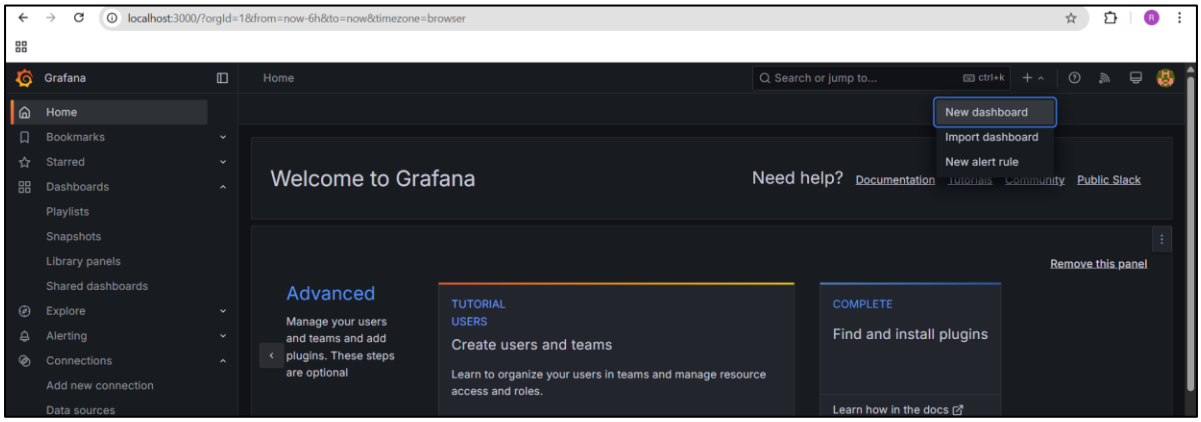
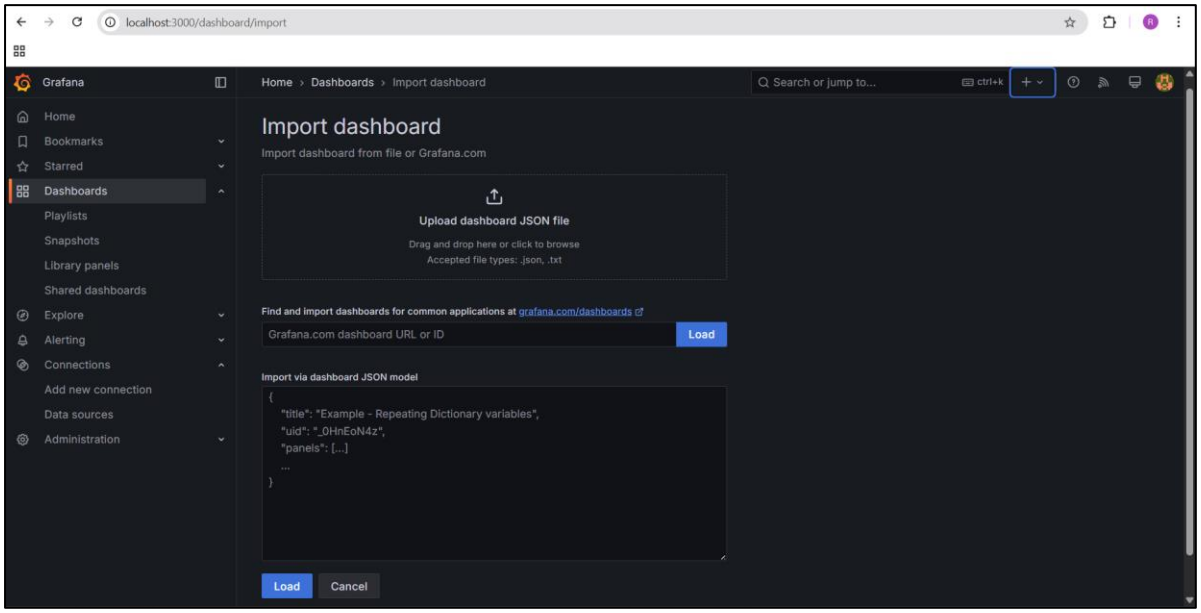
How to import a dashboard in Grafana:

