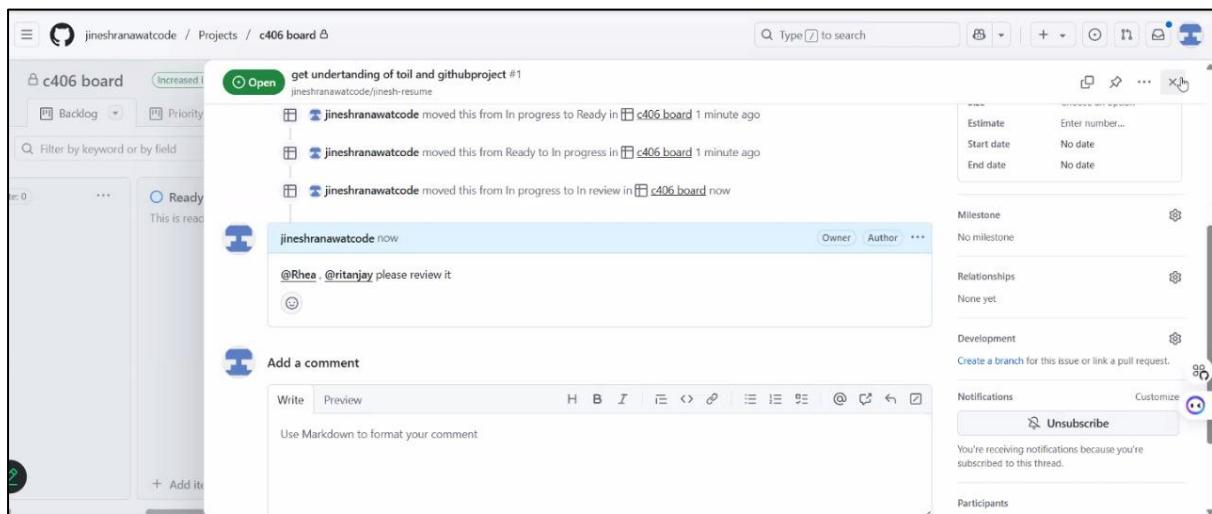
second in timesheet, where we have a timesheet where we post daily how much time we spend on a particular task everyday.



Creating agile sheet using git hub:



we can also generate a review on a specific task



Installation and creation of website using Grafana script:

