

Grafana Monitoring Setup Guide

Overview

This guide shows you how to set up Grafana to monitor your Heart Disease Prediction API with beautiful real-time dashboards.

What You'll See in the Dashboard

- **Prediction Request Rate:** Real-time requests per second
 - **Total Predictions:** Cumulative prediction count
 - **Predictions by Class:** Distribution of disease vs no-disease predictions
 - **Average Prediction Time:** API response performance
 - **API Health Status:** Service availability
 - **CPU & Memory Usage:** Resource utilization
 - **HTTP Requests:** Traffic analysis by method
-

Method 1: Using Docker Compose (Easiest)

Step 1: Start All Services

```
cd
/Users/v0s01jh/Documents/BITS/ML0psExperimentalLearning_Assignment_1_Group
_81/heart-disease-mlops

# Start API, Prometheus, and Grafana
docker-compose up -d

# Check if all services are running
docker-compose ps
```

Step 2: Access Grafana

1. Open browser: **http://localhost:3000**
2. Login credentials:
 - Username: **admin**
 - Password: **admin**
3. Dashboard will be auto-loaded: **"Heart Disease Prediction API Dashboard"**

Step 3: Generate Some Traffic

```
# Run multiple predictions to see metrics
for i in {1..20}; do
  curl -X POST http://localhost:8000/predict \
    -H "Content-Type: application/json" \
```

```
-d @sample_input.json  
sleep 1  
done
```

Step 4: View the Dashboard

- Dashboard auto-refreshes every 5 seconds
- Time range: Last 15 minutes (adjustable)
- All panels will show live data

Method 2: Local Installation (Without Docker)

Prerequisites

```
# Install Prometheus  
brew install prometheus  
  
# Install Grafana  
brew install grafana  
  
# Or download from:  
# Prometheus: https://prometheus.io/download/  
# Grafana: https://grafana.com/grafana/download
```

Step 1: Configure Prometheus

The Prometheus configuration is already at:

```
deployment/prometheus/prometheus.yml
```

Start Prometheus:

```
cd  
/Users/v0s01jh/Documents/BITS/ML0psExperimentalLearning_Assignment_1_Group  
_81/heart-disease-mlops  
  
# Start Prometheus with our config  
prometheus --config.file=deployment/prometheus/prometheus.yml --  
web.listen-address=:9090
```

Keep this terminal running!

Step 2: Start Your API

In a **new terminal**:

```
cd
/Users/v0s01jh/Documents/BITS/ML0psExperimentalLearning_Assignment_1_Group
_81/heart-disease-mlops
source venv/bin/activate
PYTHONPATH=. python src/app.py
```

Step 3: Verify Prometheus is Scraping Metrics

Open browser: **http://localhost:9090**

Test a query:

- Type: `prediction_requests_total`
- Click "Execute"
- You should see metrics data

Step 4: Start Grafana

In a **new terminal**:

```
# Start Grafana
grafana-server --homepath=/opt/homebrew/share/grafana --
config=/opt/homebrew/etc/grafana/grafana.ini

# Or if installed differently:
brew services start grafana
```

Step 5: Configure Grafana

1. **Open Grafana:** `http://localhost:3000`

2. **Login:**

- Default username: `admin`
- Default password: `admin`
- Change password when prompted (or skip)

3. **Add Prometheus Data Source:**

- Go to: Configuration (⚙️) → Data Sources
- Click "Add data source"
- Select "Prometheus"
- URL: `http://localhost:9090`
- Click "Save & Test" (should show green checkmark)

4. **Import Dashboard:**

- Go to: Create (+) → Import
 - Click "Upload JSON file"
 - Select: `deployment/grafana/dashboard.json`
 - Click "Load"
 - Select Prometheus data source
 - Click "Import"
-

Dashboard Panels Explained

1. Prediction Request Rate

- Shows requests per second over time
- Useful for: Understanding traffic patterns
- Formula: `rate(prediction_requests_total[5m])`

2. Total Prediction Requests

- Cumulative count of all predictions
- Color thresholds:
 - Green: 0-1000 requests
 - Yellow: 1000-5000 requests
 - Red: 5000+ requests

3. Predictions by Class

- Split view: Disease vs No-Disease predictions
- Tracks: `predictions_by_class_total{class="0"}` and `class="1"`

4. Average Prediction Time

- Mean response time in seconds
- Lower is better for API performance
- Alert if > 1 second

5. API Health Status

- Binary indicator: Up (1) or Down (0)
- Auto-checks: `http://localhost:8000/health`

6. CPU Usage

- Percentage of CPU being used
- Thresholds:
 - Green: < 50%
 - Yellow: 50-80%
 - Red: > 80%