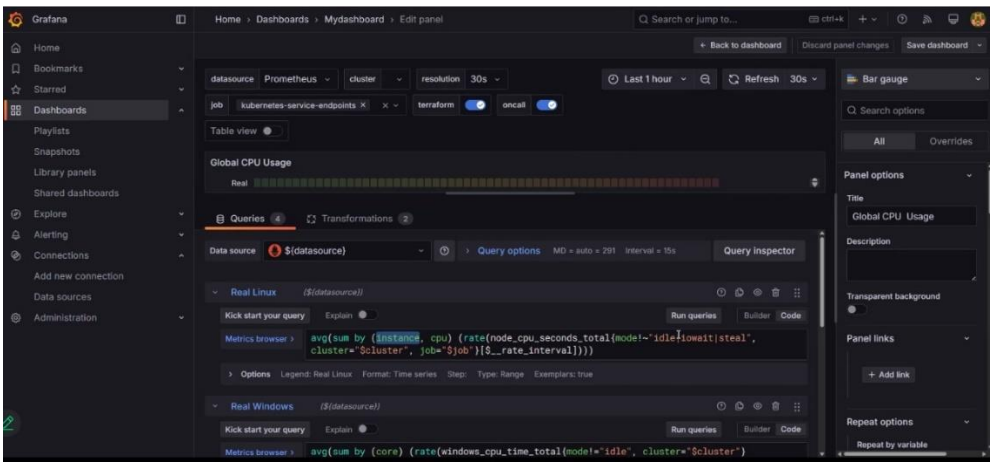
Firstly we click on " + " and click import dashboard and then paste the json code for that:

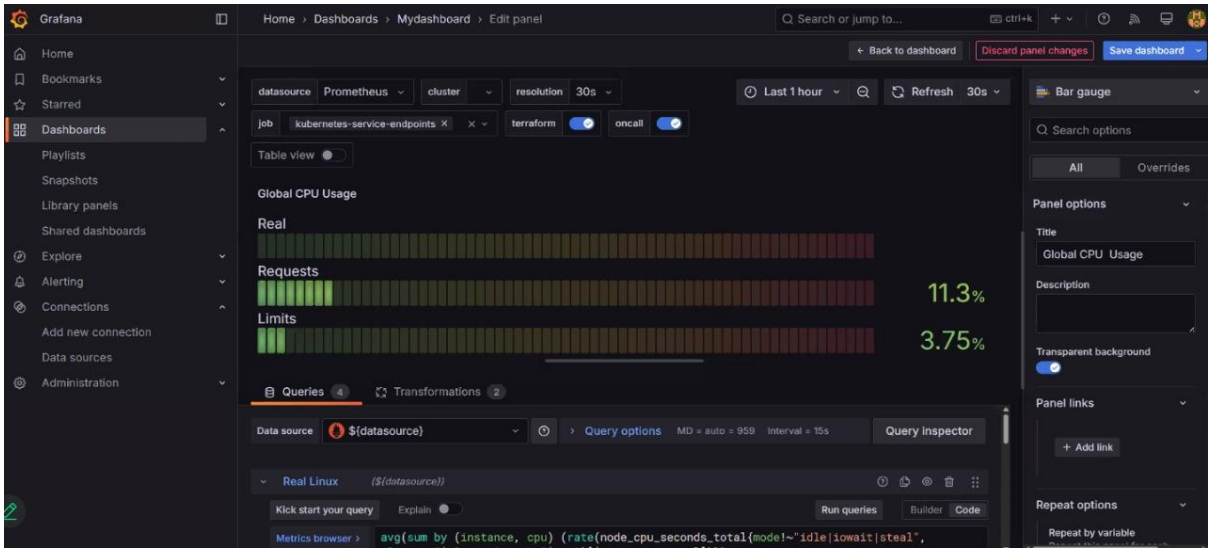




After this simply upload the code and click on load.