Script code:

```
#!/bin/bash

# Exit on error
set -e #This option tells the script to exit immediately if any command fails
       (returns a non-zero exit code). This helps prevent cascading errors

echo "===== SIMPLE KUBERNETES MONITORING SETUP ====="

# Reset Minikube if requested
read -p "Do you want to reset Minikube? (y/n): " reset_choice
if [[ "$reset_choice" == "y" ]]; then
    echo "Stopping and deleting Minikube..."
    minikube stop || true
    minikube delete || true

    echo "Starting fresh Minikube cluster..."
    minikube start --driver=docker --cpus=2 --memory=3072
else
    echo "Using existing Minikube cluster..."
fi

# Verify Minikube
echo "Checking Minikube status..."
minikube status # to check minikube cluster status
kubectl get nodes #Lists the nodes in the Kubernetes cluster.

# Create a sample app
echo "Creating a sample application with logging..."
kubectl create namespace sample-app 2>/dev/null || true #Creates a namespace
       called sample-app if it doesn't exist, suppressing any error messages || true
       prevents the script failure if the namespace already exists

# Create a simple app that generates logs
cat <<EOF > sample-app.yaml #Creates a YAML file for a deployment named
       sample-logger in the sample-app namespace.
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-logger
  namespace: sample-app
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    app: sample-logger
template:
  metadata:
    labels:
      app: sample-logger
spec:
  containers:
    - name: logger
      image: busybox
      command: ["/bin/sh", "-c"]
      args:
        - >
          while true; do
            echo "[INFO] Log entry at \$(date)";
            sleep 3;
            echo "[DEBUG] Processing data...";
            sleep 2;
            if [ \$((RANDOM % 10)) -eq 0 ]; then
              echo "[ERROR] Sample error occurred!";
            fi;
            sleep 1;
          done
  resources:
    requests:
      memory: "32Mi"
      cpu: "50m"
    limits:
      memory: "64Mi"
      cpu: "100m"
EOF
```

```
kubectl apply -f sample-app.yaml #Applies the YAML file to create the deployment.
```

```
# Install Prometheus
echo "Installing Prometheus..."

# Clean up monitoring namespace if it exists
kubectl delete namespace monitoring 2>/dev/null || true #Deletes the monitoring namespace if it exists, suppressing any error messages || true prevents the script failure if the namespace doesn't exist
kubectl create namespace monitoring #Creates a new namespace called monitoring.

# Helm repositories are HTTP servers that host packaged Helm charts.
# Think of them as app stores for Kubernetes applications. They store and serve index files
```

```

# containing information about available charts, their versions, and their
locations.

# helm repo add - This command adds a new Helm repository to your local Helm
configuration. This allows your Helm client to access the charts
# hosted in that repository

# Add Helm repos
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts #Adds the Prometheus Helm repository to your
local Helm configuration.
helm repo add grafana https://grafana.github.io/helm-charts #Adds the Grafana
Helm repository to your local Helm configuration.
helm repo update #Updates the Helm repositories to ensure you have the latest
information about available charts.

# Install Prometheus with minimal settings
cat <<EOF > prometheus-values.yaml #Creates a YAML file for Prometheus
configuration.
alertmanager:
  enabled: false
pushgateway:
  enabled: false
server:
  persistentVolume:
    enabled: false
  resources: #Specifies the resources (CPU and memory) that Prometheus will
request and limit.
    requests:
      cpu: 100m
      memory: 256Mi
    limits: #Specifies the maximum resources that Prometheus can use.
      cpu: 200m
      memory: 512Mi
EOF

helm install prometheus prometheus-community/prometheus \
--namespace monitoring \
--values prometheus-values.yaml #Installs the Prometheus application in the
monitoring namespace using the specified values from the prometheus-
values.yaml file.

# Wait for the prometheus server pod to be running
echo "Waiting for Prometheus server pod to start..." #Waits for the Prometheus
server pod to be running.
kubectl wait --for=condition=ready pod --
selector="app.kubernetes.io/name=prometheus,app.kubernetes.io/component=server
" -n monitoring --timeout=120s

```

```
# Install Loki + Promtail
echo "Installing Loki Stack..."
helm install loki grafana/loki-stack \
--namespace monitoring \
--set grafana.enabled=false \
--set prometheus.enabled=false \
--set loki.persistence.enabled=false #Installs the Loki stack in the
monitoring namespace with Grafana disabled, Prometheus disabled, and
persistence disabled.

# Install Grafana with Prometheus and Loki datasources preconfigured
echo "Installing Grafana..."
cat <<EOF > grafana-values.yaml #Creates a YAML file for Grafana
configuration.
persistence:
  enabled: false
adminPassword: admin
datasources:
  datasources.yaml:
    apiVersion: 1
    datasources:
      - name: Prometheus
        type: prometheus
        url: http://prometheus-server.monitoring.svc.cluster.local
        access: proxy
        isDefault: true
      - name: Loki
        type: loki
        url: http://loki.monitoring.svc.cluster.local:3100
        access: proxy
dashboardProviders:
  dashboardproviders.yaml:
    apiVersion: 1
    providers:
      - name: 'default'
        orgId: 1
        folder: ''
        type: file
        disableDeletion: false
        editable: true
        options:
          path: /var/lib/grafana/dashboards/default
dashboards:
  default:
    k8s-pod-logs:
      gnetId: 12019
      revision: 2
```

```

datasource: Prometheus
k8s-simple-dashboard:
  url: https://raw.githubusercontent.com/dotdc/grafana-dashboards-
kubernetes/master/dashboards/k8s-views-global.json #Specifies the URL of the
dashboard to be imported.
EOF

# Install Grafana
helm install grafana grafana/grafana \
--namespace monitoring \
--values grafana-values.yaml #Installs the Grafana application in the
monitoring namespace using the specified values from the grafana-values.yaml
file.

# Wait for Grafana to be ready
echo "Waiting for Grafana to be ready..."
kubectl wait --for=condition=ready pod --
selector="app.kubernetes.io/name=grafana" -n monitoring --timeout=180s #Waits
for the Grafana pod to be running.

# Create port-forward for Grafana
echo "Setting up port-forward for Grafana..."
kubectl port-forward svc/grafana -n monitoring 3000:80 & #Creates a port-
forward to allow you to access the Grafana service from your local machine.
PORT_FORWARD_PID=$! #Stores the process ID of the port-forward in a variable.
# ! stores the process ID of the most recent background process in a variable.

# Create a simple custom dashboard
echo "Waiting 5 seconds for port-forward to stabilize..."
sleep 5 #Waits for 5 seconds to allow the port-forward to stabilize.

echo "Creating custom dashboard for our logs..."
cat <<EOF > dashboard.json #Creates a JSON file for a custom dashboard.
{
  "dashboard": {
    "title": "Application Logs",
    "uid": "app-logs",
    "panels": [
      {
        "id": 1,
        "title": "Sample App Logs",
        "type": "logs",
        "datasource": "Loki",
        "targets": [
          {
            "expr": "{namespace=\"sample-app\", app=\"sample-logger\"}"
#Specifies the query to retrieve logs from the sample-app application. # expr

```

```

is a keyword in Grafana that specifies the query to retrieve data from the
datasource.
        }
    ],
    "gridPos": {"h": 12, "w": 24, "x": 0, "y": 0}, #Specifies the position
of the panel on the dashboard.
    "options": {
        "showTime": true #Specifies that the time should be displayed in the
logs.
    }
},
{
    "id": 2,
    "title": "Error Logs",
    "type": "logs",
    "datasource": "Loki",
    "targets": [
        {
            "expr": "{namespace=\"sample-app\"} |= \"ERROR\"" #Specifies the
query to retrieve error logs from the sample-app application. # expr is a
keyword in Grafana that specifies the query to retrieve data from the
datasource.
        }
    ],
    "gridPos": {"h": 8, "w": 24, "x": 0, "y": 12} #Specifies the position
of the panel on the dashboard.
},
    "refresh": "5s" #Specifies the refresh interval for the dashboard.
},
"folderUid": null,
"overwrite": true
}
EOF

# Import dashboard
curl -s -X POST -H "Content-Type: application/json" -d @dashboard.json
http://admin:admin@localhost:3000/api/dashboards/db #Imports the custom
dashboard into Grafana.

# Get access info
echo ""
echo =====
echo "Setup complete! Access your monitoring dashboards:"
echo ""
echo "Grafana: http://localhost:3000"
echo "Username: admin"
echo "Password: admin"

```

```

echo ""
echo "Your dashboards should now be visible in Grafana"
echo ""
echo "To access Kubernetes Dashboard, run in a separate terminal:"
echo "minikube dashboard"
echo ""
echo "Press Ctrl+C when you're done to terminate port-forwarding"
echo "====="

# Keep script running to maintain port-forward
wait $PORT_FORWARD_PID

```

while installation it may look like:

```

root1@LAPTOP-R268MI6J:~$ mkdir grafana
root1@LAPTOP-R268MI6J:~$ mv simple-grafana-monitoring.sh grafana
root1@LAPTOP-R268MI6J:~$ cd grafana
root1@LAPTOP-R268MI6J:~/grafana$ ls
simple-grafana-monitoring.sh
root1@LAPTOP-R268MI6J:~/grafana$ ./simple-grafana-monitoring.sh
===== SIMPLE KUBERNETES MONITORING SETUP =====
Do you want to reset Minikube? (y/n): y
Stopping and deleting Minikube...
Stopping node "minikube" ...
1 node stopped.
Deleting "minikube" in docker ...
Deleting container "minikube" ...
Removing /home/root1/.minikube/machines/minikube ...
Removed all traces of the "minikube" cluster.
Starting fresh Minikube cluster...
minikube v1.35.0 on Ubuntu 24.04 (amd64)
Using the docker driver based on user configuration
Using Docker driver with root privileges
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.46 ...
Downloading Kubernetes v1.32.0 preload ...
> preloaded-images-k8s-v18-v1...: 23.28 MiB / 333.57 MiB 6.98% 1.57 MiB p|

