สารบัญ

Prometheus and Grafana

1. **Tool นี้มันคืออะไร ?**

Prometheus and Grafana

Prometheus เป็นเครื่องมือโอเพนซอร์สที่ออกแบบมาเพื่อมอนิเตอร์และเก็บข้อมูลจากระบบซอฟต์แวร์ต่างๆ โดยเฉพาะในสภาพแวดล้อมที่มีการเปลี่ยนแปลงอย่างรวดเร็วเช่น Kubernetes หรือระบบคลาวด์ Prometheus ถูกพัฒนาโดยบริษัท SoundCloud ในปี 2012 และได้รับการยอมรับอย่างกว้างขวางในชุมชน DevOps

Grafana เป็นเครื่องมือสำหรับการแสดงผลข้อมูลที่มีลักษณะแบบเรียลไทม์ และการสร้างแผนภูมิและพาเนลต่างๆ เพื่อการทำข้อมูลให้มีประโยชน์ในการตัดสินใจและการวิเคราะห์ ซึ่งสามารถเชื่อมต่อกับหลายแหล่งข้อมูลและฐานข้อมูลต่างๆ

คุณสมบัติหลักของ Prometheus

Data Model แบบ Time Series

Prometheus จัดเก็บข้อมูลในรูปแบบ time series ซึ่งเป็นการจัดเก็บข้อมูลที่มีการบันทึกค่าตัวเลข (metrics) พร้อมกับ timestamp

PromQL (Prometheus Query Language)

ภาษาคิวรีที่ทรงพลังสำหรับดึงและวิเคราะห์ข้อมูลที่เก็บอยู่ใน Prometheus สามารถใช้งานในการดึงข้อมูลเพื่อสร้างกราฟ, ตั้งค่าการแจ้งเตือน หรือการวิเคราะห์อื่นๆ

การเก็บข้อมูลด้วยการ Pull

Prometheus ใช้รูปแบบการดึงข้อมูล (pull-based) โดยการเข้าไปดึงข้อมูลจาก endpoints ที่ได้ตั้งค่าไว้ ซึ่งทำให้มีความยืดหยุ่นและสามารถควบคุมการเก็บข้อมูลได้ง่าย

Exporters

โปรแกรมหรือสคริปต์ที่ใช้ในการแปลงข้อมูลจากระบบต่างๆ มาเป็นรูปแบบที่ Prometheus สามารถดึงไปใช้งานได้ เช่น Node Exporter สำหรับดึงข้อมูลระบบปฏิบัติการ, Blackbox Exporter สำหรับตรวจสอบสถานะของเว็บไซต์

การแจ้งเตือน (Alerting)

Prometheus มีระบบการแจ้งเตือนผ่าน Alertmanager ที่สามารถกำหนดเงื่อนไขการแจ้งเตือนต่างๆ และกำหนดวิธีการแจ้งเตือน เช่น ผ่านอีเมล, Slack, PagerDuty เป็นต้น

การรวมเข้ากับระบบอื่น

Prometheus สามารถรวมเข้ากับเครื่องมือและแพลตฟอร์มอื่นๆ ได้อย่างง่ายดาย เช่น Grafana สำหรับการสร้าง dashboard ที่สวยงามและสามารถแสดงข้อมูลจาก Prometheus ได้

การทำงานของ **Prometheus**

Scraping:

Prometheus จะทำการดึงข้อมูลจาก endpoints ที่เรียกว่า "targets" ซึ่งสามารถกำหนดได้ในไฟล์คอนฟิกโดยใช้ URL ที่จะดึงข้อมูล metrics

Storage:

ข้อมูลที่ดึงมาได้จะถูกจัดเก็บใน local storage ของ Prometheus โดยใช้ time series database (TSDB) ที่มีประสิทธิภาพสูง

Querying:

ผู้ใช้สามารถใช้ PromQL เพื่อคิวรีข้อมูลที่เก็บไว้ใน Prometheus เพื่อการวิเคราะห์และแสดงผล

Alerting:

ระบบแจ้งเตือนของ Prometheus จะตรวจสอบเงื่อนไขที่กำหนดไว้และส่งการแจ้งเตือนไปยัง Alertmanager เมื่อมีการตรวจพบปัญหา

Grafana เป็นเครื่องมือโอเพนซอร์สที่ใช้สำหรับการวิเคราะห์และแสดงผลข้อมูลจากแหล่งข้อมูลต่างๆ โดยเฉพาะอย่างยิ่งในการสร้างแดชบอร์ด (dashboard) ที่สวยงามและใช้งานง่ายเพื่อมอนิเตอร์ระบบและแอปพลิเคชัน Grafana ถูกพัฒนาโดย Torkel Ödegaard ในปี 2014 และกลายเป็นหนึ่งในเครื่องมือยอดนิยมในกลุ่ม DevOps และ SRE (Site Reliability Engineering)

คุณสมบัติหลักของ **Grafana**

การสนับสนุนแหล่งข้อมูลหลากหลาย:

Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลายประเภทได้ เช่น Prometheus, InfluxDB, Elasticsearch, MySQL, PostgreSQL, Graphite และอื่นๆ อีกมากมาย ทำให้มีความยืดหยุ่นในการใช้งาน

การสร้างและปรับแต่งแดชบอร์ด:

ผู้ใช้สามารถสร้างและปรับแต่งแดชบอร์ดได้อย่างง่ายดายด้วยการลากและวาง (drag-and-drop) และเลือกประเภทของกราฟ, ตาราง, และ visualizations ต่างๆ เพื่อแสดงข้อมูลตามความต้องการ

การแจ้งเตือน (Alerting):

Grafana มีระบบการแจ้งเตือนที่สามารถตั้งค่าให้ตรวจสอบเงื่อนไขต่างๆ และส่งการแจ้งเตือนผ่านช่องทางต่างๆ เช่น อีเมล, Slack, PagerDuty, และอื่นๆ

การแชร์และการนำเสนอ:

Grafana ช่วยให้ผู้ใช้สามารถแชร์แดชบอร์ดกับทีมงานหรือบุคคลอื่นๆ ได้ง่ายๆ ผ่านลิงก์หรือการฝังแดชบอร์ดในเว็บไซต์อื่นๆ นอกจากนี้ยังมีโหมดการนำเสนอที่ช่วยให้การแสดงผลข้อมูลเป็นไปอย่างมืออาชีพ

ปลั๊กอิน:

Grafana มีระบบปลั๊กอินที่ช่วยขยายความสามารถเพิ่มเติม เช่น การเพิ่มแหล่งข้อมูลใหม่ๆ, ประเภทของ visualizations ใหม่ๆ และการอินทิเกรตกับเครื่องมืออื่นๆ

การทำงานของ **Grafana**

การเชื่อมต่อกับแหล่งข้อมูล:

ผู้ใช้สามารถกำหนดแหล่งข้อมูลใน Grafana โดยเลือกประเภทของแหล่งข้อมูลและใส่ข้อมูลการเชื่อมต่อ เช่น URL, ชื่อผู้ใช้, รหัสผ่าน เป็นต้น

การสร้างแดชบอร์ดและแผงควบคุม (Panels):

ผู้ใช้สามารถสร้างแดชบอร์ดใหม่และเพิ่มแผงควบคุม (panels) ในแดชบอร์ดนั้น โดยเลือกประเภทของการแสดงผล เช่น กราฟ, ตาราง, ฮีตแมป (heatmap) และอื่นๆ

การตั้งค่าคิวรี:

ในแต่ละแผงควบคุม ผู้ใช้สามารถตั้งค่าคิวรีเพื่อดึงข้อมูลจากแหล่งข้อมูลที่ได้กำหนดไว้ โดยใช้ภาษาคิวรีที่แหล่งข้อมูลนั้นๆ สนับสนุน เช่น PromQL สำหรับ Prometheus, SQL สำหรับฐานข้อมูลทั่วไป เป็นต้น

การตั้งค่าและจัดการการแจ้งเตือน:

ผู้ใช้สามารถตั้งค่าเงื่อนไขการแจ้งเตือนในแผงควบคุมต่างๆ และกำหนดช่องทางการแจ้งเตือนเพื่อให้ทราบเมื่อเกิดเหตุการณ์ที่กำหนดไว้

การแชร์และการนำเสนอ:

ผู้ใช้สามารถแชร์แดชบอร์ดหรือแผงควบคุมเฉพาะกับคนอื่นๆ ได้ผ่านลิงก์ที่สามารถกำหนดสิทธิการเข้าถึงหรือฝังแดชบอร์ดในเว็บไซต์อื่นๆ ได้

การผสมผสานระหว่าง **Prometheus** และ **Grafana** ช่วยให้ผู้ใช้สามารถติดตามและวิเคราะห์ข้อมูลการทำงานของระบบคอมพิวเตอร์ได้อย่างมีประสิทธิภาพและสะดวกสบาย โดย Grafana สามารถเรียกใช้ข้อมูลที่ถูกเก็บรวบรวมไว้ใน Prometheus และแสดงผลในรูปแบบของแผนภูมิและพาเนลต่างๆ ที่สวยงามและมีประสิทธิภาพ

2.ทำไมต้องใช้ tool นี้

มีหลายเหตุผลที่ทำให้ **Prometheus** เป็นตัวเลือกที่ยอดเยี่ยมสำหรับการมอนิเตอร์ระบบ

1. ใช้งานง่าย: Prometheus ติดตั้งและใช้งานง่าย เหมาะสำหรับผู้เริ่มต้นใช้งาน

2. มีประสิทธิภาพ: รองรับการ Scale แนวขนาน เหมาะสำหรับระบบขนาดใหญ่

3. รองรับระบบหลากหลาย: รองรับการใช้งานกับระบบต่างๆ เช่น Cloud, Container, Application, Infrastructure

4. มี Community ขนาดใหญ่: หาข้อมูลและความช่วยเหลือได้ง่าย

5. ฟีเจอร์ครบครัน: รองรับการเก็บข้อมูลแบบ Time Series, การ Query ข้อมูล, ระบบแจ้งเตือน ฯลฯ

เหตุผลที่ใช้ **Grafana** ในการ สร้าง **Dashboard**

การแสดงผลที่สวยงามและเข้าใจง่าย:

Grafana มีอินเตอร์เฟซที่ใช้งานง่ายและสามารถสร้างแดชบอร์ดที่สวยงามและเข้าใจง่ายด้วยการลากและวาง รวมถึงมีประเภทของ visualizations หลากหลาย

การสนับสนุนแหล่งข้อมูลหลายประเภท:

Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลากหลาย เช่น Prometheus, InfluxDB, Elasticsearch, MySQL, PostgreSQL เป็นต้น ทำให้สามารถรวมข้อมูลจากแหล่งต่างๆ ไว้ในแดชบอร์ดเดียวกันได้

การตั้งค่าและจัดการการแจ้งเตือน:

Grafana มีระบบการแจ้งเตือนที่สามารถกำหนดเงื่อนไขการแจ้งเตือนและส่งการแจ้งเตือนผ่านช่องทางต่างๆ ได้ เช่น อีเมล, Slack, PagerDuty เป็นต้น

การแชร์และการนำเสนอ:

Grafana ช่วยให้ผู้ใช้สามารถแชร์แดชบอร์ดกับทีมงานหรือบุคคลอื่นๆ ได้ง่ายๆ ผ่านลิงก์หรือการฝังแดชบอร์ดในเว็บไซต์อื่นๆ นอกจากนี้ยังมีโหมดการนำเสนอที่ช่วยให้การแสดงผลข้อมูลเป็นไปอย่างมืออาชีพ

ระบบปลั๊กอินที่หลากหลาย:

Grafana มีระบบปลั๊กอินที่ช่วยขยายความสามารถเพิ่มเติม เช่น การเพิ่มแหล่งข้อมูลใหม่ๆ, ประเภทของ visualizations ใหม่ๆ และการอินทิเกรตกับเครื่องมืออื่นๆ

การใช้ Prometheus ร่วมกับ Grafana มีข้อดีหลายประการที่ทำให้เป็นคู่เครื่องมือที่ได้รับความนิยมมากในกลุ่มผู้พัฒนาและผู้ดูแลระบบเมื่อเปรียบเทียบกับเครื่องมือมอนิเตอร์และแสดงผลข้อมูลอื่นๆ ข้อดีหลักๆ มีดังนี้:

การรวมกันที่สมบูรณ์แบบ (Perfect Integration)

-**การเก็บข้อมูลที่เชื่อถือได้** Prometheus มีการเก็บข้อมูลแบบ time series ที่มีประสิทธิภาพสูง และสามารถดึงข้อมูลจากระบบต่างๆ ได้อย่างมีประสิทธิภาพ

-**การแสดงผลที่ทรงพลัง** Grafana สามารถแสดงผลข้อมูลจาก Prometheus ได้อย่างสวยงามและเข้าใจง่าย โดยมีเครื่องมือการสร้างแดชบอร์ดที่หลากหลายและยืดหยุ่น

-**การสนับสนุนแหล่งข้อมูลหลากหลาย** Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลายประเภท ไม่เพียงแต่ Prometheus ยังรวมถึง InfluxDB, Elasticsearch, Graphite, MySQL, PostgreSQL และอื่นๆ ซึ่งทำให้สามารถรวมข้อมูลจากแหล่งต่างๆ ในแดชบอร์ดเดียวได้