And it would show us the following windows or panels in dashboard:




Grafana
Home
Bookmarks
Starred
Dashboards
Playlists
Snapshots
Library panels
Shared dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Home › Dashboards › Mydashboard › Edit panel
Search or jump to...
ctrl+k
Back to dashboard
Discard panel changes
Save dashboard
datasource Prometheus
cluster
resolution 30s
Last 1 hour
Refresh 30s
job kubernetes-service-endpoints
terraform
oncall
Table view
Global CPU Usage
Real
Requests
Limits
11.3%
3.75%
Queries 4
Transformations 2
Data source ${datasource}
Query options MD = auto = 959 Interval = 15s
Query inspector
Real Linux ($datasource)
Kick start your query
Explain
Run queries
Builder Code
Metrics browser
avg(sum by (instance, cpu) (rate(node_cpu_seconds_total{mode!~"idle|iowait|steal",
Bar gauge
Search options
All
Overrides
Panel options
Title
Global CPU Usage
Description
Transparent background
Panel links
+ Add link
Repeat options
Repeat by variable


Home › Dashboards › Mydashboard
Search or jump to...
ctrl+k
Add
Settings
Exit edit
Save dashboard
datasource Prometheus
cluster
resolution 30s
job kubernetes-service-endpoints
Last 1 hour
Refresh 30s
terraform
oncall
Overview
Global CPU Usage
Real
Requests
Limits
11.3%
3.75%
Global RAM Usage
51.72%
11.95%
19.46%
Nodes
1
Nam...
6
Kubernetes Resource Count
Name Min Max
Configmaps 16 18
Running Containers 13 16
Running Pods 12 15
Endpoints 10 11
Services 9 10
Secrets 5 7
Nodes 1 1
09:00
09:30
CPU Usage
Real Requests Limits Total
0 0.900 0.300 8
RAM Usage
Real Requests Limits Total
1.94 GiB 458 MiB 746 MiB 3.74 GiB
Runn...
15
Resources

Theory of prometheous :

```
# Learning Prometheus and Loki Queries for Grafana

I'll guide you through learning Prometheus and Loki queries and how to use
them in Grafana UI step by step.

## Prometheus Basics

Prometheus is a time-series database used for monitoring and alerting. PromQL
(Prometheus Query Language) is its query language.

### Basic PromQL Concepts:

1. Metrics and Labels: Metrics are time-series data points with labels (key-
value pairs)
   - Example: http_requests_total{status="200", method="GET"}

2. Key Functions:
   - rate(): Calculate per-second rate of increase
   - sum(): Aggregate values
   - by(): Group by specific labels
   - avg(), min(), max(): Statistical operations

## Loki Basics

Loki is a log aggregation system designed to work with Grafana. LogQL is its
query language.

### Basic LogQL Concepts:

1. Log Stream Selection: Uses labels to select log streams
   - Example: {app="nginx", env="production"}

2. Log Content Filtering:
   - Example: {app="nginx"} |= "ERROR"

3. Log Parsing and Metrics Extraction:
   - Use | pattern or | json operators to extract fields

## Using Prometheus in Grafana

### Step 1: Add Prometheus Data Source
1. Navigate to Configuration > Data Sources
2. Click "Add data source"
3. Select "Prometheus"
4. Enter your Prometheus server URL (typically http://localhost:9090)
5. Click "Save & Test"
```

```
### Step 2: Create a Dashboard
1. Click "+ Create" > "Dashboard"
2. Click "Add new panel"

### Step 3: Write Prometheus Queries
1. In the panel editor, select "Prometheus" as the data source
2. In the query field, enter your PromQL:

   rate(http_requests_total{job="api-server"}[5m])

3. Use the "Metrics browser" to explore available metrics
4. Apply functions using the Function dropdown

## Using Loki in Grafana

### Step 1: Add Loki Data Source
1. Navigate to Configuration > Data Sources
2. Click "Add data source"
3. Select "Loki"
4. Enter your Loki server URL
5. Click "Save & Test"

### Step 2: Create a Logs Panel
1. In your dashboard, add a new panel
2. Select "Logs" as the visualization

### Step 3: Write Loki Queries
1. Select "Loki" as the data source
2. Enter your LogQL:

   {app="nginx"} |= "ERROR" | json

3. Use the "Log browser" to explore available log streams

## Common Prometheus Query Examples

1. CPU Usage:

   100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) *
100)


2. Memory Usage:

   (node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) /
node_memory_MemTotal_bytes * 100
```