7. Memory Usage

- RAM consumed by the API process
- Measured in bytes
- Monitor for memory leaks

8. HTTP Requests by Method

- Bar chart showing GET, POST, etc.
- Useful for API usage patterns

Testing the Dashboard

Generate Realistic Traffic

```
# Script to generate varied traffic
cd
/Users/v0s01jh/Documents/BITS/ML0psExperimentalLearning_Assignment_1_Group
_81/heart-disease-mlops

# Health checks
for i in {1..10}; do curl http://localhost:8000/health; sleep 1; done

# Predictions with variations
for i in {1..30}; do
  AGE=$((50 + RANDOM % 30))
  curl -X POST http://localhost:8000/predict \
    -H "Content-Type: application/json" \
    -d '{"age\: $AGE, \"sex\: 1, \"cp\: 3, \"trestbps\: 145,
\"chol\: 233, \"fbs\: 1, \"restecg\: 0, \"thalach\: 150, \"exang\: 0,
\"oldpeak\: 2.3, \"slope\: 3, \"ca\: 0, \"thal\: 6}"
  sleep 2
done

# Metrics endpoint
curl http://localhost:8000/metrics
```

Customizing the Dashboard

Add a New Panel

1. Click "Add panel" → "Add new panel"
2. Select data source: Prometheus
3. Enter query (examples below)
4. Configure visualization type
5. Click "Apply"

Useful Prometheus Queries

```
# Total predictions
prediction_requests_total

# Prediction rate (per second)
rate(prediction_requests_total[5m])

# Predictions by outcome
predictions_by_class_total

# Average prediction time
rate(prediction_duration_seconds_sum[5m]) /
rate(prediction_duration_seconds_count[5m])

# 95th percentile prediction time
histogram_quantile(0.95, rate(prediction_duration_seconds_bucket[5m]))

# Memory usage
process_resident_memory_bytes

# CPU usage
rate(process_cpu_seconds_total[5m])

# API availability
up{job="fastapi"}
```

Troubleshooting

Dashboard shows "No Data"

Check 1: Is the API running?

```
curl http://localhost:8000/health
```

Check 2: Is Prometheus scraping?

- Open: <http://localhost:9090/targets>
- Status should be "UP" for all targets

Check 3: Are metrics being exposed?

```
curl http://localhost:8000/metrics | grep prediction
```

Check 4: Prometheus data source connected?

- Grafana → Configuration → Data Sources
- Click "Prometheus" → "Test" button

Grafana not starting

```
# Check if port 3000 is in use
lsof -i :3000

# Kill any conflicting process
kill -9 <PID>

# Restart Grafana
brew services restart grafana
```

Prometheus not scraping

Check configuration:

```
cat deployment/prometheus/prometheus.yml
```

Should include:

```
scrape_configs:
  - job_name: 'fastapi'
    static_configs:
      - targets: ['localhost:8000']
    metrics_path: '/metrics'
```

Screenshots for Assignment

Take screenshots of:

1. **Dashboard Overview** - Full dashboard with all panels showing data
2. **Prediction Rate Graph** - Close-up of request rate over time
3. **Predictions by Class** - Disease distribution chart
4. **Health Gauges** - CPU, Memory, and API status
5. **Prometheus Targets** - <http://localhost:9090/targets> showing UP status
6. **Raw Metrics** - Output from <http://localhost:8000/metrics>

Quick Start (TL;DR)

```
# Terminal 1: Start Prometheus
cd heart-disease-mlops
prometheus --config.file=deployment/prometheus/prometheus.yml

# Terminal 2: Start API
```

```
source venv/bin/activate
PYTHONPATH=. python src/app.py

# Terminal 3: Start Grafana
brew services start grafana

# Browser 1: Grafana Dashboard
open http://localhost:3000
# Login: admin/admin
# Import: deployment/grafana/dashboard.json

# Terminal 4: Generate traffic
for i in {1..50}; do curl -X POST http://localhost:8000/predict -H
"Content-Type: application/json" -d @sample_input.json; sleep 2; done
```

URLs Reference

Service	URL	Purpose
API	http://localhost:8000	Main prediction service
API Docs	http://localhost:8000/docs	Swagger UI
Metrics	http://localhost:8000/metrics	Prometheus metrics
Prometheus	http://localhost:9090	Metrics database
Grafana	http://localhost:3000	Visualization dashboard
MLflow	http://localhost:5000	Experiment tracking

Additional Resources

- [Prometheus Documentation](#)
- [Grafana Documentation](#)
- [Prometheus Python Client](#)
- [Dashboard JSON Schema](#)

🌟 Your Grafana dashboard is now ready to provide real-time insights into your ML API!