-**การขยายตัวที่ดี** Prometheus มี exporters หลากหลายที่สามารถดึงข้อมูลจากระบบต่างๆ เช่น ระบบปฏิบัติการ, ฐานข้อมูล, แอปพลิเคชัน และเครือข่าย ทำให้สามารถขยายความสามารถได้ตามความต้องการ

-**การจัดการการแจ้งเตือน** ใน Grafana Grafana เพิ่มความยืดหยุ่นในการจัดการการแจ้งเตือน โดยสามารถส่งการแจ้งเตือนไปยังช่องทางต่างๆ เช่น อีเมล, Slack, PagerDuty และอื่นๆ

-**มีความยืดหยุ่นและการใช้งานที่ง่าย** ด้วย ภาษา PromQL ที่ทรงพลัง PromQL (Prometheus Query Language) ช่วยให้สามารถคิวรีและวิเคราะห์ข้อมูลได้อย่างมีประสิทธิภาพและละเอียด

การพัฒนาและการอัปเดตอย่างต่อเนื่อง ทั้ง Prometheus และ Grafana มีการพัฒนาและอัปเดตเวอร์ชันใหม่ๆ อย่างต่อเนื่อง เพื่อเพิ่มประสิทธิภาพและฟีเจอร์ใหม่ๆ ตามความต้องการของผู้ใช้

เปรียบเทียบกับเครื่องมืออื่นๆ

เทียบกับ Nagios:

Prometheus และ Grafana มีความยืดหยุ่นและสามารถขยายตัวได้ดีกว่า ในขณะที่ Nagios อาจมีการตั้งค่าที่ยุ่งยากกว่าและไม่เน้นที่การเก็บข้อมูลแบบ time series

เทียบกับ Zabbix:

Zabbix มีความสามารถในการมอนิเตอร์ที่ดี แต่ Prometheus และ Grafana มอบการแสดงผลที่สวยงามและมีประสิทธิภาพมากกว่า รวมถึงการใช้ PromQL ที่ทรงพลัง

เทียบกับ Elasticsearch + Kibana:

Elasticsearch และ Kibana มักใช้ในการวิเคราะห์ log และการค้นหาข้อมูลใน log ซึ่ง Grafana สามารถใช้งานในด้านนี้ได้เช่นกัน แต่การเก็บข้อมูลแบบ time series ของ Prometheus ทำให้เหมาะสมกับการมอนิเตอร์ระบบและการแจ้งเตือนมากกว่า

3. ยกตัวอย่าง usecase ที่ใช้

โดย usecase ที่ต้องใช้ Prometheus และ Grafana มีดังนี้

1.การมอนิเตอร์เซิร์ฟเวอร์และระบบปฏิบัติการ

รายละเอียด: มอนิเตอร์ทรัพยากรของเซิร์ฟเวอร์ เช่น CPU, หน่วยความจำ, การใช้งานดิสก์, การใช้งานเครือข่าย

ประโยชน์:

-สามารถตรวจสอบการใช้งานทรัพยากรได้แบบเรียลไทม์

-แจ้งเตือนเมื่อการใช้งานทรัพยากรเกินค่าที่กำหนด

2. การมอนิเตอร์แอปพลิเคชัน

รายละเอียด: มอนิเตอร์เมตริกต่างๆ ของแอปพลิเคชัน เช่น จำนวนคำขอ, เวลาในการตอบสนอง, อัตราการเกิดข้อผิดพลาด

สามารถติดตามประสิทธิภาพของแอปพลิเคชันได้ ตรวจจับปัญหาที่เกิดขึ้นในแอปพลิเคชันและแจ้งเตือนเมื่อมีข้อผิดพลาด

ประโยชน์:

-สามารถติดตามประสิทธิภาพของแอปพลิเคชันได้

-ตรวจจับปัญหาที่เกิดขึ้นในแอปพลิเคชันและแจ้งเตือนเมื่อมีข้อผิดพลาด

3. การมอนิเตอร์คอนเทนเนอร์และ Kubernetes

รายละเอียด: มอนิเตอร์การทำงานของคอนเทนเนอร์และ Kubernetes เช่น การใช้งานทรัพยากรของคอนเทนเนอร์, สถานะของ pods, nodes

ประโยชน์:

-สามารถตรวจสอบการใช้งานทรัพยากรและสถานะของคลัสเตอร์ Kubernetes ได้อย่างละเอียด

-แจ้งเตือนเมื่อเกิดปัญหากับคอนเทนเนอร์หรือ pods

4. การมอนิเตอร์ระบบเครือข่าย

รายละเอียด: มอนิเตอร์ประสิทธิภาพของเครือข่าย เช่น แบนด์วิธ, ความหน่วงเวลา (latency), การสูญหายของแพ็กเก็ต

ประโยชน์:

-สามารถตรวจสอบสถานะและประสิทธิภาพของเครือข่ายได้แบบเรียลไทม์

-แจ้งเตือนเมื่อมีปัญหาการเชื่อมต่อหรือการตอบสนองของบริการเครือข่าย

5. การมอนิเตอร์ฐานข้อมูล

รายละเอียด: มอนิเตอร์สถานะและประสิทธิภาพของฐานข้อมูล เช่น การใช้งาน CPU, จำนวนคำสั่งที่รัน, เวลาตอบสนองของคำสั่ง

ประโยชน์:

-สามารถติดตามประสิทธิภาพและสถานะของฐานข้อมูลได้

-แจ้งเตือนเมื่อการใช้งานทรัพยากรเกินค่าที่กำหนดหรือเกิดปัญหาในการเชื่อมต่อฐานข้อมูล

6. การมอนิเตอร์การพัฒนาและการปล่อยระบบ (CI/CD)

รายละเอียด: มอนิเตอร์กระบวนการ CI/CD เช่น สถานะของ build, จำนวนการ deploy, เวลาในการ deploy

ประโยชน์:

สามารถติดตามกระบวนการพัฒนาและการปล่อยระบบได้อย่างละเอียด

แจ้งเตือนเมื่อเกิดปัญหาในกระบวนการ CI/CD

การใช้ Prometheus และ Grafana ร่วมกันช่วยให้สามารถมอนิเตอร์และแสดงผลข้อมูลระบบได้อย่างมีประสิทธิภาพ ทำให้สามารถตรวจสอบและตอบสนองต่อปัญหาที่เกิดขึ้นได้อย่างรวดเร็ว