3. HTTP Error Rate:

```
   sum(rate(http_requests_total{status=~"5.."}[5m])) /
sum(rate(http_requests_total[5m])) * 100
```

4. Request Latency (99th percentile):

```
   histogram_quantile(0.99,
sum(rate(http_request_duration_seconds_bucket[5m])) by (le))
```

Common Loki Query Examples

1. Filter for Error Logs:

```
   {app="myapp"} |= "ERROR"
```

2. Count Error Logs by Service:

```
   sum by(service) (count_over_time({app="myapp"} |= "ERROR" [5m]))
```

3. Extract and Count Status Codes:

```
   {app="nginx"} | pattern <_> - - <_> "<_> <_> <_>" <status> <_> |
count_over_time([5m]) by (status)
```

4. Parse JSON Logs:

```
   {app="api"} | json | response_time > 200
```

Practice with these examples in your Grafana instance to get comfortable with both query languages. Start with simple queries and gradually increase complexity as you get more familiar with the syntax.

Steps to create the Grafana dashboard for comprehensive error Monitoring:

```
# Detailed Guide: Creating a Comprehensive Error Monitoring Dashboard in
Grafana
## Step 1: Access Grafana and Start a New Dashboard

1. Ensure port-forwarding is active for Grafana:
   bash
   kubectl port-forward svc/grafana -n monitoring 3000:80


2. Open your browser and navigate to http://localhost:3000

3. Log in with username admin and password admin

4. Create a new dashboard:
   - Click the + icon in the left sidebar
   - Select "Dashboard" from the dropdown menu
   - You'll see an empty dashboard with a welcome message
   - Click "Add visualization"

## Step 2: Create a Container Restart Count Panel

Container restarts are often the first indicator of application errors:

1. Select your data source:
   - Choose "Prometheus" from the data source dropdown

2. Enter the following PromQL query:

   sum by(pod) (kube_pod_container_status_restarts_total{namespace="sample-
app"})


3. Configure the visualization:
   - On the right panel, select "Visualization" and choose "Stat"
   - Under "Panel options," name it "Container Restarts"
   - Under "Value options":
     - Set "Show" to "Calculate"
     - Set "Calculation" to "Last non-null value"
     - Set "Fields" to "All fields"

4. Set up thresholds for visual indicators:
   - Find "Thresholds" in the right panel
   - Click "Add threshold"
   - Set:
     - 0-1: Green (default)
```

```
     - 1-5: Orange (warning)
     - 5+: Red (critical)

5. Under "Text size" set the value to "Large"

6. Click the blue "Apply" button in the top-right corner

## Step 3: Create a Pod Health Status Panel

This panel will show the health status of all your pods:

1. Add a new visualization:
   - Click "Add" in the top menu
   - Select "Visualization"

2. Choose "Prometheus" as your data source

3. Enter this query to track unhealthy pods:

   sum by(pod) (kube_pod_status_phase{namespace="sample-app",
phase!="Running"})


4. Configure visualization:
   - Select "Table" visualization
   - Under "Panel options," name it "Pod Health Issues"
   - Under "Table options":
     - Enable "Show header"

5. Under "Column styles":
   - Add a field override for "Value"
   - Set "Cell display mode" to "Color background"
   - Add thresholds:
     - 0: Green
     - 1: Red

6. Click "Apply" to save this panel

## Step 4: Create a Memory Usage Panel

High memory usage often precedes application errors:

1. Add another visualization and select "Prometheus"

2. Enter this query for memory usage percentage:

   sum by(pod) (container_memory_usage_bytes{namespace="sample-app"}) / sum
by(pod) (container_spec_memory_limit_bytes{namespace="sample-app"}) * 100
```

```
3. Configure visualization:
   - Select "Gauge" as the visualization type
   - Name the panel "Memory Usage (%)"

4. Under "Value options":
   - Set "Show" to "Calculate"
   - Set "Calculation" to "Last *"
   - Set "Fields" to match your pod names

5. Under "Gauge options":
   - Set "Min" to 0
   - Set "Max" to 100

6. Configure thresholds:
   - 0-70: Green
   - 70-85: Orange
   - 85-100: Red

7. Click "Apply" to save

## Step 5: Create a CPU Usage Panel

1. Add a new visualization with "Prometheus" data source

2. Enter this CPU usage query:

   sum by(pod) (rate(container_cpu_usage_seconds_total{namespace="sample-
app"}[5m])) / sum by(pod)
(kube_pod_container_resource_limits_cpu_cores{namespace="sample-app"}) * 100


3. Configure visualization:
   - Select "Gauge" visualization
   - Name it "CPU Usage (%)"

4. Configure value options and thresholds similar to the memory gauge:
   - 0-70: Green
   - 70-90: Orange
   - 90-100: Red

5. Click "Apply" to save

## Step 6: Create an HTTP Error Rate Panel

1. Add a new visualization with "Prometheus" data source
```