```

And using this link we would login to Grafana:

```

Grafana: http://localhost:3000
Username: admin
Password: admin

Your dashboards should now be visible in Grafana

To access Kubernetes Dashboard, run in a separate terminal:
minikube dashboard

Press Ctrl+C when you're done to terminate port-forwarding
=====
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000

```

And then lastly we would create a new panel using the following list of commands:

The user is asking for step-by-step instructions to create a Grafana dashboard. Let me provide clear, detailed instructions for manually creating a dashboard in Grafana after the monitoring infrastructure is set up.

I'll break this down into very explicit steps, assuming they have the monitoring stack already running (Prometheus, Loki, and Grafana).

```
# Step-by-Step Guide to Create a Grafana Dashboard
```

Here's a detailed walkthrough to create your own Grafana dashboard:

Step 1: Access Grafana

1. Ensure Grafana is running and accessible

```
bash  
kubectl port-forward svc/grafana -n monitoring 3000:80
```

2. Open your web browser and navigate to:

```
http://localhost:3000
```

3. Log in with the credentials:

- Username: admin
- Password: admin

Step 2: Create a New Dashboard

1. Click the "+" icon in the left sidebar
2. Select "Dashboard" from the dropdown menu
3. Click "Add a new panel"

Step 3: Configure Your First Panel (Logs Panel)

1. In the panel editor, select "Loki" as your data source from the dropdown
2. Enter the following query in the query field:

```
{namespace="sample-app"}
```

3. From the visualization options on the right side, select "Logs"
4. Configure the panel display options:
 - Set panel title to "Application Logs"
 - Under Options, enable "Show time"
 - Set "Sort order" to "Descending" (newest logs first)

5. Click "Apply" in the top-right corner to add this panel to your dashboard

Step 4: Add a Filtered Error Logs Panel

1. Click the "Add panel" button (+ icon) in the top menu
2. Select "Add a new panel"
3. Select "Loki" as your data source
4. Enter this query to show only error logs:

```
{namespace="sample-app"} |= "ERROR"
```

5. Select "Logs" visualization
6. Configure the panel:
 - Set title to "Error Logs"
 - Enable "Show labels"
 - Set appropriate panel size by dragging its corners

7. Click "Apply" to add this panel

Step 5: Add a Metrics Panel

1. Click "Add panel" again
2. Select "Prometheus" as your data source
3. Enter the following query:

```
sum(rate(container_cpu_usage_seconds_total{namespace="sample-app"})[5m]) by (pod)
```

4. Select "Time series" visualization
5. Configure the panel:
 - Set title to "CPU Usage by Pod"
 - Under "Legend" settings, enable "Show values"
 - In Field tab, set Unit to "Percent (0-100)"

6. Click "Apply" to add this panel

Step 6: Add a Log Volume Panel

1. Click "Add panel" once more
2. Select "Loki" as your data source
3. Change the query type from "Range" to "Instant" in the query editor
4. Enter the following query:

```
sum(count_over_time({namespace="sample-app"})[5m]) by (pod_name)
```

5. Select "Time series" visualization
6. Configure the panel:

- Set title to "Log Volume by Pod"
 - Adjust visualization settings as needed
7. Click "Apply" to add this panel
- ## Step 7: Arrange and Resize Panels
1. Drag panels to reposition them on the dashboard
 2. Resize panels by dragging their corners
 3. Create an intuitive layout with related metrics grouped together
- ## Step 8: Configure Dashboard Settings
1. Click the gear icon ⚙ in the top-right to access dashboard settings
 2. Under "General":
 - Name your dashboard "Kubernetes Application Monitoring"
 - Add a description if desired
 - Add tags like "kubernetes" and "application"
 3. Under "Time options":
 - Set the default time range to "Last 15 minutes"
 - Set refresh rate to "5s" for real-time monitoring
 4. Click "Save" to apply settings

- ## Step 9: Save Your Dashboard
1. Click the save icon 🗂 in the top-right corner
 2. Provide a meaningful name for your dashboard
 3. Add a description (optional)
 4. Click "Save"

Step 10: Set Up Dashboard Refresh

1. In the top-right of your dashboard, locate the refresh interval dropdown
2. Set it to "5s" for real-time updates or choose another interval as needed

Your dashboard is now created and will automatically update according to your refresh settings. You can share this dashboard with your team or continue to

To create more sophisticated dashboards, you can also:

- Add template variables to make the dashboard dynamic
- Set up alerts based on thresholds
- Add annotations to mark significant events
- Create different rows to organize panels by category

Common labels: app-sample-logger container-logger job-sample-app/sample-logger namespace-sample-app node_name-minikube pod-sample-logger-67fb64f5d3c

```

|> 2025-03-17 15:09:49.891 {"log": "[INFO] Log entry at Mon Mar 17 09:39:49 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:49.8958993981Z"}
|> 2025-03-17 15:09:46.882 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:39:46.799496417Z"}
|> 2025-03-17 15:09:43.873 {"log": "[INFO] Log entry at Mon Mar 17 09:39:43 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:43.7988216552Z"}
|> 2025-03-17 15:09:40.863 {"log": "[INFO] Log entry at Mon Mar 17 09:39:40 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:40.7988442452Z"}
|> 2025-03-17 15:09:37.853 {"log": "[INFO] Log entry at Mon Mar 17 09:39:37 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:37.7988216552Z"}
|> 2025-03-17 15:09:34.843 {"log": "[INFO] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:39:34.786657Z"}
|> 2025-03-17 15:09:28.828 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:39:28.778426687Z"}
|> 2025-03-17 15:09:25.819 {"log": "[INFO] Log entry at Mon Mar 17 09:39:25 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:25.777842508Z"}
|> 2025-03-17 15:09:22.810 {"log": "[INFO] Log entry at Mon Mar 17 09:39:22 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:22.777842508Z"}
|> 2025-03-17 15:09:19.800 {"log": "[INFO] Log entry at Mon Mar 17 09:39:19 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:19.6781955274Z"}
|> 2025-03-17 15:09:17.788 {"log": "[INFO] Log entry at Mon Mar 17 09:39:17 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:17.6781955274Z"}
|> 2025-03-17 15:09:14.776 {"log": "[INFO] Sample error occurred\\n", "stream": "stdout", "time": "2025-03-17T09:39:14.646791899Z"}
|> 2025-03-17 15:09:11.663 {"log": "[INFO] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:39:11.642431654Z"}
|> 2025-03-17 15:09:08.653 {"log": "[INFO] Log entry at Mon Mar 17 09:39:08 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:39:08.641986154Z"}

```

Queries 1 Transformations 0 Alert 0

Data source Loki Query options MD = auto - 500 Interval = 30s

Query inspector

Logs

Time Unique labels Common labels Wrap lines Pretty JSON Enable log details

Transparent background Panel links Repeat options

Save dashboard

New dashboard

Details Changes 9

Title

Description

Folder

Cancel **Save**

Accessing log files using loki:

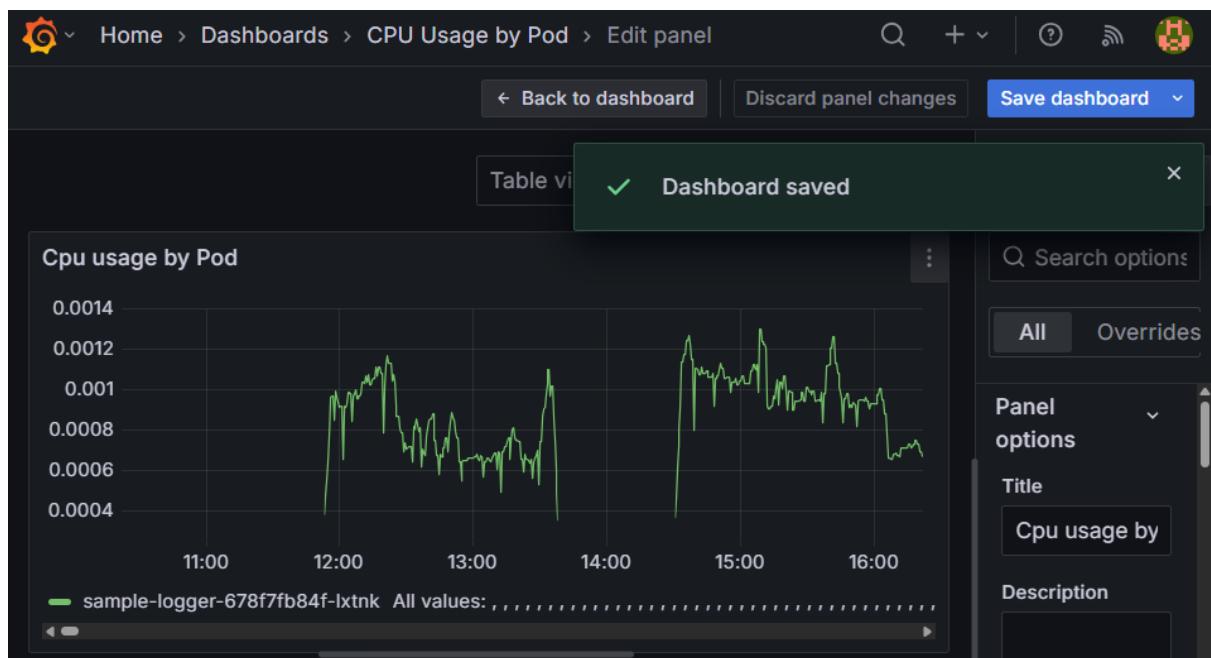
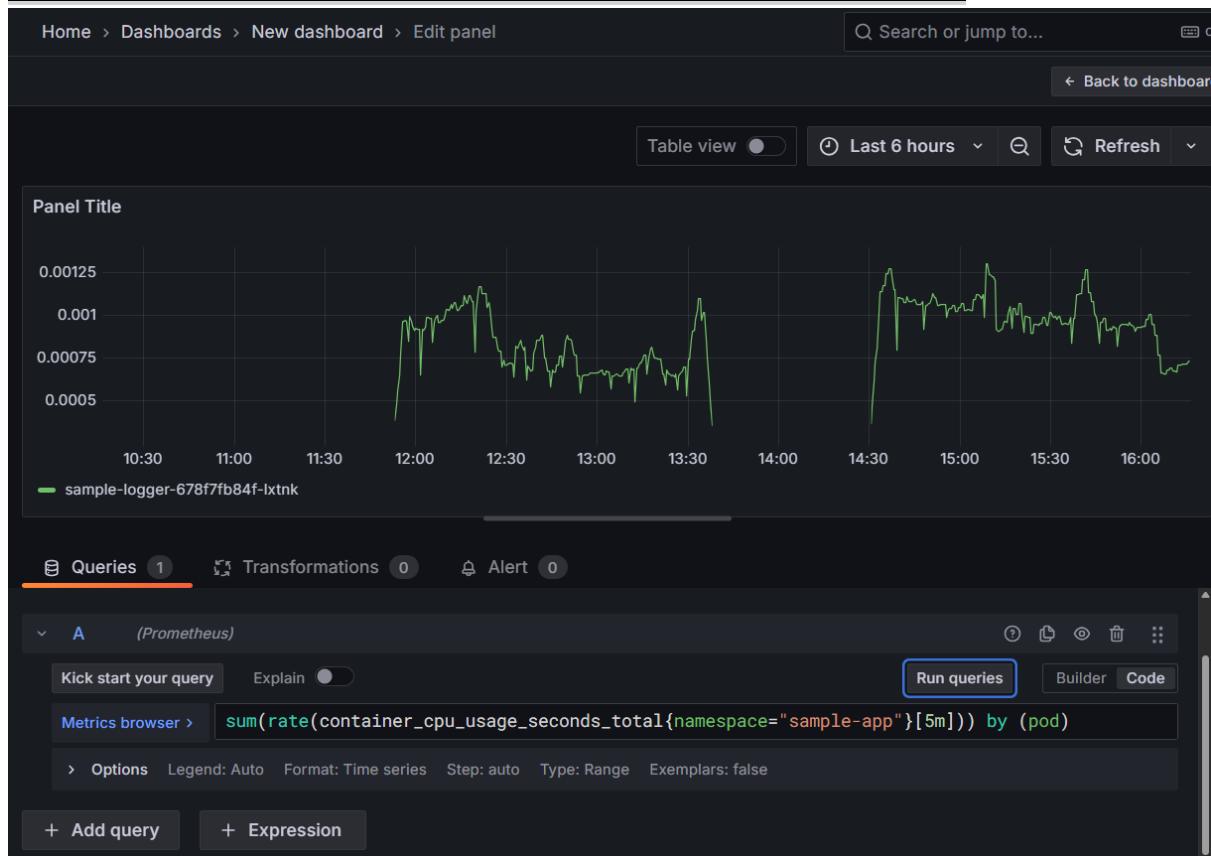
Common labels: app-sample-logger container-logger job-sample-app/sample-logger namespace-sample-app node_name-minikube pod-sample-logger-67fb64f5d3c

```

|> 2025-03-17 15:24:00.531 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:54:00.319448275Z"}
|> 2025-03-17 15:23:57.522 {"log": "[INFO] Log entry at Mon Mar 17 09:53:57 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:53:57Z"}
|> 2025-03-17 15:23:56.521 {"log": "[ERROR] Sample error occurred\\n", "stream": "stdout", "time": "2025-03-17T09:53:56.313365872Z"}
|> 2025-03-17 15:23:54.514 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:53:54.310866076Z"}
|> 2025-03-17 15:23:51.504 {"log": "[INFO] Log entry at Mon Mar 17 09:53:51 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:53:51Z"}
|> 2025-03-17 15:23:48.495 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:53:48.304393393Z"}
|> 2025-03-17 15:23:46.486 {"log": "[INFO] Log entry at Mon Mar 17 09:53:46 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:53:46Z"}
|> 2025-03-17 15:23:43.501 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:53:43.320857696Z"}
|> 2025-03-17 15:23:40.493 {"log": "[INFO] Log entry at Mon Mar 17 09:53:40 UTC 2025\n", "stream": "stdout", "time": "2025-03-17T09:53:40Z"}
|> 2025-03-17 15:23:37.404 {"log": "[DEBUG] Processing data...\\n", "stream": "stdout", "time": "2025-03-17T09:53:37.314166062Z"}

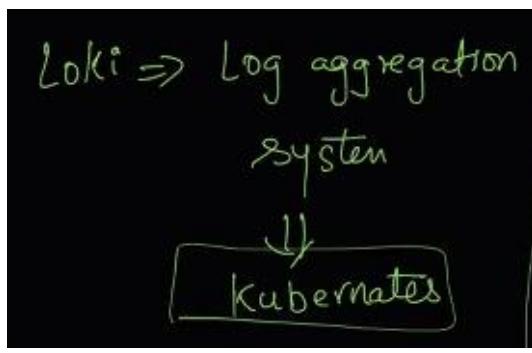
```

Creating and adding a metric panel using Prometheus:

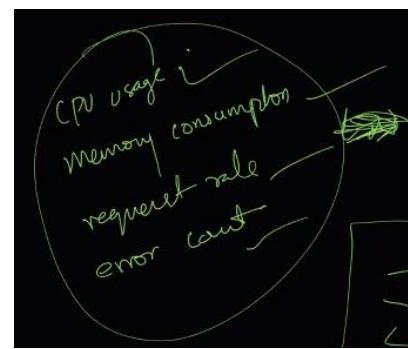


Difference between Loki and Prometheus:

- loki is logs aggregation system for Kubernetes.
- Prometheus is used for tracking all the activities and logs for our application.
- if we want to see exactly everything in details like how it working which process running and all then we use loki.
- And if we want to track details about it then we use Prometheus.



LOKI



PROMETHEUS