4.การทำ Demo

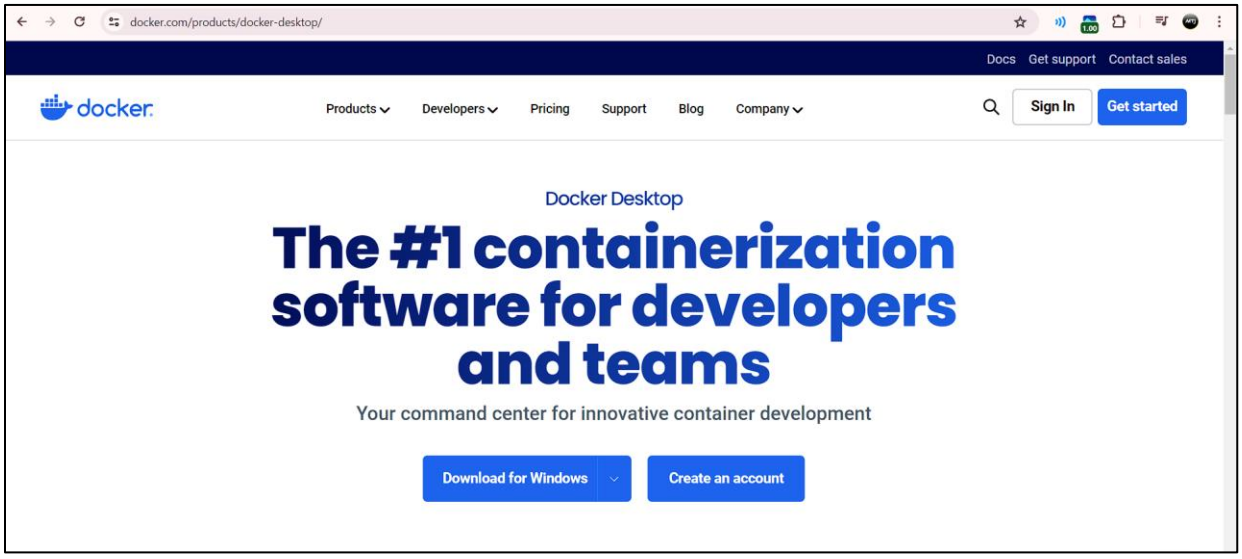
โดย use case **ที่นำมาใช้ใน คือการ** monitoring server network **และ** OS

การทำ Demo ของระบบปฏิบัติการและเครือข่าย นั้นลองทำเองได้ ตามขั้นตอนดังนี้

1. การติดตั้ง Docker บน Windows

ดาวน์โหลด Docker Desktop:ไปที่เว็บไซต์ Docker Desktop

คลิก "Download for Windows" เพื่อดาวน์โหลด Docker Desktop Installer



ติดตั้ง Docker Desktop:

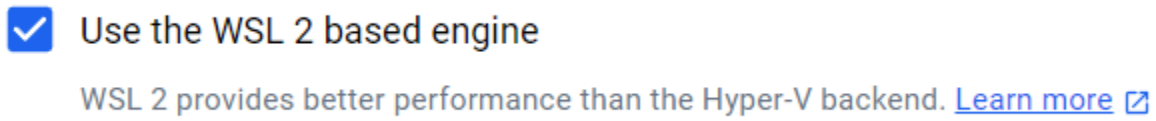
เปิดไฟล์ที่ดาวน์โหลดมา (Docker Desktop Installer.exe)

ทำตามคำแนะนำบนหน้าจอเพื่อดำเนินการติดตั้ง จากนั้น รีสตาร์ทเครื่องหากจำเป็น

เปิดใช้งาน Docker Desktop:

หลังการติดตั้งเสร็จสิ้น ให้เปิด Docker Desktop

ระบบอาจขอสิทธิ์ในการใช้ WSL 2 (Windows Subsystem for Linux 2) ให้ทำการติดตั้งและตั้งค่า WSL 2 ตามคำแนะนำ



หรือ ทำตาม Docs ใน Docker Website

https://docs.docker.com/desktop/install/windows-install/

การตรวจสอบการติดตั้ง

เปิด Command Prompt หรือ PowerShell และพิมพ์คำสั่ง:

docker –version

หากการติดตั้งสำเร็จ จะมีข้อความแสดงเวอร์ชั่นของ Docker ซึ่งจะได้หน้าตาประมาณนี้

```
C:\Users\Yannawut So cool>docker version
Client:
 Cloud integration: v1.0.35+desktop.13
 Version:           26.0.0
 API version:       1.45
 Go version:        go1.21.8
 Git commit:        2ae903e
 Built:             Wed Mar 20 15:18:56 2024
 OS/Arch:           windows/amd64
 Context:           default

Server: Docker Desktop 4.29.0 (145265)
 Engine:
  Version:          26.0.0
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.21.8
  Git commit:       8b79278
  Built:            Wed Mar 20 15:18:01 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.28
  GitCommit:        ae07eda36dd25f8a1b98dfbf587313b99c0190bb
 runc:
  Version:          1.1.12
  GitCommit:        v1.1.12-0-g51d5e94
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

เมื่อติดตั้ง Docker เรียบร้อย แล้ว

2.ทำการสร้าง Rest API POST ลง database โดยใช้ mysql และ GET มาใช้ในขั้นตอนต่อๆไป

ส่วนของโค้ด Rest API

เราจะต้องติดตั้ง Library ที่ import เข้ามาด้วย ใช้ คำสั่ง pip install <ชื่อของlibrary>

```
from flask import Flask, request, jsonify
from flask_cors import CORS, cross_origin
from flask_sqlalchemy import SQLAlchemy
from prometheus_client import start_http_server, Summary, Gauge

# สร้างแอป Flask
app = Flask(__name__)
# กำหนดการเชื่อมต่อฐานข้อมูล MySQL
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:rootpassword@mysql:3306/python_api'
# สร้าง SQLAlchemy object เพื่อจัดการฐานข้อมูล
db = SQLAlchemy(app)

# เปิดใช้งาน CORS สำหรับแอปนี้
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

# Prometheus Metrics
# Summary เพื่อวัดเวลาที่ใช้ในการประมวลผลคำขอ
REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')
# Gauge เพื่อวัดจำนวนรายการในฐานข้อมูล
ITEM_COUNT = Gauge('item_count', 'Number of items in the database')

# สร้าง Model สำหรับตาราง 'Item'
class Item(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    price = db.Column(db.Float, nullable=False)
```

```
    currency = db.Column(db.String(10), nullable=False)
    quantity = db.Column(db.Integer, nullable=False)

# สร้าง endpoint สำหรับสร้างรายการใหม่
@app.route('/items', methods=['POST'])
def create_item():
    data = request.json
    new_item = Item(
        name=data['name'],
        price=data['price'],
        currency=data['currency'],
        quantity=data['quantity']
    )
    db.session.add(new_item)
    db.session.commit()
    update_item_count()  # อัปเดตจำนวนรายการในฐานข้อมูล
    return jsonify({'message': 'Item created successfully'}), 201

# สร้าง endpoint สำหรับดึงรายการทั้งหมด
@app.route('/items', methods=['GET'])
@cross_origin()
def get_items():
    items = Item.query.all()
    items_list = [
        {
            'id': item.id,
            'name': item.name,
            'price': item.price,
            'currency': item.currency,
            'quantity': item.quantity
        }
        for item in items
    ]
    update_item_count()  # อัปเดตจำนวนรายการในฐานข้อมูล
    return jsonify(items_list), 200
```

```
# สร้าง endpoint สำหรับดึง metrics ของ Prometheus
@app.route('/metrics', methods=['GET'])
def metrics():
    from prometheus_client import generate_latest
    return generate_latest(), 200

# ฟังก์ชันสำหรับอัปเดตจำนวนรายการในฐานข้อมูล
def update_item_count():
    count = Item.query.count()
    ITEM_COUNT.set(count)

if __name__ == '__main__':
    with app.app_context():
        db.create_all()  # สร้างตารางในฐานข้อมูลถ้ายังไม่มี
    start_http_server(8000)  # เริ่มต้น HTTP server สำหรับ Prometheus metrics ที่ port 8000
    app.run(debug=True, host='0.0.0.0', port=5000)  # รันแอป Flask ที่ host '0.0.0.0' และ port 5000
```

3.จากนั้นเราจะทำการสร้างไฟล์ docker-compose.yml เพื่อสร้าง container ที่จำเป็นต้องใช้ เหตุผลที่ต้องใช้ docker-compose.yml เพราะว่า จะช่วยให้จัดการ Container หลายๆตัวในแอปพลิเคชั่น ได้อย่างง่าย กำหนดการเชื่อมต่อ ตั้งค่า config ที่ซับซ้อนต่างๆ

สิ่งที่เราต้องติดตั้ง

1. Prometheus
2. Grafana
3. Mysql
4. Mysql-exporter
5. Phpmyadmin
6. Flask-api

ส่วนของโค้ด ในไฟล์ docker-compose.yml

```
version: '3.7'
```

```yaml
services:



# บริการ MySQL
  mysql:
    image: mysql:latest  # ใช้ภาพ MySQL เวอร์ชันล่าสุด
    container_name: mysql  # ตั้งชื่อคอนเทนเนอร์เป็น mysql
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword  # กำหนดรหัสผ่านสำหรับ root
      MYSQL_DATABASE: python_api  # สร้างฐานข้อมูลชื่อ python_api
    volumes:
      - mysql-data:/var/lib/mysql  # เก็บข้อมูลใน volume ชื่อ mysql-data
    ports:
      - "3306:3306"  # เปิดพอร์ต 3306
    restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด

  # บริการ MySQL Exporter สำหรับ Prometheus
  mysqld-exporter:
    image: prom/mysqld-exporter:latest  # ใช้ภาพ mysqld-exporter เวอร์ชันล่าสุด
    container_name: mysqld-exporter  # ตั้งชื่อคอนเทนเนอร์เป็น mysqld-exporter
    environment:
      DATA_SOURCE_NAME: root:rootpassword@tcp(mysql:3306)/  # กำหนดข้อมูลการเชื่อมต่อ MySQL
      MYSQLD_EXPORTER_EXPORTER_DISABLE: "slave_status"  # ปิดการส่งข้อมูล slave status
    command:
      - --config.my-cnf=/etc/mysql/my.cnf  # ใช้ไฟล์ my.cnf เป็นการตั้งค่า
    ports:
      - "9104:9104"  # เปิดพอร์ต 9104
    depends_on:
      - mysql  # ขึ้นอยู่กับบริการ mysql
    volumes:
      - ./my.cnf:/etc/mysql/my.cnf  # ใช้ไฟล์ my.cnf จากโฟลเดอร์ปัจจุบัน
    restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด

  # บริการ phpMyAdmin
  phpmyadmin:
```

```
  image: phpmyadmin/phpmyadmin:latest  # ใช้ภาพ phpMyAdmin เวอร์ชันล่าสุด
  container_name: phpmyadmin  # ตั้งชื่อคอนเทนเนอร์เป็น phpmyadmin
  environment:
    PMA_HOST: mysql  # กำหนด host เป็น mysql
    MYSQL_ROOT_PASSWORD: rootpassword  # กำหนดรหัสผ่านสำหรับ root
  ports:
    - "8080:80"  # เปิดพอร์ต 8080
  depends_on:
    - mysql  # ขึ้นอยู่กับบริการ mysql
  restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด

# บริการ Prometheus
prometheus:
  image: prom/prometheus:latest  # ใช้ภาพ Prometheus เวอร์ชันล่าสุด
  container_name: prometheus  # ตั้งชื่อคอนเทนเนอร์เป็น prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml  # ใช้ไฟล์ prometheus.yml จากโฟลเดอร์ปัจจุบัน
  ports:
    - "9090:9090"  # เปิดพอร์ต 9090
  command:
    - "--config.file=/etc/prometheus/prometheus.yml"  # ใช้ไฟล์การตั้งค่า prometheus.yml
  depends_on:
    - mysqld-exporter  # ขึ้นอยู่กับบริการ mysqld-exporter
  restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด

# บริการ Grafana
grafana:
  image: grafana/grafana:latest  # ใช้ภาพ Grafana เวอร์ชันล่าสุด
  container_name: grafana  # ตั้งชื่อคอนเทนเนอร์เป็น grafana
  ports:
    - "3000:3000"  # เปิดพอร์ต 3000
  volumes:
    - grafana-storage:/var/lib/grafana  # เก็บข้อมูลใน volume ชื่อ grafana-storage
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin  # กำหนดรหัสผ่านผู้ดูแลระบบ
```

```
    depends_on:
      - prometheus  # ขึ้นอยู่กับบริการ prometheus
    restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด

  # บริการ Flask API
  flask-api:
    build: .  # สร้างภาพจาก Dockerfile ในโฟลเดอร์ปัจจุบัน
    container_name: flask-api  # ตั้งชื่อคอนเทนเนอร์เป็น flask-api
    environment:
      - SQLALCHEMY_DATABASE_URI=mysql+pymysql://root:rootpassword@mysql:3306/python_api  # กำหนด
URI สำหรับเชื่อมต่อฐานข้อมูล
    ports:
      - "5000:5000"  # เปิดพอร์ต 5000
    depends_on:
      - mysql  # ขึ้นอยู่กับบริการ mysql
    restart: always  # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
    command: /bin/sh -c "sleep 10 && flask run --host=0.0.0.0"  # หน่วงเวลาการรัน Flask 10 วินาทีเพื่อให้บริการอื่น ๆ
พร้อมใช้งาน

# กำหนด volumes ที่ใช้เก็บข้อมูล
volumes:
  mysql-data:
  grafana-storage:
```

4.และเราจะต้อง config Prometheus โดยจะต้องสร้างไฟล์ Prometheus.yml **เพื่อใช้สำหรับการตั้งค่า ใน Prometheus**

ในส่วนของไฟล์ Prometheus.yml

```
global:
  scrape_interval: 15s  # กำหนดช่วงเวลาเริ่มต้นในการดึงข้อมูล (scrape) ทุกๆ 15 วินาที

scrape_configs:
  - job_name: 'prometheus'  # ชื่อ job สำหรับ Prometheus
    scrape_interval: 5s  # กำหนดช่วงเวลาในการดึงข้อมูล (scrape) สำหรับ job นี้ทุกๆ 5 วินาที
    static_configs:
```

```
    - targets: ['localhost:9090']  # กำหนดเป้าหมาย (target) คือ Prometheus เองที่รันอยู่บนพอร์ต 9090

  - job_name: 'mysqld-exporter'  # ชื่อ job สำหรับ MySQL exporter
    static_configs:
      - targets: ['mysqld-exporter:9104']  # กำหนดเป้าหมาย (target) คือ mysqld-exporter ที่รันอยู่บนพอร์ต 9104

  - job_name: 'flask-api'  # ชื่อ job สำหรับ Flask API
    metrics_path: /metrics  # กำหนดเส้นทาง (path) สำหรับดึงข้อมูล metrics
    static_configs:
      - targets: ['flask-api:5000']  # กำหนดเป้าหมาย (target) คือ Flask API ที่รันอยู่บนพอร์ต 5000
```

5.สร้าง Dockerfile เพื่อสร้าง Docker image สำหรับแอป Flask

ในส่วนของ dockerfile.yml

```
# Use the official Python image from the Docker Hub
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Run app.py when the container launches
CMD ["flask", "run", "--host=0.0.0.0"]
```

6.สร้างไฟล์ requirements.txt เพื่อระบุ dependencies ที่จำเป็น

ใน ส่วน requirements.txt

```
Flask  # ติดตั้ง Flask
flask_sqlalchemy  # ติดตั้ง Flask SQLAlchemy
flask_cors  # ติดตั้ง Flask CORS
pymysql  # ติดตั้ง PyMySQL
prometheus_client  # ติดตั้ง Prometheus Client
```

จากนั้น ให้รัน ไฟล์ docker-compose.yml ที่ config ไว้

โดยใช้ คำสั่ง docker-compose up -d ใน terminal หรือ power shell

เมื่อรันสำเร็จ จะได้ขึ้นแบบนี้

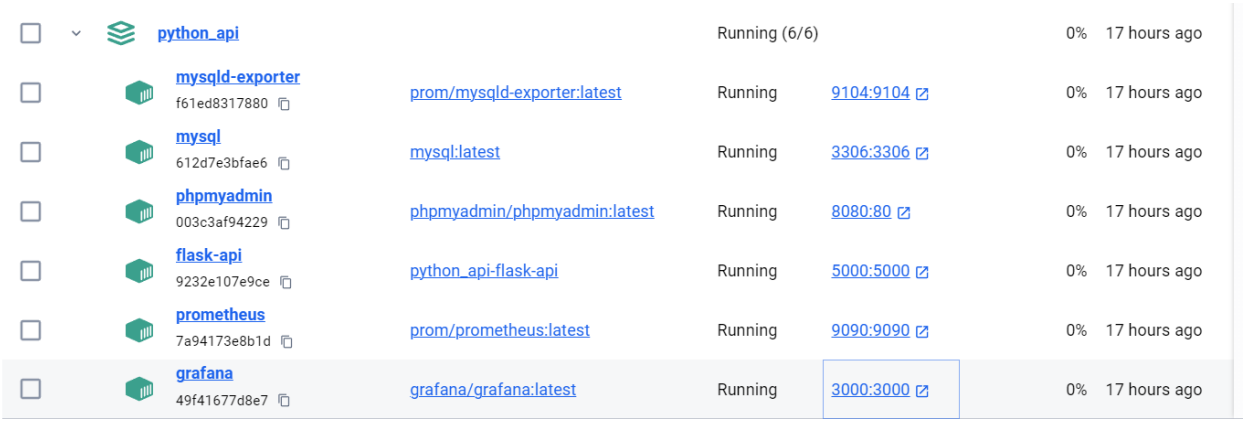
```
PS D:\Coding\Python_API> docker-compose up -d
time="2024-05-27T00:26:10+07:00" level=warning msg="D:\\Coding\\Python_API\\docker-compose.yml: `version` is obsolete"
[+] Running 7/7
 ✓ Network python_api_default  Created
 ✓ Container mysql             Started
 ✓ Container flask-api         Started
 ✓ Container mysqld-exporter   Started
 ✓ Container phpmyadmin        Started
 ✓ Container prometheus        Started
 ✓ Container grafana           Started
PS D:\Coding\Python_API>
```

เมื่อรันได้แล้วเราจะทำการเช็ค ว่า container ทั้งหมดถูกรันบน docker ครบทุกตัวไหม

โดยการใช้ คำสั่ง docker ps หรือดูที่ docker desktop ของเรา

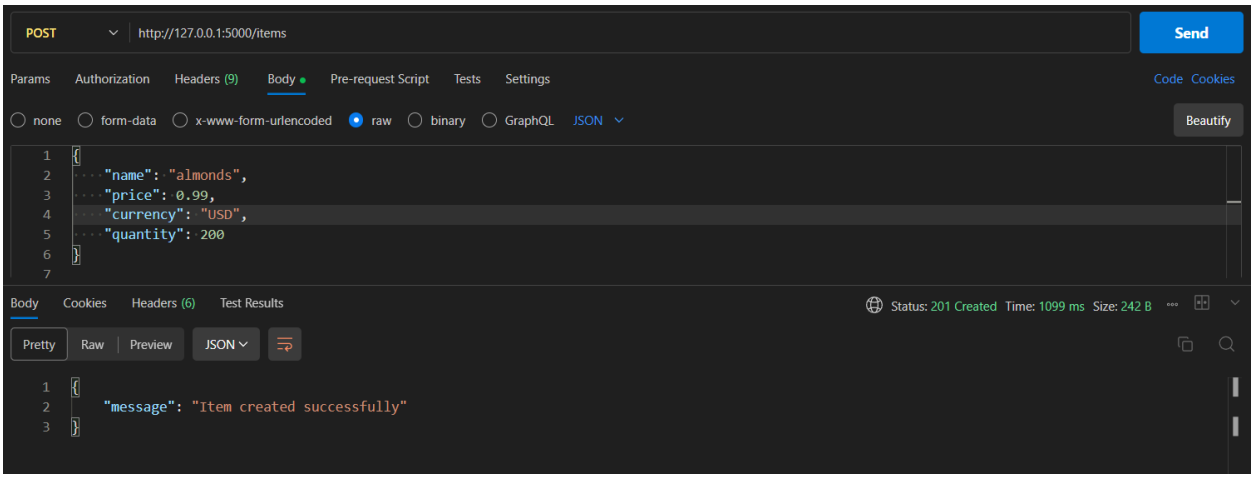
```
PS D:\Coding\Python_API> docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS       PORTS                                              NAMES
49f41677d8e7   grafana/grafana:latest         "/run.sh"                17 hours ago  Up 17 hours  0.0.0.0:3000->3000/tcp                             grafana
7a94173e8b1d   prom/prometheus:latest         "/bin/prometheus --c…"   17 hours ago  Up 17 hours  0.0.0.0:9090->9090/tcp                             prometheus
f61ed8317880   prom/mysqld-exporter:latest    "/bin/mysqld_exporte…"   17 hours ago  Up 17 hours  0.0.0.0:9104->9104/tcp                             mysqld-exporter
003c3af94229   phpmyadmin/phpmyadmin:latest   "/docker-entrypoint.…"   17 hours ago  Up 17 hours  0.0.0.0:8080->80/tcp                               phpmyadmin
9232e107e9ce   python_api-flask-api           "/bin/sh -c 'sleep 1…"   17 hours ago  Up 17 hours  0.0.0.0:5000->5000/tcp                             flask-api
612d7e3bfae6   mysql:latest                   "docker-entrypoint.s…"   17 hours ago  Up 17 hours  0.0.0.0:3306->3306/tcp, 33060/tcp                  mysql
```

หรือ ดูบน docker desktop



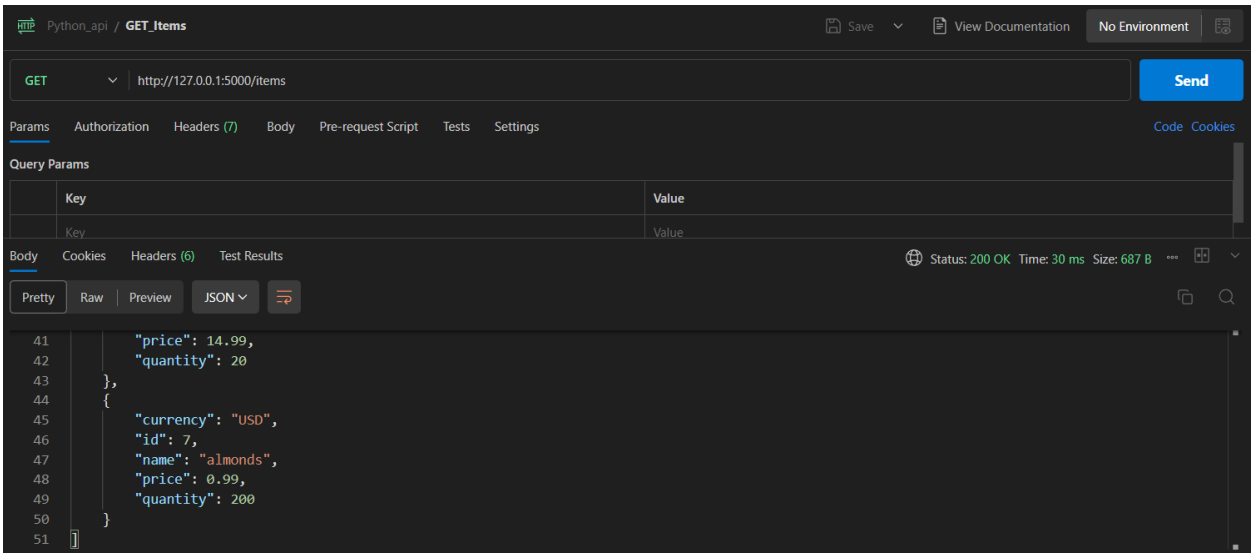
ถ้า status เป็น up หรือขึ้นว่า running ทั้งหมดแปลว่า ทุก container สามารถรันได้อย่างถูกต้อง บน docker แล้ว

7.ทำการ ทดสอบ POST data ลงไป mysql ของเรา ที่รันอยู่บน docker โดยใช้ postman



ถ้าสามารถ POST ลง database ได้สำเร็จ จะขึ้น message ว่า item create successfully

จากนั้นให้เราลอง GET data ที่เก็บไว้ โดยใช้ postman เหมือนเดิม

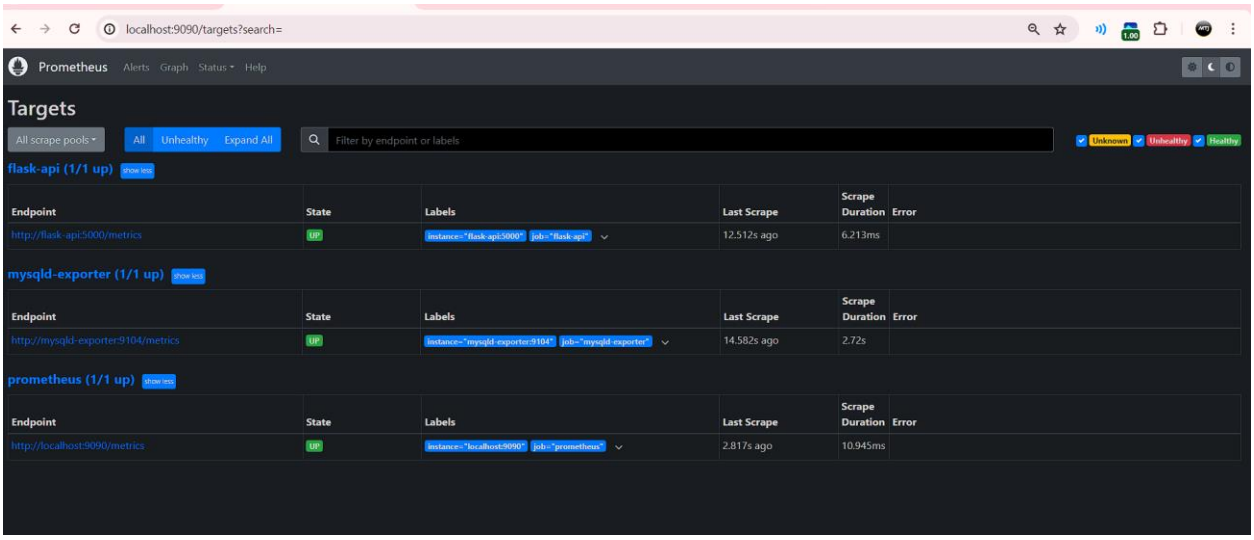


เราก็จะได้ข้อมูล ที่เก็บไปออกมาแบบนี้ แปลว่า ไฟล์ API เราทำงานได้ปกติ

เสร็จแล้วเราจะตรวจสอบ Prometheus ว่าแสดงสถานะของ target ซึ่งคือ endpoint ของ services ที่เราได้กำหนดไว้ใน ไฟล์ prometheus.yml ที่ Prometheus ทำการ scrape ข้อมูลอยู่ว่าสามารถดึงข้อมูลได้รึป่าว

โดยเข้าไปที่ localhost:9090 ที่หรือดู port ที่รัน อยู่บน container ใน docker desktop ก็ได้

ถ้า status up แบบนี้ สามารถดึงข้อมูลได้ทุกตัว



8.จากนั้น เราจะเข้าไปที่ Grafana ที่ localhost:3000

ไปที่ Data Sources > Add data source

เลือก Prometheus

ตั้งค่า URL เป็น http://prometheus:9090

คลิก Save & Test เพื่อทดสอบการเชื่อมต่อ

9.สร้าง Flask API Request Duration Dashboard

วัตถุประสงค์: แสดงเวลาในการประมวลผลคำขอของ Flask API

ไปที่ Grafana และสร้าง Dashboard ใหม่

เพิ่ม Panel ใหม่ ตั้งชื่อ Panel เป็น "API Request Duration"

เลือก Data Source เป็น Prometheus

ใส่ Query rate(request_processing_seconds_sum[5m]) / rate(request_processing_seconds_count[5m])

เลือกรูปแบบการแสดงผลเป็น Table หรือ ตามต้องการ

บันทึก Panel

10.สร้าง MySQL Database Item Count Dashboard

วัตถุประสงค์: แสดงจำนวนรายการในฐานข้อมูล MySQL

ไปที่ Grafana และสร้าง Dashboard ใหม่ (หรือใช้ Dashboard เดิม)

เพิ่ม Panel ใหม่ ตั้งชื่อ Panel เป็น "Database Item Count"

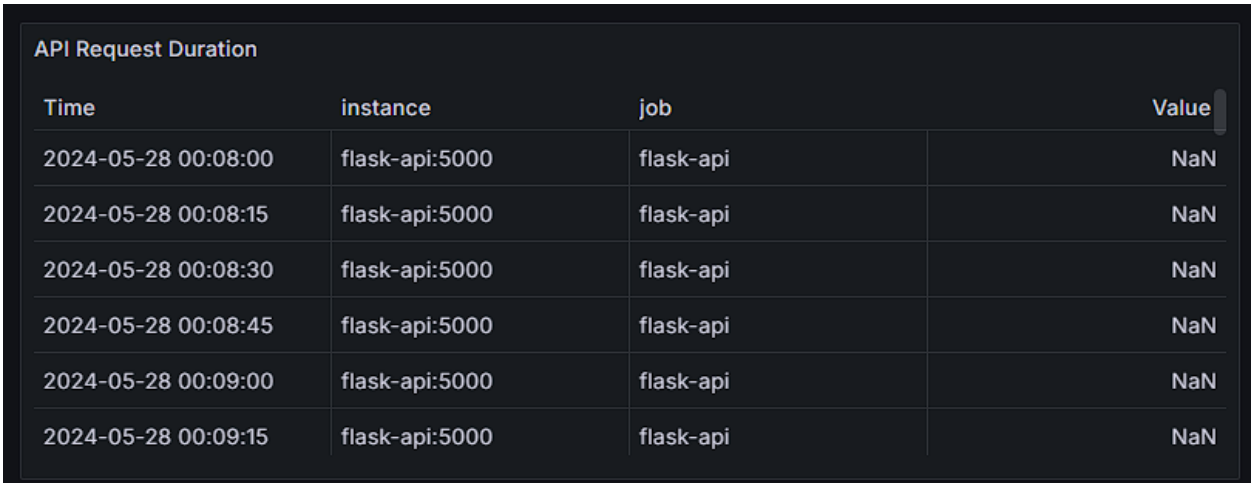
เลือก Data Source เป็น Prometheus

ใส่ Query item_count

เลือกรูปแบบการแสดงผลเป็น time series หรือตามต้องการ

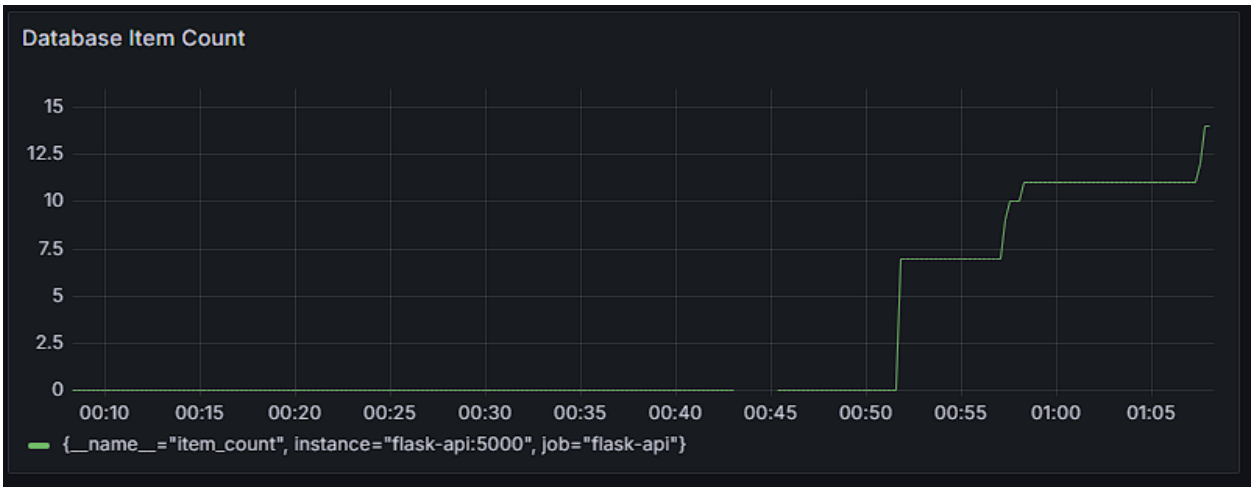
บันทึก Panel

จะได้เป็น Dashbroad **ดังรูปนี้**



API Request Duration

Time	instance	job	Value
2024-05-28 00:08:00	flask-api:5000	flask-api	NaN
2024-05-28 00:08:15	flask-api:5000	flask-api	NaN
2024-05-28 00:08:30	flask-api:5000	flask-api	NaN
2024-05-28 00:08:45	flask-api:5000	flask-api	NaN
2024-05-28 00:09:00	flask-api:5000	flask-api	NaN
2024-05-28 00:09:15	flask-api:5000	flask-api	NaN



สามารถปรับแต่งได้ตามความสะดวกใน การตรวจสอบหรือเพิ่ม Dashbroad รูปแบบได้

Link Github : https://github.com/YannawutRoumsuk/Prometheus-and-Grfana.git