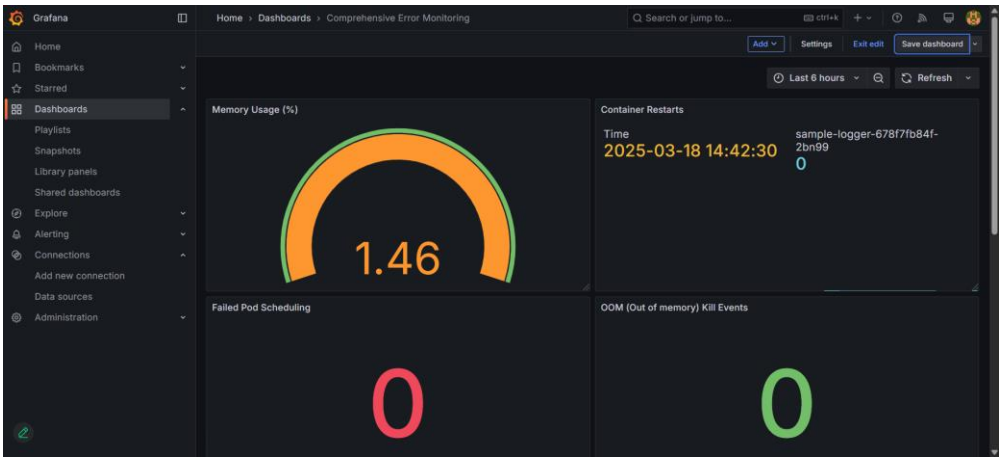
```
2. If you're using a standard metrics exporter, enter this query for HTTP errors:

   sum(rate(http_requests_total{namespace="sample-app", status_code=~"5.."}[5m])) / sum(rate(http_requests_total{namespace="sample-app"}[5m])) * 100

   Note: This assumes your application exposes standard Prometheus metrics. You may need to adjust the metric names to match your specific application.

3. Configure visualization:
   - Select "Time series" visualization
   - Name it "HTTP Error Rate (%)"

4. Under "Standard options":
   - Set "Unit" to "Percent (0-100)"

5. Under "Thresholds":
   - 0-1: Green
   - 1-5: Orange
   - 5+: Red

6. Under "Graph styles":
   - Set "Line width" to 2
   - Set "Fill opacity" to 20
   - Set "Line interpolation" to "Smooth"

7. Click "Apply" to save

## Step 7: Create a Slow Response Panel

1. Add a new visualization with "Prometheus" data source

2. Enter this query for response time:

   histogram_quantile(0.95, sum by(le) (rate(http_request_duration_seconds_bucket{namespace="sample-app"}[5m])))

   Note: Adjust metric names to match your application's exposed metrics.

3. Configure visualization:
   - Select "Time series" visualization
   - Name it "95th Percentile Response Time"

4. Under "Standard options":
   - Set "Unit" to "Seconds"
```

