

# TOIL, GRAFANA

## Toil:-

Toil refers to manual, repetitive tasks.

## Toil is problematic:-

- Consumes Time → Wastes hours on repetitive tasks.
- Scales Linearly → Becomes harder to manage as workload increases.
- Burnout → Leads to frustration, reducing developer efficiency.

## How to Eliminate Toil:

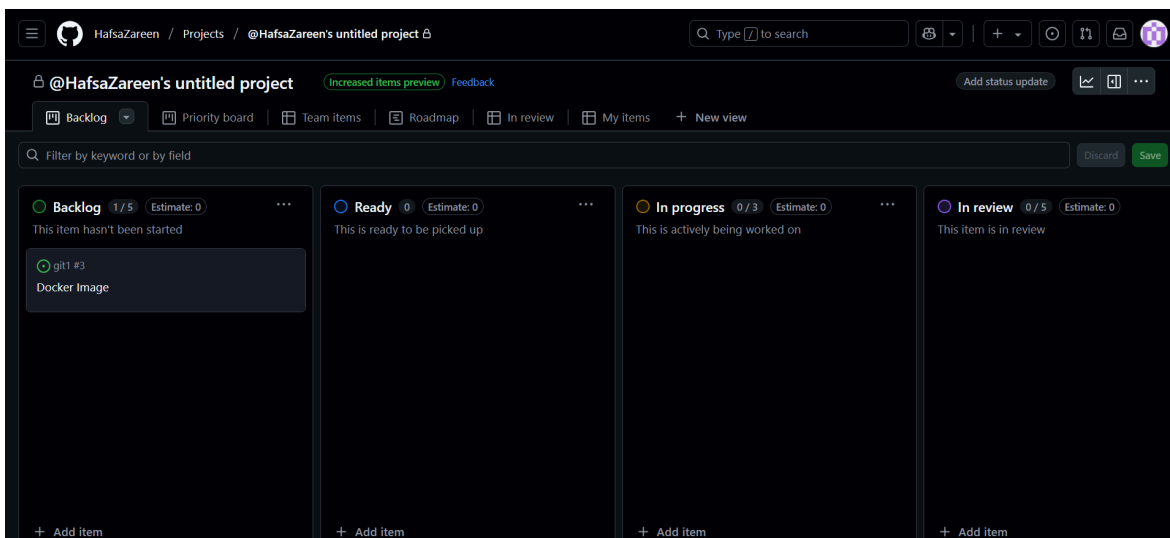
Toil can be managed efficiently using **ticketing systems** and **timesheets**:

1. **Ticketing System (e.g., ServiceNow)** → When a person encounters an error on the website or during development, they log a ticket using a platform like **ServiceNow**. This ticket is then assigned to an **SRE (Site Reliability Engineer)**, who investigates the issue and provides updates through comments.
2. **Timesheets** → We maintain a **timesheet** where we log the amount of time spent on each task daily. This helps track and analyze time-consuming tasks, allowing us to optimize workflows and reduce repetitive toil.

## Agile board using GitHub:-

Click on the **Projects** tab → Select **New Project**.

Choose **"Table"** or **"Board"** view (Board is similar to a Kanban board).



<b><u>Prometheus (Metrics)</u></b>	<b><u>Loki (Logs)</u></b>
Collects numerical metrics (CPU, memory, HTTP requests, etc.)	Stores application logs (errors, debug messages, events)
Triggers alerts when metrics exceed thresholds	Helps debug issues by searching logs
Integrates with Grafana for visualization	Integrates with Grafana for log exploration

## **GRAFANA:-**

Grafana is an open-source monitoring and visualization tool used for analyzing, querying, and displaying real-time metrics from multiple data sources like Prometheus, Loki, InfluxDB, Elasticsearch, MySQL, PostgreSQL, and more.

### **What are Prometheus and Loki?**

Prometheus and Loki are observability tools that help monitor and log system performance in real-time. They are widely used in DevOps, Kubernetes, and cloud monitoring.




---

### **1) Prometheus: Metrics Monitoring & Alerting**

#### What is Prometheus?

Prometheus is an open-source monitoring system designed for collecting metrics (numerical data) from applications and infrastructure. It stores time-series data and supports real-time monitoring, querying, and alerting.

#### Key Features of Prometheus:

-  Collects & Stores Time-Series Data (e.g., CPU, memory, network usage).
-  Powerful Query Language (PromQL) → Used for extracting and analyzing data.
-  Alerting System → Sends notifications based on metric thresholds.

- 🏗️ Works with Kubernetes, Docker, and Cloud services.

### 🔧 How Prometheus Works:

1. Scrapes Metrics → Collects numerical data (CPU, RAM, request rates, etc.) from applications.
  2. Stores Data → Saves it as time-series data (timestamped values).
  3. Queries & Analyzes Data → Uses PromQL to filter and process data.
  4. Triggers Alerts → Notifies teams if something goes wrong.
- 

## 📄 2 Loki: Log Aggregation & Analysis

### 📌 What is Loki?

Loki is a log aggregation system that works similarly to Prometheus but for logs instead of metrics. It helps store, query, and analyze application logs efficiently.

### ✅ Key Features of Loki:

- 📄 Stores & Indexes Logs Efficiently → Unlike traditional logging systems, Loki does not index log content (only metadata).
- 🔍 Query Logs Using LogQL → Similar to PromQL but for logs.
- 📊 Highly Scalable & Cost-Effective → Uses less storage than Elasticsearch.
- 🔗 Integrates with Prometheus, Grafana, Kubernetes, and Docker.

### 🔧 How Loki Works:

1. Collects Logs → From applications (using Promtail, Fluentd, or syslog).
  2. Stores Logs Efficiently → Uses metadata-based indexing (e.g., namespace, pod, container).
  3. Queries Logs → Uses LogQL to filter and analyze logs.
  4. Visualizes in Grafana → Displays logs in dashboards for easy debugging.
-

# Setting Up a Grafana Dashboard for Kubernetes Monitoring(run this file [grafana.sh](#))

## Step 1: Access Grafana

1. Ensure Grafana is running and accessible:  
kubectrl port-forward svc/grafana -n monitoring 3000:80
2. Open your web browser and navigate to:
  - <http://localhost:3000>
3. Log in using the default 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 **Visualization**

## Step 3: Configure a Logs Panel

1. In the panel editor, select **Loki** as the data source.
2. Enter the following query:  
{namespace="sample-app"}
3. Choose **Logs** as the visualization type.
4. Configure the panel:
  - **Title:** "Application Logs"
  - Enable **Show time**
  - Set **Sort order:** "Descending"
5. Click **Apply** to add the panel.

## Step 4: Add a Filtered Error Logs Panel

1. Click **Add panel** (**+** icon) and select **New Panel**.
2. Select **Loki** as the data source.

3. Enter the query to show only error logs:  
`{namespace="sample-app"} |= "ERROR"`
4. Choose **Logs** as the visualization type.
5. Configure the panel:
  - **Title:** "Error Logs"
  - Enable **Show labels**
6. Click **Apply**.

## Step 5: Add a Metrics Panel (CPU Usage)

1. Click **Add panel** and select **New Panel**.
2. Choose **Prometheus** as the data source.
3. Enter the query:  
`sum(rate(container_cpu_usage_seconds_total{namespace="sample-app"}[5m])) by (pod)`
4. Choose **Time series** as the visualization type.
5. Configure the panel:
  - **Title:** "CPU Usage by Pod"
  - Enable **Show values** in Legend settings.
  - Set **Unit:** Percent (0-100)
6. Click **Apply**.

## Step 6: Add a Log Volume Panel

1. Click **Add panel** and select **New Panel**.
2. Choose **Loki** as the data source.
3. Switch query type from **Range** to **Instant**.

Enter the query:

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


- 4.
5. Choose **Time series** as the visualization type.
6. Configure the panel:
  - **Title:** "Log Volume by Pod"
7. Click **Apply**.

## Step 7: Arrange and Resize Panels


1. Drag and reposition panels to create an intuitive layout.

2. Resize panels by dragging their corners.
3. Group related metrics together for better readability.

## Step 8: Configure Dashboard Settings

1. Click the **gear icon**  to access dashboard settings.
2. Under **General** settings:
  - Set **Dashboard Name**: "Kubernetes Application Monitoring1"
  - Add **Tags**: "kubernetes", "monitoring"
3. Under **Time options**:
  - Set default time range to **Last 15 minutes**
  - Set refresh rate to **5s** for real-time monitoring
4. Click **Save**.

## Step 9: Save Your Dashboard

1. Click the **Save icon**  at the top-right corner.
2. Provide a meaningful dashboard name.
3. Optionally, add a description.
4. Click **Save**.

## Step 10: Set Up Auto-Refresh

1. Locate the **refresh interval dropdown** at the top-right of your dashboard.
2. Set the refresh rate to **5s** for real-time updates or choose a suitable interval.

```

hafsa_027@Dell:~$ cd Grafana
hafsa_027@Dell:~/Grafana$ ls
dashboard.json grafana-values.yaml prometheus-values.yaml sample-app.yaml simple-grafana-monitoring.sh
hafsa_027@Dell:~/Grafana$ ./simple-grafana-monitoring.sh
===== SIMPLE KUBERNETES MONITORING SETUP =====
Do you want to reset Minikube? (y/n): n
Using existing Minikube cluster...
Checking Minikube status...
^X^C
hafsa_027@Dell:~/Grafana$ vi simple-grafana-monitoring.sh
hafsa_027@Dell:~/Grafana$ sudo systemctl start grafana-server
[sudo] password for hafsa_027:
Failed to start grafana-server.service: Unit grafana-server.service not found.
hafsa_027@Dell:~/Grafana$ sudo systemctl start grafana-server
Failed to start grafana-server.service: Unit grafana-server.service not found.
hafsa_027@Dell:~/Grafana$ sudo systemctl status grafana-server
Unit grafana-server.service could not be found.
hafsa_027@Dell:~/Grafana$ vi simple-grafana-monitoring.sh
hafsa_027@Dell:~/Grafana$ code .
hafsa_027@Dell:~/Grafana$ cursor .
hafsa_027@Dell:~/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" ...
🔴 Powering off "minikube" via SSH ...
🔴 1 node stopped.
🔥 Deleting "minikube" in docker ...
🔥 Deleting container "minikube" ...

```

```

3. Login with the password from step 1 and the username: admin
#####
##### WARNING: Persistence is disabled!!! You will lose your data when #####
##### the Grafana pod is terminated. #####
#####
Waiting for Grafana to be ready...
pod/grafana-5bfb6885b5-nwt7c condition met
Setting up port-forward for Grafana...
Waiting 5 seconds for port-forward to stabilize...
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Creating custom dashboard for our logs...
Handling connection for 3000
{"folderUid":"","id":3,"slug":"application-logs","status":"success","uid":"app-logs","url":"/d/app-logs/application-logs","version":1}
=====
Setup complete! Access your monitoring dashboards:

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

```