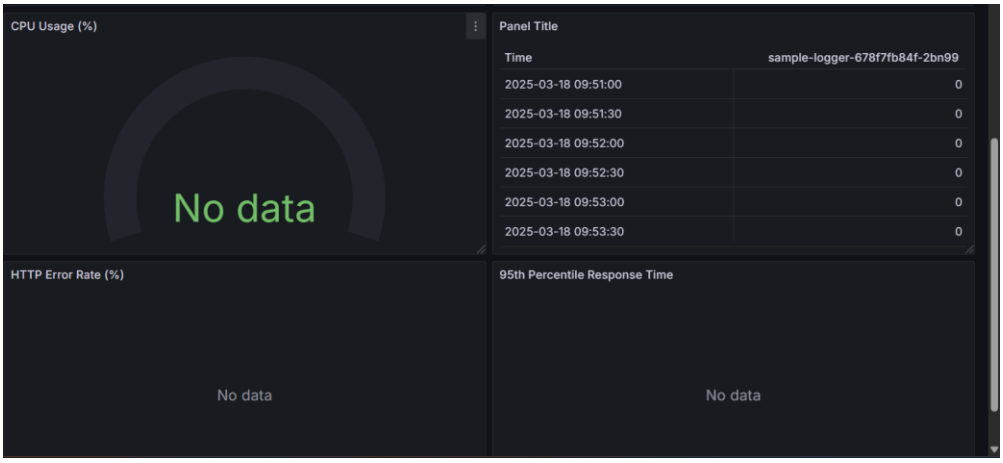
```
5. Under "Thresholds":
   - 0-0.5: Green
   - 0.5-1: Orange
   - 1+: Red

6. Click "Apply" to save

## Step 8: Create a Failed Pod Scheduling Panel

1. Add a new visualization with "Prometheus" data source

2. Enter this query:

   sum(kube_pod_status_scheduled{namespace="sample-app", condition="false"})


3. Configure visualization:
   - Select "Stat" visualization
   - Name it "Failed Pod Scheduling"

4. Under "Value options":
   - Set "Show" to "Calculate"
   - Set "Calculation" to "Last *"

5. Under "Thresholds":
   - 0: Green
   - 1+: Red

6. Click "Apply" to save

## Step 9: Create OOM (Out of Memory) Kill Counter

1. Add a new visualization with "Prometheus" data source

2. Enter this query:

   sum(container_oom_events_total{namespace="sample-app"})


3. Configure visualization:
   - Select "Stat" visualization
   - Name it "OOM Kill Events"

4. Under "Value options":
   - Set "Show" to "Calculate"
   - Set "Calculation" to "Last *"
```

```
5. Under "Thresholds":
   - 0: Green
   - 1+: Red

6. Click "Apply" to save

## Step 10: Arrange Your Dashboard

1. Resize and arrange all panels in a logical order:
   - Place the most critical error indicators at the top
   - Group related metrics together
   - Make sure the most important panels are prominent

2. For a suggested layout:
   - Top row: Container Restarts, Pod Health Issues, OOM Kill Events
   - Middle row: Memory Usage, CPU Usage
   - Bottom row: HTTP Error Rate, Response Time, Failed Pod Scheduling

3. Save your dashboard:
   - Click the save icon in the top-right corner
   - Name it "Comprehensive Error Monitoring"
   - Add a description if desired
   - Add tags like "errors", "monitoring", "kubernetes"
   - Click "Save"

## Step 11: Add Dashboard Settings

1. Click the gear icon in the top-right to access dashboard settings

2. Under "General":
   - Confirm your dashboard title
   - Add a detailed description

3. Under "Variables" (to make your dashboard reusable):
   - Add a variable named "namespace"
   - Set Type to "Query"
   - Set Data source to "Prometheus"
   - Query: label_values(kube_namespace_labels, namespace)
   - Under Selection options, enable "Multi-value" and "Include All option"
   - Click "Update" then "Apply"

4. Under "Time options":
   - Set auto-refresh to "On"
   - Set default refresh interval to "10s"

5. Save your dashboard again

## Step 12: Final Verification and Testing
```

```
1. Verify your dashboard shows meaningful data:
   - Check that all panels are populating with data
   - If any panel shows "No data," revisit the query to ensure it matches your
metrics

2. Test error detection:
   - If possible, trigger a test error in your application
   - Verify it appears in your dashboard
   - Check the time it takes for the error to be reflected

3. Set up alerts (optional advanced step):
   - Click on a critical panel like "Container Restarts"
   - Go to Alert tab
   - Create a new alert rule based on thresholds
   - Set notification channels if configured

Your comprehensive error monitoring dashboard is now complete. It provides
multiple perspectives on potential errors and issues in your Kubernetes
applications, helping you identify and resolve problems quickly.
```

Output:





AGILE MODEL FOR OUR PROJECT: