



Section : Prometheus and Grafana

Documents create date : 21/05/2024

Version : 1.0

Status : initial Version

Document Author : Yannawut Roumsuk

Document Reviewer : Kittimasak Wangsri (Mentor Engineer)

คำนำ

เอกสารฉบับนี้มีวัตถุประสงค์เพื่อให้ความรู้และข้อมูลเกี่ยวกับการใช้งาน Prometheus และ Grafana ซึ่งเป็นเครื่องมือสำคัญในการมอนิเตอร์และวิเคราะห์ข้อมูลในระบบคอมพิวเตอร์สมัยใหม่ Prometheus เป็นระบบเก็บข้อมูลและแจ้งเตือนที่มีประสิทธิภาพสูง ซึ่งสามารถเก็บข้อมูลในรูปแบบของ time series และทำงานร่วมกับ Grafana ที่เป็นเครื่องมือแสดงผลข้อมูลที่มีความยืดหยุ่นสูง

ในเอกสารนี้จะครอบคลุมถึงการติดตั้งและการตั้งค่าเบื้องต้นของ Prometheus และ Grafana การนำเข้าข้อมูลจากแหล่งข้อมูลต่างๆ การสร้างแดชบอร์ดที่แสดงผลข้อมูลแบบเรียลไทม์ รวมถึงการตั้งค่าการแจ้งเตือนเพื่อตรวจสอบเหตุการณ์ที่สำคัญต่างๆ ในระบบ

เราเชื่อว่าเอกสารฉบับนี้จะเป็นประโยชน์สำหรับนักพัฒนาระบบ ผู้ดูแลระบบเครือข่าย และผู้ที่สนใจในเทคโนโลยีการมอนิเตอร์และการแสดงผลข้อมูล และหวังเป็นอย่างยิ่งว่าจะสามารถช่วยให้คุณนำความรู้ที่ได้รับไปประยุกต์ใช้ในการทำงานได้อย่างมีประสิทธิภาพ

นายณณวุฒิ ร่วมสุข

สารบัญ

คำนำ	A
Prometheus and Grafana.....	1
Prometheus and Grafana คืออะไร.....	1
Grafana	1
คุณสมบัติหลักของ Prometheus โอเคไหมครัตรงนี้	1
การทำงานของ Prometheus	2
คุณสมบัติหลักของ Grafana.....	2
การทำงานของ Grafana	3
เหตุผลที่ทำให้ต้องใช้ Prometheus.....	4
เหตุผลที่ใช้ Grafana ในการ สร้าง Dashboard.....	4
การใช้ Prometheus ร่วมกับ Grafana.....	5
Use case ที่ต้องใช้ Prometheus และ Grafana มีดังนี้	6
การทำ Demo เพื่อใช้ Tool นี้ตาม use case ที่ยกตัวอย่างขึ้นมา	8
1) การติดตั้ง Docker บน Windows.....	8
2) ทำการ Clone Git.....	10
3) ทำการสร้าง ในส่วนของ docker-compose.yml	12
4) สร้างในส่วนของ prometheus.yml	14
5) ในส่วนของ dockerfile เพื่อสร้าง docker Image สำหรับ app flask.....	15
6) ในส่วนของไฟล์ requirement.txt เพื่อระบุ dependencies ที่จำเป็นต้องใช้	15
7) จากนั้น ให้รัน ไฟล์ docker-compose.yml ที่ config ไว้.....	15
8) เข้าไปที่ phpMyAdmin บน port 8080 ใน docker desktop ของเรา เพื่อสร้าง table	17

9) ทำการ ทดสอบ ส่งและเก็บ data ลงไป mysql ของเรา ที่รันอยู่บน docker โดยใช้ postman	20
10) ทำการ GET ค่าข้อมูลจาก database โดยใช้ POSTMAN เหมือนกับในขั้นตอนที่ 9	21
11) เข้าไปที่ Prometheus ของเรา จากใน docker desktop บน port 9090	23
12) เข้าไปที่ Grafana บน docker desktop ของเรา บน port 3000.....	24
13) เริ่มสร้าง dashboard ใน Grafana ของเรา.....	26
สามารถอ่าน วิธีการเพิ่มติ่มได้ที่ ไฟล์ REAME.md ได้ที่ Link GitHub	32
อ้างอิง.....	33

Prometheus and Grafana

Prometheus and Grafana คืออะไร

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

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

คุณสมบัติหลักของ Prometheus โอเคไหมครับ

- **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 Alert manager 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 Alert manager เมื่อมีการตรวจพบปัญหา

คุณสมบัติหลักของ Grafana

- **การสนับสนุนแหล่งข้อมูลหลากหลาย**: Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลายประเภทได้ เช่น Prometheus, Influx DB, 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 ได้ผ่านลิงก์ที่สามารถกำหนดสิทธิ์การเข้าถึงหรือฝังแดชบอร์ดในเว็บไซต์อื่นๆ ได้

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

เหตุผลที่ทำให้ต้องใช้ Prometheus

- ใช้งานง่าย: Prometheus ติดตั้งและใช้งานง่าย เหมาะสำหรับผู้เริ่มต้นใช้งาน
- มีประสิทธิภาพ: รองรับการ Scale แนวขนาน เหมาะสำหรับระบบขนาดใหญ่
- รองรับระบบหลากหลาย: รองรับการใช้งานกับระบบต่างๆ เช่น Cloud, Container, Application, Infrastructure
- มี Community ขนาดใหญ่: หาข้อมูลและความช่วยเหลือได้ง่าย
- พีเจียร์ครบครัน: รองรับการเก็บข้อมูลแบบ Time Series, การ Query ข้อมูล, ระบบแจ้งเตือน ฯลฯ

เหตุผลที่ใช้ Grafana ในการ สร้าง Dashboard

- การแสดงผลที่สวยงามและเข้าใจง่าย: Grafana มีอินเทอร์เฟซที่ใช้งานง่ายและสามารถสร้างแดชบอร์ดที่สวยงามและเข้าใจง่ายด้วยการลากและวาง รวมถึงมีประเภทของ visualizations หลากหลาย
- การสนับสนุนแหล่งข้อมูลหลายประเภท: Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลากหลาย เช่น Prometheus, Influx DB, 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

มีข้อดีหลายประการที่ทำให้เป็นคู่เครื่องมือที่ได้รับความนิยมมากในกลุ่มผู้พัฒนาและผู้ดูแลระบบ เมื่อเปรียบเทียบกับเครื่องมือมอนิเตอร์และแสดงผลข้อมูลอื่นๆ ข้อดีหลักๆ มีดังนี้:

- **การเก็บข้อมูลที่เชื่อถือได้**

Prometheus มีการเก็บข้อมูลแบบ time series ที่มีประสิทธิภาพสูง และสามารถดึงข้อมูลจากระบบต่างๆ ได้อย่างมีประสิทธิภาพ

- **การแสดงผลที่ทรงพลัง**

Grafana สามารถแสดงผลข้อมูลจาก Prometheus ได้อย่างสวยงามและเข้าใจง่าย โดยมีเครื่องมือการสร้างแดชบอร์ดที่หลากหลายและยืดหยุ่น

- **การสนับสนุนแหล่งข้อมูลหลากหลาย**

Grafana สามารถเชื่อมต่อกับแหล่งข้อมูลหลายประเภท ไม่เพียงแต่ Prometheus ยังรวมถึง Influx DB, Elasticsearch, Graphite, MySQL, PostgreSQL และอื่นๆ ซึ่งทำให้สามารถรวมข้อมูลจากแหล่งต่างๆ ในแดชบอร์ดเดียวได้

- **การขยายตัวที่ดี**

Prometheus มี exporters หลากหลายที่สามารถดึงข้อมูลจากระบบต่างๆ เช่น ระบบปฏิบัติการ, ฐานข้อมูล, แอปพลิเคชัน และเครือข่าย ทำให้สามารถขยายความสามารถได้ตามความต้องการ

- **การจัดการการแจ้งเตือน**

ใน Grafana เพิ่มความยืดหยุ่นในการจัดการการแจ้งเตือน โดยสามารถส่งการแจ้งเตือนไปยังช่องทางต่างๆ เช่น อีเมล, Slack, PagerDuty และอื่นๆ

- **มีความยืดหยุ่นและการใช้งานที่ง่าย**

ด้วย ภาษา PromQL ที่ทรงพลัง PromQL (Prometheus Query Language) ช่วยให้สามารถคิวรีและวิเคราะห์ข้อมูลได้อย่างมีประสิทธิภาพและละเอียด

Use case ที่ต้องใช้ Prometheus และ Grafana มีดังนี้

- **การมอนิเตอร์เซิร์ฟเวอร์และระบบปฏิบัติการ**

รายละเอียด: มอนิเตอร์ทรัพยากรของเซิร์ฟเวอร์ เช่น CPU, หน่วยความจำ, การใช้งานดิสก์, การใช้
งานเครือข่าย

ประโยชน์:

- สามารถตรวจสอบการใช้งานทรัพยากรได้แบบเรียลไทม์
- แจ้งเตือนเมื่อการใช้งานทรัพยากรเกินค่าที่กำหนด

- **การมอนิเตอร์แอปพลิเคชัน**

รายละเอียด: มอนิเตอร์เมตริกต่างๆ ของแอปพลิเคชัน เช่น จำนวนคำขอ, เวลาในการตอบสนอง,
อัตราการเกิดข้อผิดพลาด

สามารถติดตามประสิทธิภาพของแอปพลิเคชันได้ ตรวจสอบปัญหาที่เกิดขึ้นในแอปพลิเคชันและแจ้ง
เตือนเมื่อมีข้อผิดพลาด

ประโยชน์:

- สามารถติดตามประสิทธิภาพของแอปพลิเคชันได้
- ตรวจสอบปัญหาที่เกิดขึ้นในแอปพลิเคชันและแจ้งเตือนเมื่อมีข้อผิดพลาด

- **การมอนิเตอร์คอนเทนเนอร์และ Kubernetes**

รายละเอียด: มอนิเตอร์การทำงานของคอนเทนเนอร์และ Kubernetes เช่น การใช้งานทรัพยากร
ของคอนเทนเนอร์, สถานะของ pods, nodes

ประโยชน์:

- สามารถตรวจสอบการใช้งานทรัพยากรและสถานะของคลัสเตอร์ Kubernetes ได้อย่างละเอียด
- แจ้งเตือนเมื่อเกิดปัญหากับคอนเทนเนอร์หรือ pods

- **การมอนิเตอร์ระบบเครือข่าย**

รายละเอียด: มอนิเตอร์ประสิทธิภาพของเครือข่าย เช่น แบนด์วิธ, ความหน่วงเวลา (latency), การ
สูญหายของแพ็คเก็ต

ประโยชน์:

- สามารถตรวจสอบสถานะและประสิทธิภาพของเครือข่ายได้แบบเรียลไทม์
- แจ้งเตือนเมื่อมีปัญหาการเชื่อมต่อหรือการตอบสนองของบริการเครือข่าย

- **การมอนิเตอร์ฐานข้อมูล**

รายละเอียด: มอนิเตอร์สถานะและประสิทธิภาพของฐานข้อมูล เช่น การใช้งาน CPU, จำนวนคำสั่งที่รัน, เวลาตอบสนองของคำสั่ง

ประโยชน์:

- สามารถติดตามประสิทธิภาพและสถานะของฐานข้อมูลได้
- แจ้งเตือนเมื่อการใช้งานทรัพยากรเกินค่าที่กำหนดหรือเกิดปัญหาในการเชื่อมต่อฐานข้อมูล

- **การมอนิเตอร์การพัฒนาและการปล่อยระบบ (CI/CD)**

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

ประโยชน์:

- สามารถติดตามกระบวนการพัฒนาและการปล่อยระบบได้อย่างละเอียด
- แจ้งเตือนเมื่อเกิดปัญหาในกระบวนการ CI/CD

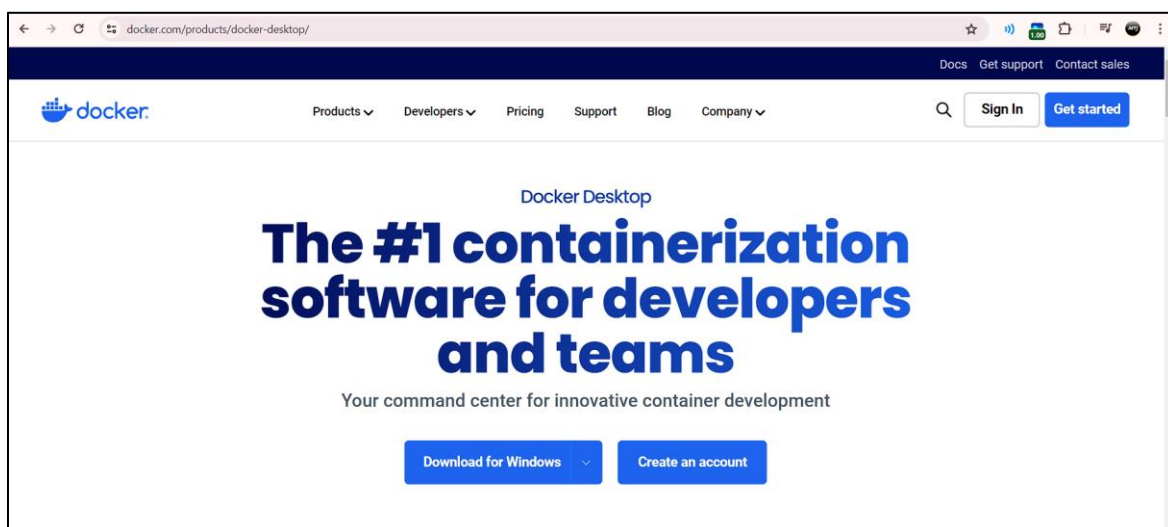
การทำ Demo เพื่อใช้ Tool นี้ตาม use case ที่ยกตัวอย่างขึ้นมา

โดย use case ที่นำมาใช้ใน คือการ monitoring server database และ OS การทำ Demo ของระบบปฏิบัติการและฐานข้อมูล สามารถทำตามขั้นตอนได้ ดังนี้ :

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

หมายเหตุ ระบบมีอุปสรรคในการใช้ WSL 2 (Windows Subsystem for Linux 2) เพื่อให้ได้ ประสิทธิภาพที่ดีกว่า ในการทำงานถ้าเลือกให้ใช้ ให้ทำการติดตั้งและตั้งค่า [WSL 2](#) ตามคำแนะนำของ Docker ได้



Use the WSL 2 based engine

WSL 2 provides better performance than the Hyper-V backend. [Learn more](#)

หรือสามารถอ่านเพิ่มเติมได้ที่ docs.docker.com

- ตรวจสอบการติดตั้ง

เปิด Command Prompt หรือ PowerShell และพิมพ์คำสั่ง:

`docker --version` หรือ `docker version`

เพื่อตรวจสอบว่า docker ถูกติดตั้งสำเร็จและใช้งานได้ ซึ่งจะได้แบบนี้

กรณีที่ใช้คำสั่ง `docker version`

```
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
```

กรณีที่ใช้คำสั่ง `docker --version`

```
C:\Users\Yannawut So cool>docker --version  
Docker version 26.0.0, build 2ae903e
```

ถ้าขึ้นแบบนี้ แปลว่าสามารถใช้ docker ได้แล้ว

2) ทำการ Clone Git

จากลิงค์นี้ : <https://github.com/YannawutRoumsuk/Prometheus-and-Grfana> หรือ สร้าง

ไฟล์ Code ตามนี้ ใน visual studio code ซึ่งจะประกอบด้วยไฟล์ หลักๆ 6 ไฟล์ :

- (a) docker-compose.yml
- (b) dockerfile
- (c) prometheus.yml
- (d) requirement.txt
- (e) my.cnf
- (f) app.py

ในส่วนของไฟล์ app.py

```

1. from flask import Flask, request, jsonify
2. from flask_cors import CORS, cross_origin
3. from flask_sqlalchemy import SQLAlchemy
4. from prometheus_client import start_http_server, Summary, Gauge
5.
6. # สร้างแอป Flask
7. app = Flask(__name__)
8. # กำหนดการเชื่อมต่อฐานข้อมูล MySQL
9. app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://root:rootpassword@mysql:3306/python_api'
10. # สร้าง SQLAlchemy object เพื่อจัดการฐานข้อมูล
11. db = SQLAlchemy(app)
12.
13. # เปิดใช้งาน CORS สำหรับแอปนี้
14. cors = CORS(app)
15. app.config['CORS_HEADERS'] = 'Content-Type'
16.
17. # Prometheus Metrics
18. # Summary เพื่อวัดเวลาที่ใช้ในการประมวลผลคำขอ
19. REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')
20. # Gauge เพื่อวัดจำนวนรายการในฐานข้อมูล
21. ITEM_COUNT = Gauge('item_count', 'Number of items in the database')
22.
23. # สร้าง Model สำหรับตาราง 'Item'
24. class Item(db.Model):
25.     id = db.Column(db.Integer, primary_key=True)
26.     name = db.Column(db.String(100), nullable=False)
27.     price = db.Column(db.Float, nullable=False)
28.     currency = db.Column(db.String(10), nullable=False)
29.     quantity = db.Column(db.Integer, nullable=False)
30.
31. # สร้าง endpoint สำหรับสร้างรายการใหม่
32. @app.route('/items', methods=['POST'])
33. def create_item():
34.     data = request.json
35.     new_item = Item(
36.         name=data['name'],
37.         price=data['price'],
38.         currency=data['currency'],
39.         quantity=data['quantity']
40.     )
41.     db.session.add(new_item)
42.     db.session.commit()
43.     update_item_count() # อัปเดตจำนวนรายการในฐานข้อมูล
44.     return jsonify({'message': 'Item created successfully'}), 201
45.
46. # สร้าง endpoint สำหรับดึงรายการทั้งหมด
47. @app.route('/items', methods=['GET'])
48. @cross_origin()
49. def get_items():
50.     items = Item.query.all()
51.     items_list = [
52.         {
53.             'id': item.id,
54.             'name': item.name,
55.             'price': item.price,
56.             'currency': item.currency,
57.             'quantity': item.quantity
58.         }
59.         for item in items

```

```

60.     ]
61.     update_item_count() # อัปเดตจำนวนรายการในฐานข้อมูล
62.     return jsonify(items_list), 200
63.
64. # สร้าง endpoint สำหรับดึง metrics ของ Prometheus
65. @app.route('/metrics', methods=['GET'])
66. def metrics():
67.     from prometheus_client import generate_latest
68.     return generate_latest(), 200
69.
70. # ฟังก์ชันสำหรับอัปเดตจำนวนรายการในฐานข้อมูล
71. def update_item_count():
72.     count = Item.query.count()
73.     ITEM_COUNT.set(count)
74.
75. if __name__ == '__main__':
76.     with app.app_context():
77.         db.create_all() # สร้างตารางในฐานข้อมูลถ้ายังไม่มี
78.     start_http_server(8000) # เริ่มต้น HTTP server สำหรับ Prometheus metrics ที่ port 8000
79.     app.run(debug=True, host='0.0.0.0', port=5000) # รันแอป Flask ที่ host '0.0.0.0' และ port 5000

```

3) ทำการสร้าง ในส่วนของ docker-compose.yml

เหตุผลที่ต้องใช้ docker-compose.yml เพราะว่า จะช่วยให้จัดการ Container หลายๆตัวในแอปพลิเคชัน ได้อย่างง่าย กำหนดการเชื่อมต่อ ตั้งค่า config ที่ซับซ้อนต่างๆ
 สิ่งที่เราต้องติดตั้ง

1. Prometheus
2. Grafana
3. Mysql
4. Mysql-exporter (มีการ config ไฟล์ย่อยที่มีชื่อว่า my.cnf)
5. PhpMyAdmin
6. Flask-API

```

1. version: '3.7'
2.
3. services:
4.
5.     # บริการ MySQL
6.     mysql:
7.         image: mysql:latest # ใช้ภาพ MySQL เวอร์ชันล่าสุด
8.         container_name: mysql # ตั้งชื่อคอนเทนเนอร์เป็น mysql
9.         environment:
10.             MYSQL_ROOT_PASSWORD: rootpassword # กำหนดรหัสผ่านสำหรับ root
11.             MYSQL_DATABASE: python_api # สร้างฐานข้อมูลชื่อ python_api
12.         volumes:
13.             - mysql-data:/var/lib/mysql # เก็บข้อมูลใน volume ชื่อ mysql-data
14.         ports:
15.             - "3306:3306" # เปิดพอร์ต 3306
16.         restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
17.

```



```

18. # บริการ MySQL Exporter สำหรับ Prometheus
19. mysql-exporter:
20.   image: prom/mysql-exporter:latest # ใช้ภาพ mysql-exporter เวอร์ชันล่าสุด
21.   container_name: mysql-exporter # ตั้งชื่อคอนเทนเนอร์เป็น mysql-exporter
22.   environment:
23.     DATA_SOURCE_NAME: root:rootpassword@tcp(mysql:3306)/ # กำหนดข้อมูลการเชื่อมต่อ MySQL
24.     MYSQLD_EXPORTER_EXPORTER_DISABLE: "slave_status" # ปิดการส่งข้อมูล slave status
25.   command:
26.     - --config.my-cnf=/etc/mysql/my.cnf # ใช้ไฟล์ my.cnf เป็นการตั้งค่า
27.   ports:
28.     - "9104:9104" # เปิดพอร์ต 9104
29.   depends_on:
30.     - mysql # ขึ้นอยู่กับบริการ mysql
31.   volumes:
32.     - ./my.cnf:/etc/mysql/my.cnf # ใช้ไฟล์ my.cnf จากโฟลเดอร์ปัจจุบัน
33.   restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
34.
35. # บริการ phpMyAdmin
36. phpmyadmin:
37.   image: phpmyadmin/phpmyadmin:latest # ใช้ภาพ phpMyAdmin เวอร์ชันล่าสุด
38.   container_name: phpmyadmin # ตั้งชื่อคอนเทนเนอร์เป็น phpmyadmin
39.   environment:
40.     PMA_HOST: mysql # กำหนด host เป็น mysql
41.     MYSQL_ROOT_PASSWORD: rootpassword # กำหนดรหัสผ่านสำหรับ root
42.   ports:
43.     - "8080:80" # เปิดพอร์ต 8080
44.   depends_on:
45.     - mysql # ขึ้นอยู่กับบริการ mysql
46.   restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
47.
48. # บริการ Prometheus
49. prometheus:
50.   image: prom/prometheus:latest # ใช้ภาพ Prometheus เวอร์ชันล่าสุด
51.   container_name: prometheus # ตั้งชื่อคอนเทนเนอร์เป็น prometheus
52.   volumes:
53.     - ./prometheus.yml:/etc/prometheus/prometheus.yml # ใช้ไฟล์ prometheus.yml จากโฟลเดอร์ปัจจุบัน
54.   ports:
55.     - "9090:9090" # เปิดพอร์ต 9090
56.   command:
57.     - "--config.file=/etc/prometheus/prometheus.yml" # ใช้ไฟล์การตั้งค่า prometheus.yml
58.   depends_on:
59.     - mysql-exporter # ขึ้นอยู่กับบริการ mysql-exporter
60.   restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
61.
62. # บริการ Grafana
63. grafana:
64.   image: grafana/grafana:latest # ใช้ภาพ Grafana เวอร์ชันล่าสุด
65.   container_name: grafana # ตั้งชื่อคอนเทนเนอร์เป็น grafana
66.   ports:
67.     - "3000:3000" # เปิดพอร์ต 3000
68.   volumes:
69.     - grafana-storage:/var/lib/grafana # เก็บข้อมูลใน volume ชื่อ grafana-storage
70.   environment:
71.     - GF_SECURITY_ADMIN_PASSWORD=admin # กำหนดรหัสผ่านผู้ดูแลระบบ
72.   depends_on:
73.     - prometheus # ขึ้นอยู่กับบริการ prometheus
74.   restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
75.
76. # บริการ Flask API

```

```

77. flask-api:
78.   build: . # สร้างภาพจาก Dockerfile ในโฟลเดอร์ปัจจุบัน
79.   container_name: flask-api # ตั้งชื่อคอนเทนเนอร์เป็น flask-api
80.   environment:
81.     - SQLALCHEMY_DATABASE_URI=mysql+pymysql://root:rootpassword@mysql:3306/python_api # กำหนด
      URI สำหรับเชื่อมต่อฐานข้อมูล
82.   ports:
83.     - "5000:5000" # เปิดพอร์ต 5000
84.   depends_on:
85.     - mysql # ขึ้นอยู่กับบริการ mysql
86.   restart: always # รีสตาร์ทคอนเทนเนอร์อัตโนมัติเมื่อเกิดข้อผิดพลาด
87.   command: /bin/sh -c "sleep 10 && flask run --host=0.0.0.0" # หน่วงเวลาการรัน Flask 10 วินาทีเพื่อให้บริการอื่น
      ่พร้อมใช้งาน
88.
89. # กำหนด volumes ที่ใช้เก็บข้อมูล
90. volumes:
91.   mysql-data:
92.   grafana-storage:

```

จากนั้นเราจะสร้างไฟล์ my.cnf ที่จะกำหนด host,username,password ของ mysql ของเรา

ในส่วนของไฟล์ my.cnf

```

1. [client]
2. user=root
3. password=rootpassword
4. host=mysql

```

4) สร้างในส่วนของ prometheus.yml

จะต้อง config Prometheus โดยจะต้องสร้างไฟล์ prometheus.yml เพื่อใช้สำหรับการตั้งค่า

ใน Prometheus

```

1. global:
2.   scrape_interval: 15s # กำหนดช่วงเวลาเริ่มต้นในการดึงข้อมูล (scrape) ทุกๆ 15 วินาที
3.   scrape_configs:
4.     - job_name: 'prometheus' # ชื่อ job สำหรับ Prometheus
5.       scrape_interval: 5s # กำหนดช่วงเวลาในการดึงข้อมูล (scrape) สำหรับ job นี้ทุกๆ 5 วินาที
6.       static_configs:
7.         - targets: ['localhost:9090'] # กำหนดเป้าหมาย (target) คือ Prometheus เองที่รันอยู่บนพอร์ต 9090
8.
9.     - job_name: 'mysqld-exporter' # ชื่อ job สำหรับ MySQL exporter
10.      static_configs:
11.        - targets: ['mysqld-exporter:9104'] # กำหนดเป้าหมาย (target) คือ mysqld-exporter ที่รันอยู่บนพอร์ต 9104
12.
13.     - job_name: 'flask-api' # ชื่อ job สำหรับ Flask API
14.       metrics_path: /metrics # กำหนดเส้นทาง (path) สำหรับดึงข้อมูล metrics
15.       static_configs:
16.         - targets: ['flask-api:5000'] # กำหนดเป้าหมาย (target) คือ Flask API ที่รันอยู่บนพอร์ต 5000

```

5) ในส่วนของ dockerfile เพื่อสร้าง docker Image สำหรับ app flask

```

1. # Use the official Python image from the Docker Hub
2. FROM python:3.8-slim
3.
4. # Set the working directory in the container
5. WORKDIR /app
6.
7. # Copy the current directory contents into the container at /app
8. COPY . /app
9.
10. # Install any needed packages specified in requirements.txt
11. RUN pip install --no-cache-dir -r requirements.txt
12.
13. # Make port 5000 available to the world outside this container
14. EXPOSE 5000
15.
16. # Define environment variable
17. ENV FLASK_APP=app.py
18.
19. # Run app.py when the container launches
20. CMD ["flask", "run", "--host=0.0.0.0"]

```

6) ในส่วนของไฟล์ requirement.txt เพื่อระบุ dependencies ที่จำเป็นต้องใช้

```

1. Flask # ติดตั้ง Flask
2. flask_sqlalchemy # ติดตั้ง Flask SQLAlchemy
3. flask_cors # ติดตั้ง Flask CORS
4. pymysql # ติดตั้ง PyMySQL
5.
6. prometheus_client # ติดตั้ง Prometheus Client

```

7) จากนั้น ให้รัน ไฟล์ docker-compose.yml ที่ config ไว้

โดยใช้ คำสั่ง :

```
docker-compose up -d
```

ใน terminal หรือ power shell เมื่อรัน สำเร็จ จะได้ ข้อความลักษณะนี้

```

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>

```

- จากนั้นทำการเช็ค container บน docker ว่าทำงาน ทั้งหมดหรือไม่ โดยใช้คำสั่ง : `docker ps`








จะได้ลักษณะนี้ โดยดูที่ แถบของ status ว่า เป็น status up ไหม

```
PS D:\Coding\Python_API> docker ps
```

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

ถ้าเป็นเหมือนในรูปทั้งหมดแปลว่า container ของเราทำงานบน docker ได้แล้ว

- หรือสามารถดูได้จาก บน docker desktop ของเรา

<input type="checkbox"/>		python_api		Running (6/6)	0% 17 hours ago
<input type="checkbox"/>		mysqld-exporter	prom/mysqld-exporter:latest	Running	9104:9104 0% 17 hours ago
<input type="checkbox"/>		mysql	mysql:latest	Running	3306:3306 0% 17 hours ago
<input type="checkbox"/>		phpmyadmin	phpmyadmin/phpmyadmin:latest	Running	8080:80 0% 17 hours ago
<input type="checkbox"/>		flask-api	python_api-flask-api	Running	5000:5000 0% 17 hours ago
<input type="checkbox"/>		prometheus	prom/prometheus:latest	Running	9090:9090 0% 17 hours ago
<input type="checkbox"/>		grafana	grafana/grafana:latest	Running	3000:3000 0% 17 hours ago

ถ้า container ขึ้น status running เหมือนในรูปทั้งหมดแปลว่า container ของเราทำงานบน docker ได้แล้ว

8) เข้าไปที่ phpMyAdmin บน port 8080 ใน docker desktop ของเรา เพื่อสร้าง table


Containers [Give feedback](#)

Container CPU usage 1.43% / 800% (8 CPUs available) Container memory usage 868.27MB / 2.78GB [Show charts](#)

Search Only show running containers

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	grafana	grafana/grafana:lat	Running	3000:3000	0.11%	1 day ago	
<input type="checkbox"/>	promethe	prom/prometheus:l	Running	9090:9090	0.71%	1 day ago	
<input type="checkbox"/>	mysqld-e	prom/mysqld-expor	Running	9104:9104	0%	1 day ago	
<input type="checkbox"/>	flask-api	python_api-flask-api	Running	5000:5000	0.02%	1 day ago	
<input type="checkbox"/>	phpmyadr	phpmyadmin/phpmy	Running	8080:80	0.01%	1 day ago	
<input type="checkbox"/>	mysql	mysql:latest	Running	3306:3306	1.25%	1 day ago	

จากนั้นจะเข้ามาที่หน้าเว็บ localhost:8080



Welcome to phpMyAdmin

You have been automatically logged out due to inactivity of 1440 seconds. Once you log in again, you should be able to resume the work where you left off.

Language: English

Log in

Username: root

Password:

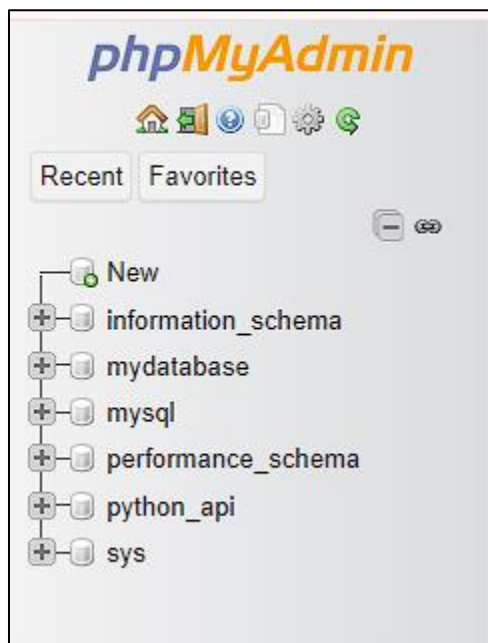
Log in

- จากนั้นให้กรอก username และ password ตามที่ config ในไฟล์ my.cnf คือ

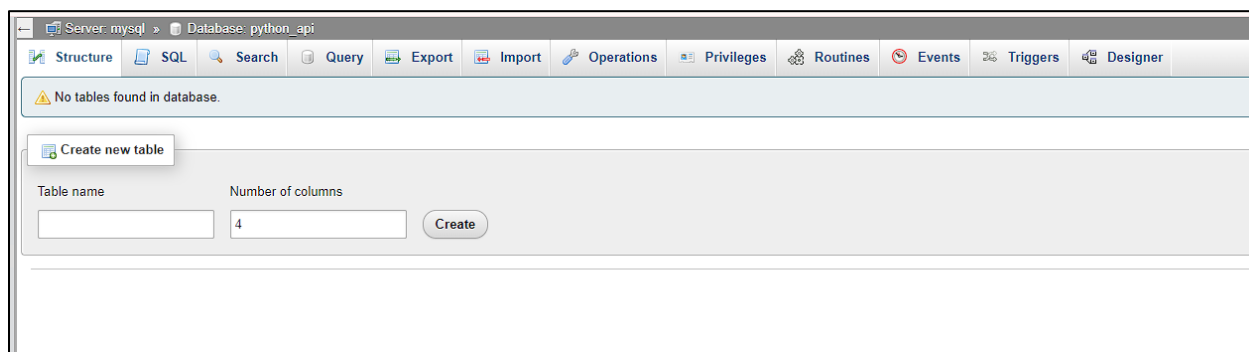
Username : root

Password : rootusername

- เมื่อเข้าสู่ระบบเรียบร้อยแล้วจะเจอกับหน้าตาแบบนี้ ให้ไปที่ database ชื่อ python_api



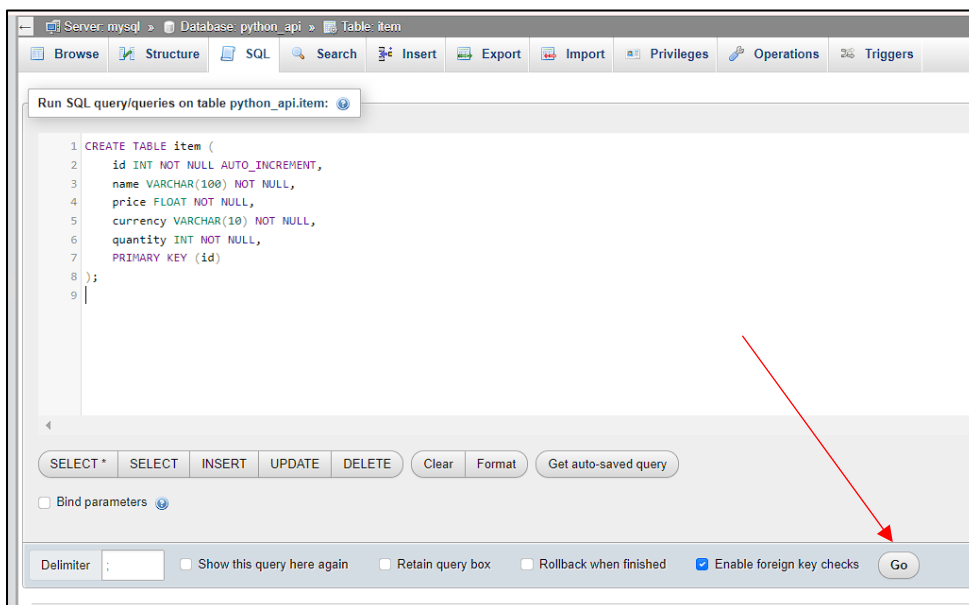
- จากนั้นให้ไปที่ แถบ SQL เพื่อสร้าง table ของ database นี้



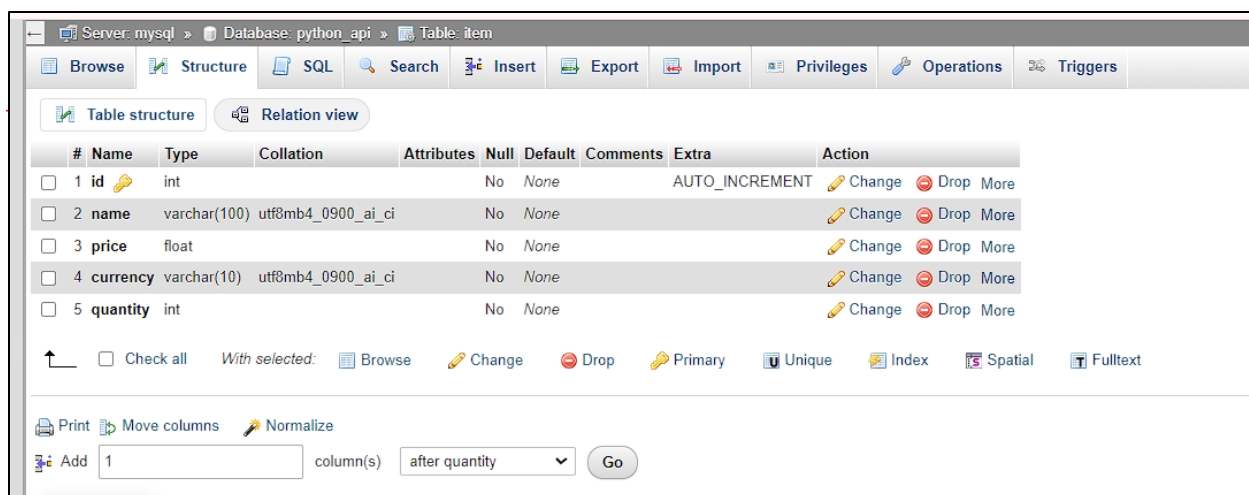
- จากนั้น ให้ใส่ คำสั่ง SQL เพื่อสร้าง table ตามนี้ :

```
1. CREATE TABLE item (
2.     id INT NOT NULL AUTO_INCREMENT,
3.     name VARCHAR(100) NOT NULL,
4.     price FLOAT NOT NULL,
5.     currency VARCHAR(10) NOT NULL,
6.     quantity INT NOT NULL,
7.     PRIMARY KEY (id)
8. );
9.
```

- จากนั้นกด go เพื่อสร้าง table

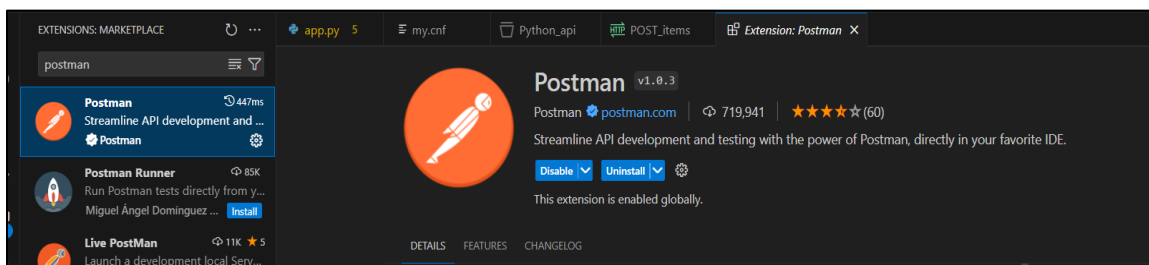


สามารถตรวจสอบ table ได้ว่าแต่ละตัวมีค่าที่ถูกตั้งไหม โดยไปที่ Structure

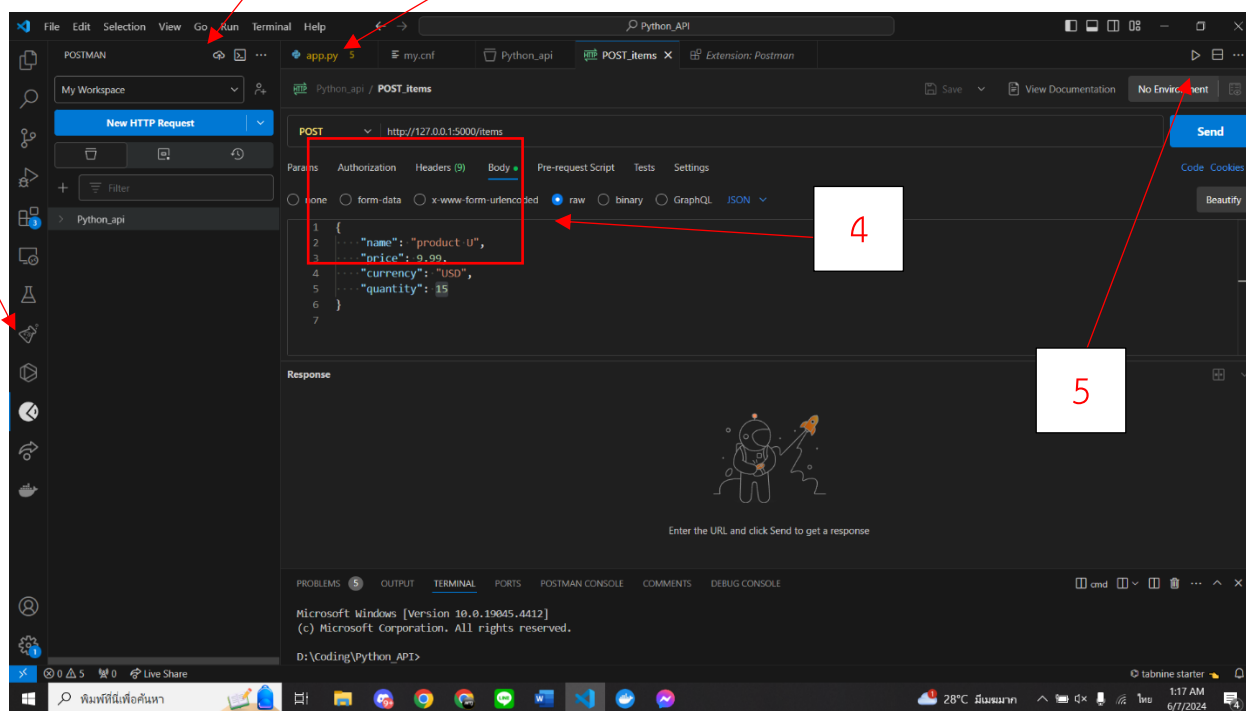


ถ้าได้เหมือนรูป เราก็สามารถ ทำการ POST dataไปที่ mysql ของเราได้แล้ว

- 9) ทำการ ทดสอบ ส่งและเก็บ data ลงไป mysql ของเรา ที่รันอยู่บน docker โดยใช้ postman
- โดยสามารถติดตั้ง postman ใน extension ใน visual studio code ได้เลย



1.เลือก icon postman ใน แถบ extension ทำการ login อะไรให้เรียบร้อย



2.เลือก New HTTP Request

3.เลือก เป็น POST

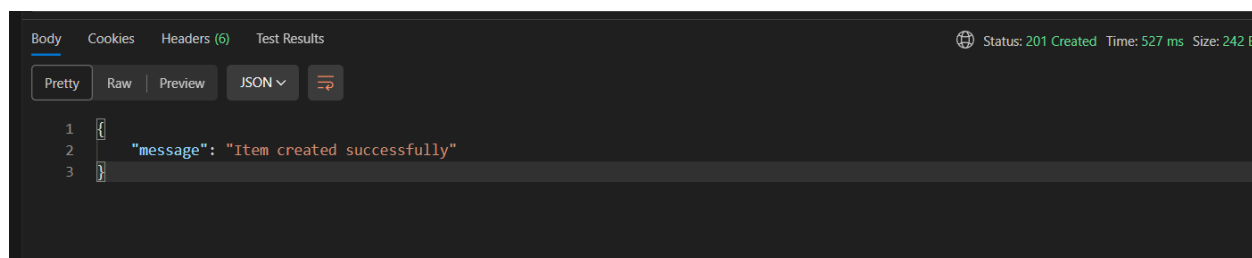
4. กรอก ข้อมูล เพิ่มลงใน database จาก table ที่สร้างไว้ ดังนี้ :

```
1. {
2.   "name": "product U", # ชื่อ product
3.   "price": 9.99,       # ราคา
4.   "currency": "USD",   # สกุลเงิน
5.   "quantity": 15       # จำนวน
6. }
```

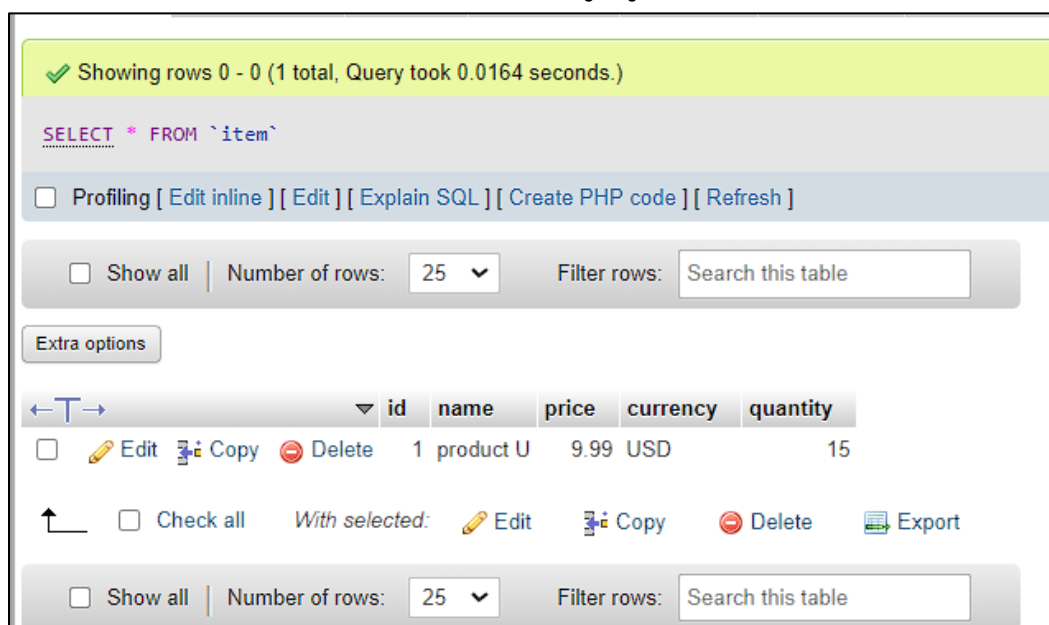
ปรับค่าได้ตามต้องการ

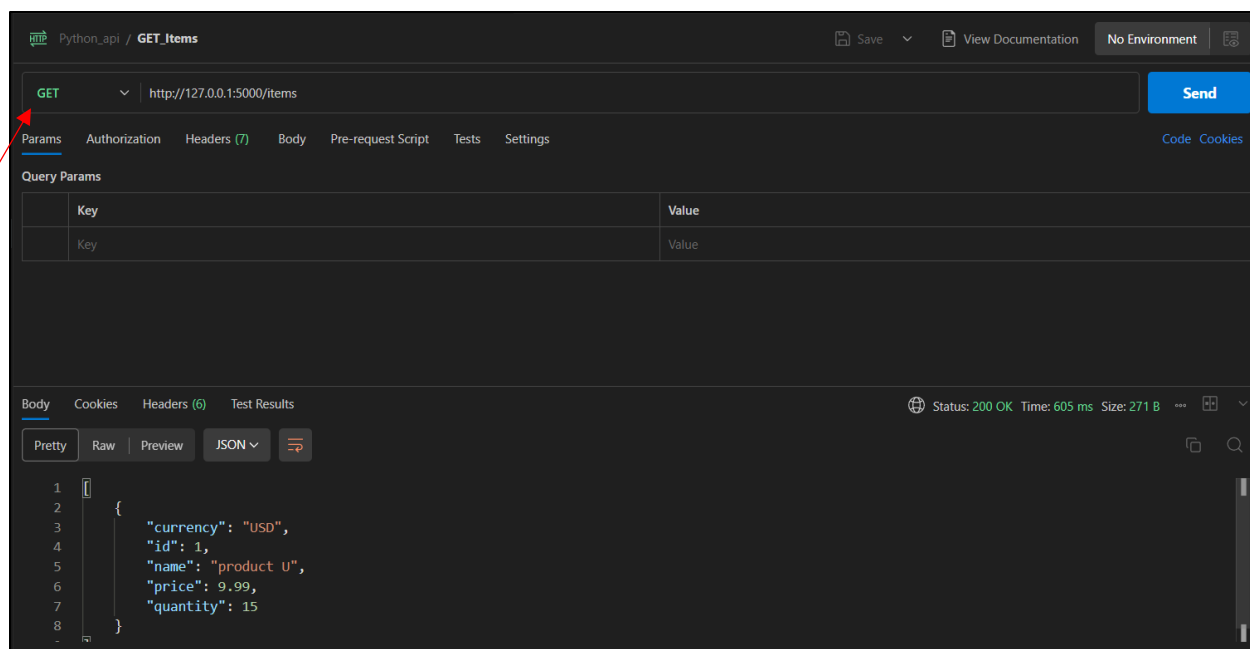
5.คลิก Send เพื่อ ส่งข้อมูลไปที่ database :

เมื่อส่งสำเร็จ จะขึ้นข้อความดังนี้



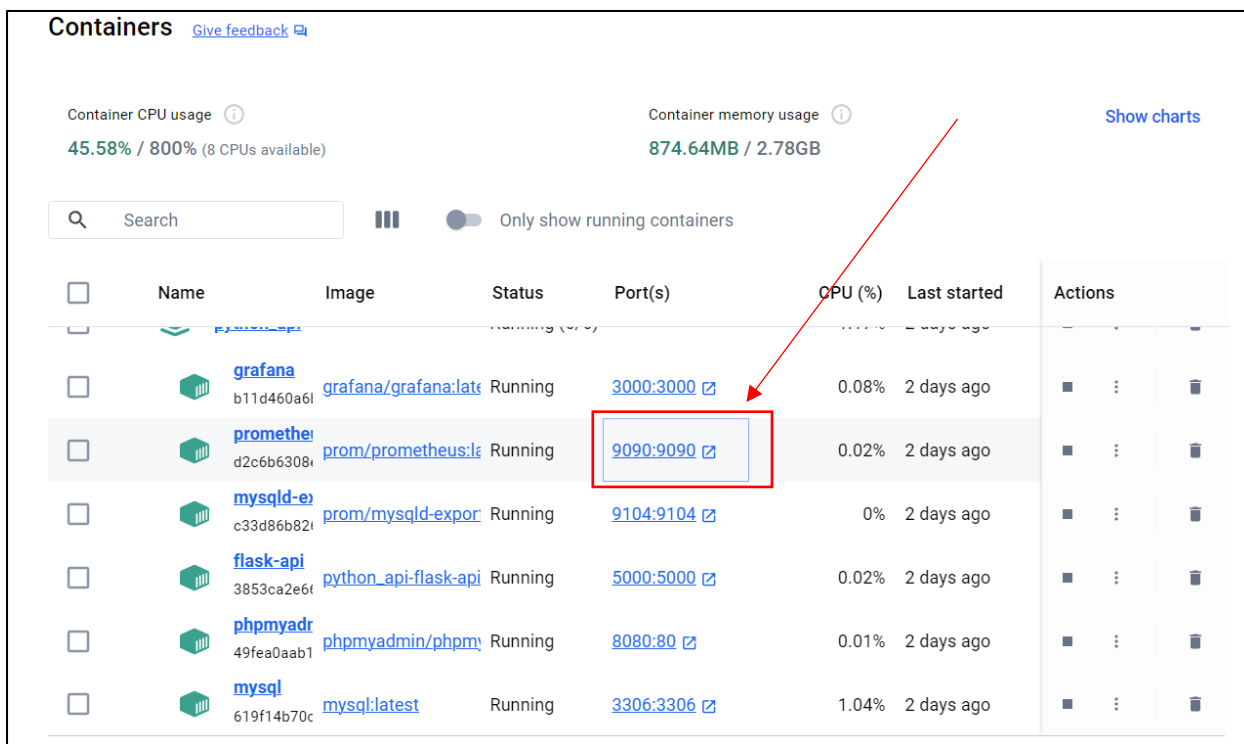
- จากไปที่ phpMyAdmin อีกครั้ง จะเห็นว่าข้อมูล ถูกเพิ่มมาที่ table ของเราแล้ว





- เมื่อกด Send ข้อมูลใน database ที่อยู่ใน database ของเราจะแสดงขึ้นมาว่ามีอะไรบ้าง ซึ่งจะ บอกแต่ละคอลัมน์ ของ table ที่เราสร้างซึ่งคือ
Id , name , currency , price , quantity

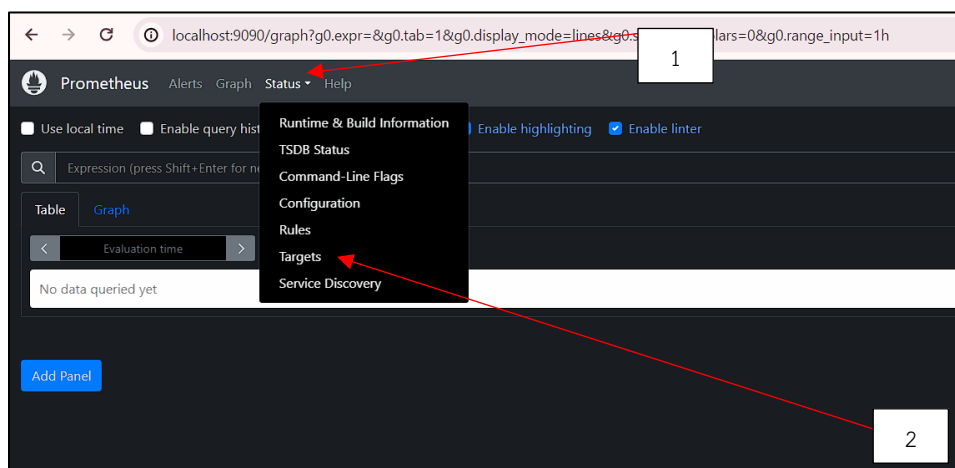
11) เข้าไปที่ Prometheus ของเรา จากใน docker desktop บน port 9090



เมื่อเราเข้ามาที่หน้าเว็บ ของ Prometheus ได้แล้ว

- เราจะทำการดู status endpoint ที่เราได้ config ไว้ในไฟล์ Prometheus.yml ว่าเป็น up ทั้งหมดไหม

โดยคลิกตามขั้นตอนในรูป



ถ้าขึ้นแบบนี้ หมายความว่า status เป็น up ทั้งหมด การดึงข้อมูล metrics ของ endpoint แต่ละตัว ถูกต้อง

The screenshot shows the Prometheus web interface at localhost:9090/targets?search=. The page displays a list of targets under the 'Targets' section. The targets are grouped by scrape pool: flask-api (1/1 up), mysql-exporter (1/1 up), and prometheus (1/1 up). Each target has a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The 'State' column for all targets is 'UP'. A red box highlights the 'State' column, and a red arrow points to the 'Last Scrape' column.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://flask-api:5000/metrics	UP	instance="flask-api:5000" job="flask-api"	8.483s ago	7.930ms	
http://mysql-exporter:9104/metrics	UP	instance="mysql-exporter:9104" job="mysql-exporter"	2.519s ago	108.687ms	
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	5.123s ago	10.981ms	

12) เข้าไปที่ Grafana บน docker desktop ของเรา บน port 3000

<input type="checkbox"/>		python_api		Running (6/6)		7.16%	2 days ago
<input checked="" type="checkbox"/>		grafana	grafana/grafana:latest	Running	3000:3000	0.11%	2 days ago
<input type="checkbox"/>		prometheus	prom/prometheus:latest	Running	9090:9090	5.81%	2 days ago
<input type="checkbox"/>		mysql-exporter	prom/mysql-exporter:latest	Running	9104:9104	0%	2 days ago
<input type="checkbox"/>		flask-api	python_api-flask-api	Running	5000:5000	0.03%	2 days ago
<input type="checkbox"/>		phpmyadmin	phpmyadmin/phpmyadmin:latest	Running	8080:80	0.01%	2 days ago
<input type="checkbox"/>		mysql	mysql:latest	Running	3306:3306	1.2%	2 days ago

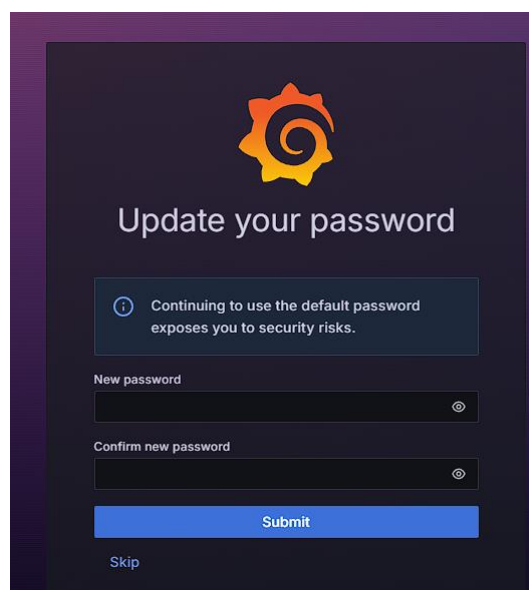
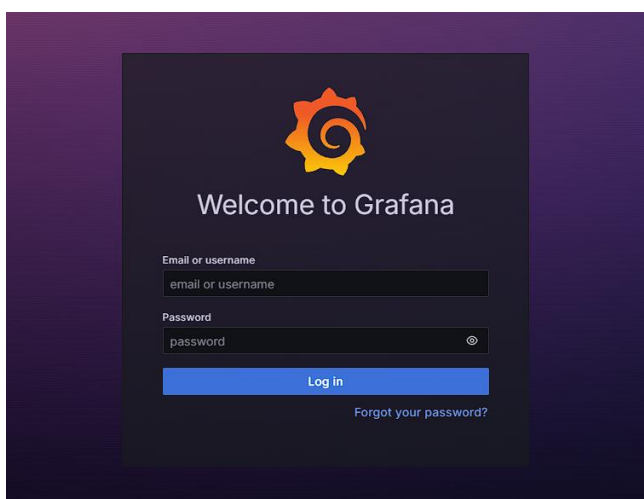
เมื่อเข้ามาแล้วจะพบกับหน้าต่าง login

- ให้กรอก

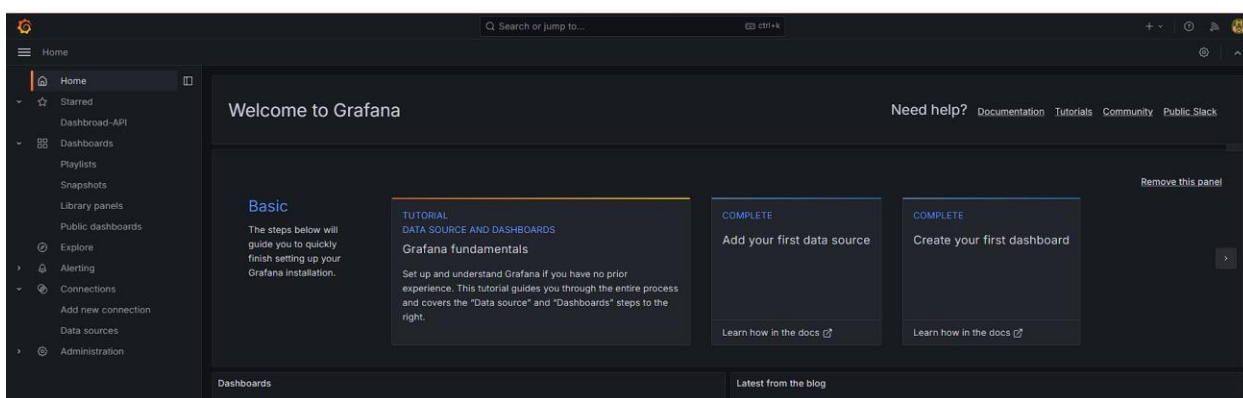
Username : admin

Password : admin

- เมื่อ login สำเร็จ ให้กด **skip** หน้า update password



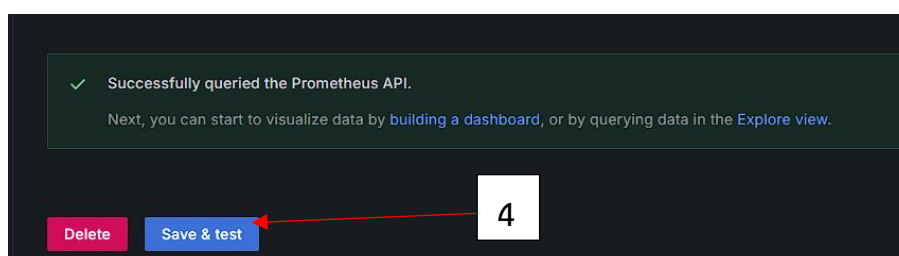
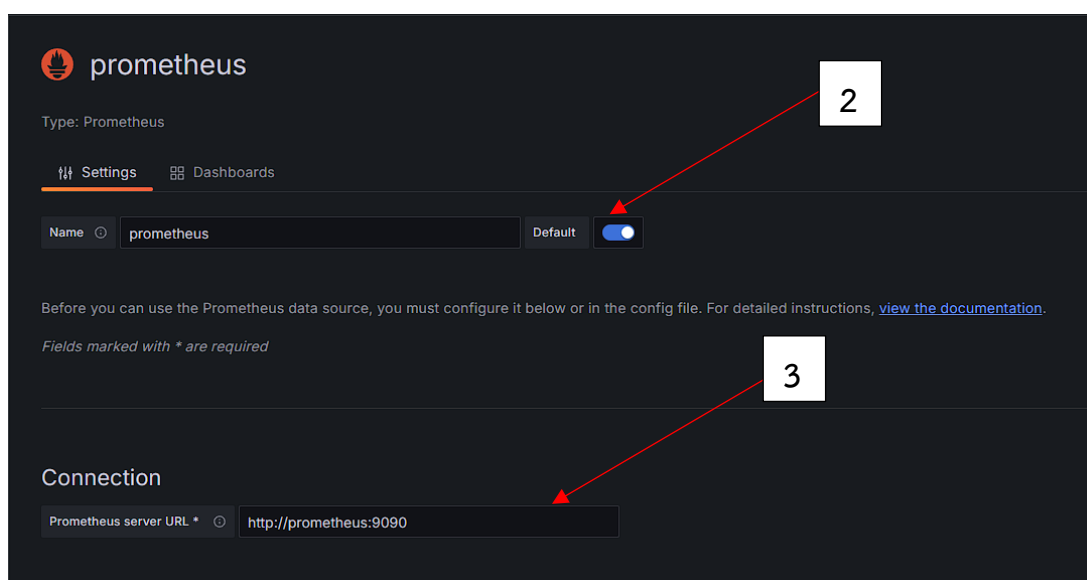
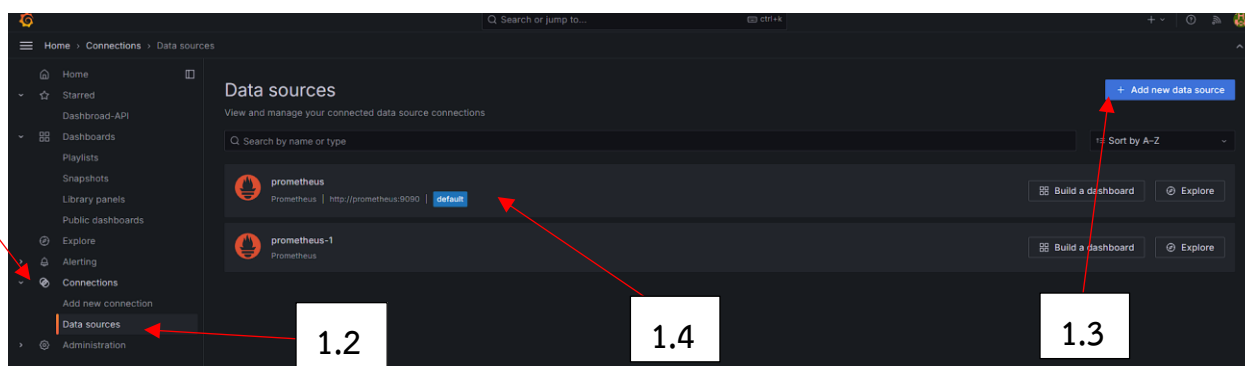
จากนั้นจะเจอกับหน้า Home Grafana



13) เริ่มสร้าง dashboard ใน Grafana ของเรา

- ตั้งค่าให้เลือกข้อมูลที่จะแสดงมาจาก Prometheus

1. ไปที่แถบเมนูฝั่งขวา คลิก **Connections > Data Sources > Add data source > Prometheus**
2. เลือก **Prometheus > เลือกให้เป็น Default**
3. ตั้งค่า URL เป็น <http://prometheus:9090> ในช่อง Connection
4. เลื่อนมาด้านล่างสุด คลิก **Save & Test** เพื่อทดสอบการเชื่อมต่อ

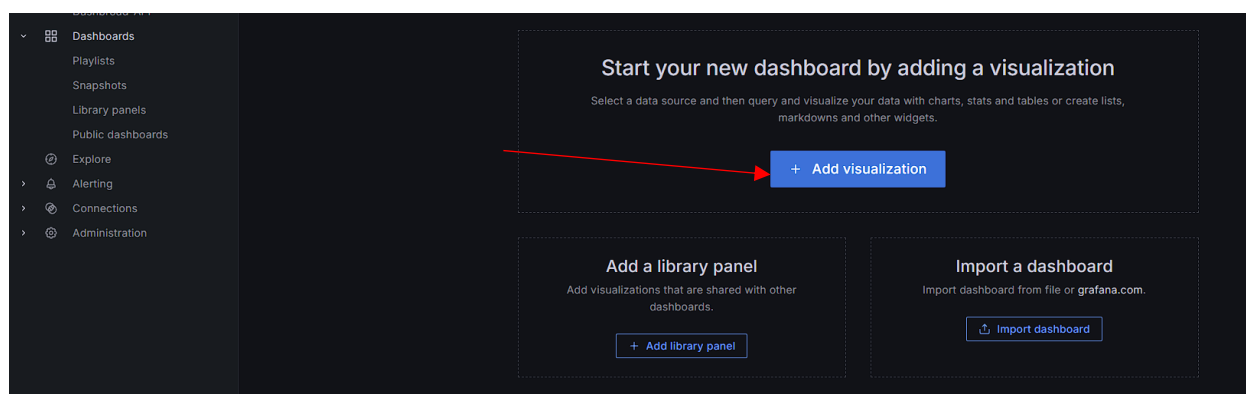


- สร้าง Flask API Request Duration Dashboard

วัตถุประสงค์: แสดงเวลาในการประมวลผลคำขอของ Flask API

1. ไปที่แถบเมนูฝั่งขวา เลือก **Dashboard**
2. คลิกไปที่ **Add visualization** > เลือก **Prometheus** ที่เป็น **default**
3. ตั้งชื่อ Panel เป็น "API Request Duration"
4. ใส่ Query :


```
1. rate(request_processing_seconds_sum[5m])
   /rate(request_processing_seconds_count[5m])
```
5. เลือกรูปแบบการแสดงผลเป็น **Table** บนแถบบนขวา
6. เลือกใน option ด้านล่าง เลือก format เป็น **Table**
7. กด Save



5

1

2

3

4

6

7

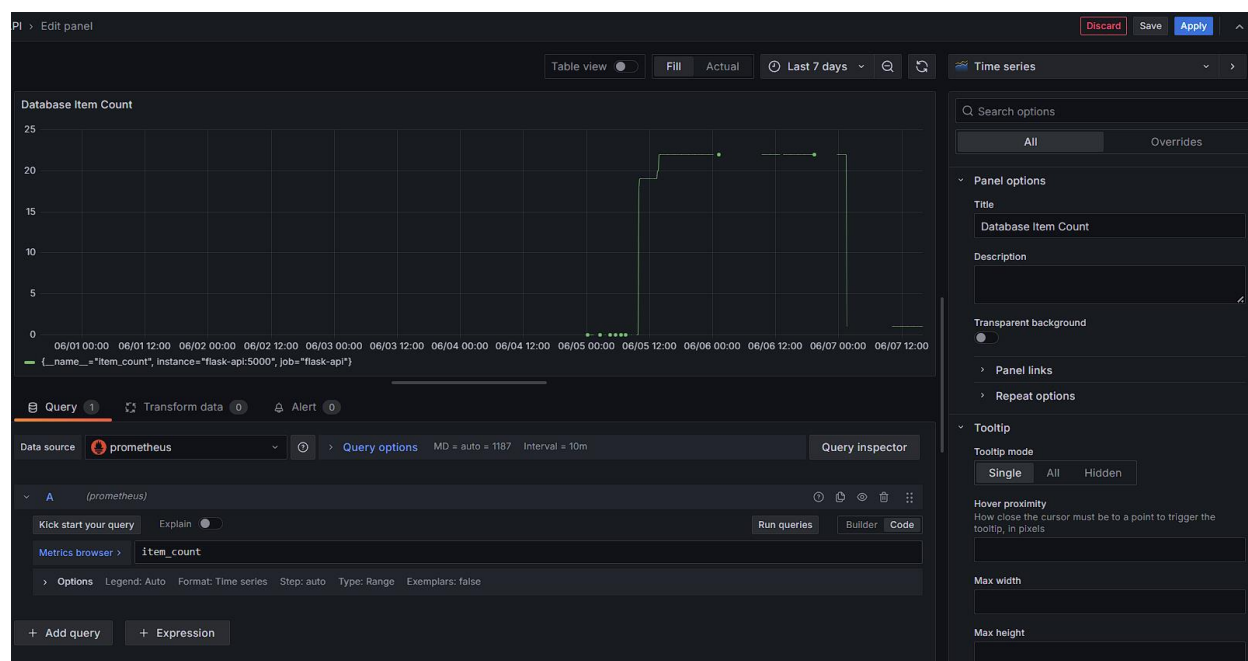
- สร้าง MySQL Database Item Count Dashboard

1. วัตถุประสงค์: แสดงจำนวนรายการในฐานข้อมูล MySQL
2. ไปที่แถบเมนูฝั่งขวา เลือก Dashboard
3. ไปที่มุมขวาบน คลิก New > Visualization > เลือก Prometheus ที่เป็น Default
4. ตั้งชื่อ Panel เป็น Database Item Count
5. ใส่ Query :

2. item_count

6. เลือกรูปแบบการแสดงผลเป็น Times series บนแถบบนขวา
7. กด Save

ขั้นตอนเหมือนกับการสร้าง Flask API Request Duration Dashboard ซึ่งจะได้กราฟลักษณะนี้



- สร้าง MySQL Connections,Uptime,Query per second Dashboard

- วัตถุประสงค์: ใช้สำหรับตรวจสอบจำนวนการเชื่อมต่อที่เปิดใช้งานอยู่ในปัจจุบันกับเซิร์ฟเวอร์ MySQL,
ใช้สำหรับตรวจสอบระยะเวลาที่เซิร์ฟเวอร์ MySQL ทำงานตั้งแต่ครั้งล่าสุดที่รีสตาร์ท,
ใช้สำหรับตรวจสอบอัตราการทำงานของคำสั่ง queries ต่อวินาทีในช่วงเวลาที่กำหนด
- ไปที่แถบเมนูฝั่งขวา เลือก Dashboard
- ไปที่มุมขวาด้านบน คลิก New > Visualization > เลือก Prometheus ที่เป็น Default
- ตั้งชื่อ Panel เป็น " MySQL Connections,Uptime,Query per second "
- ใส่ Query :

1. `mysql_global_status_threads_connected`

- กด Add query แล้วใส่ Query :

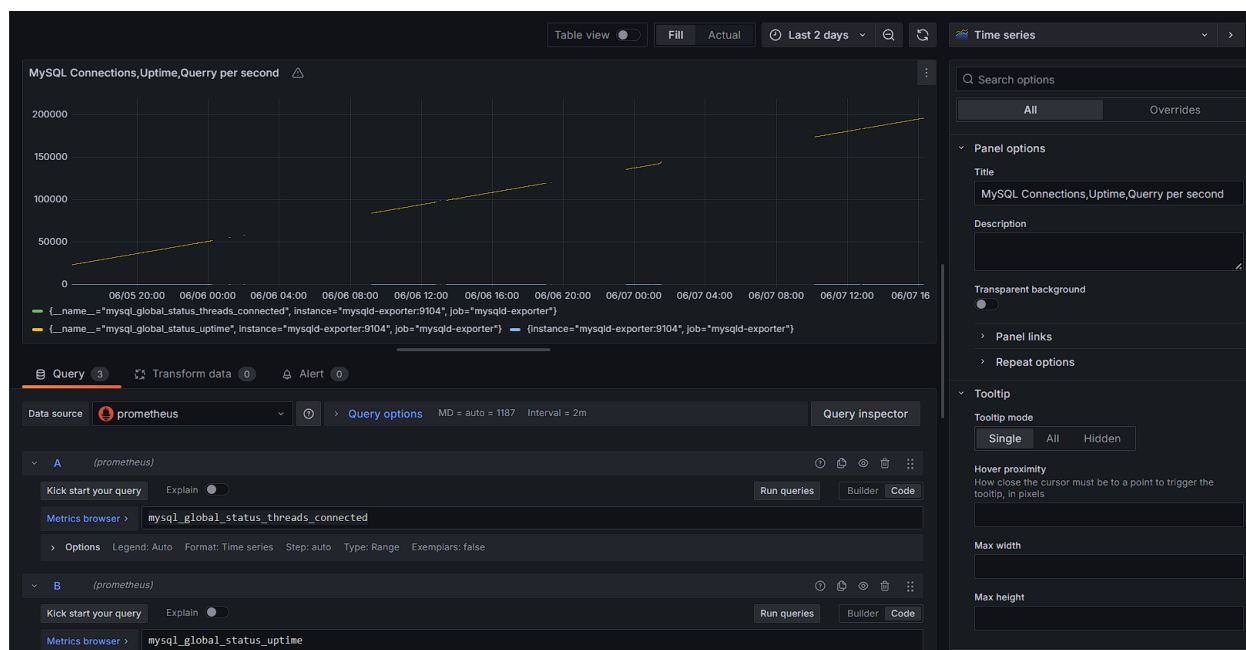
1. `mysql_global_status_uptime`

- กด Add query แล้วใส่ Query :

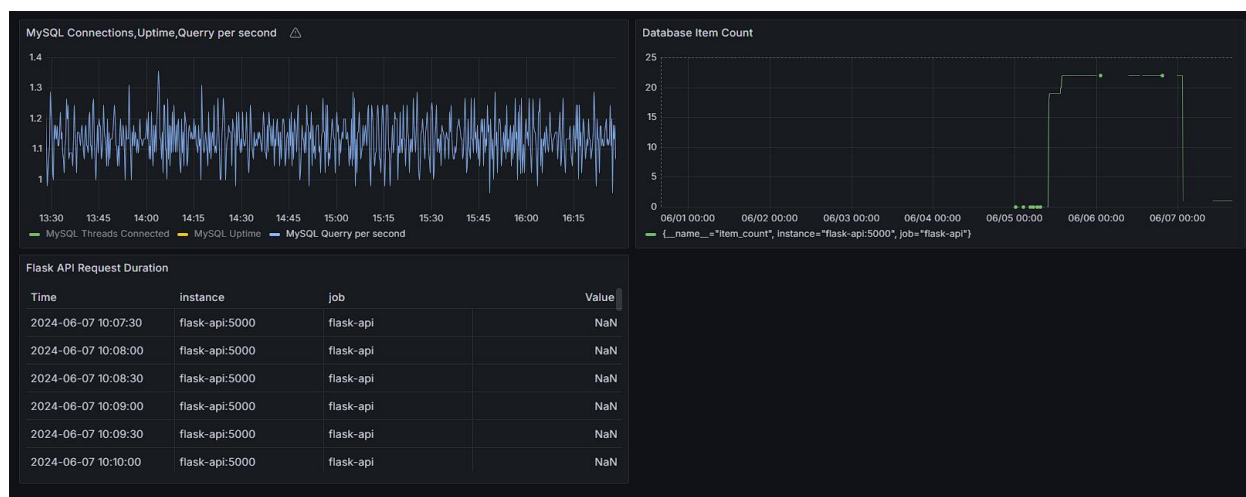
1. `rate(mysql_global_status_queries[1m])`

- เลือกรูปแบบการแสดงผลเป็น Times series บนแถบบนขวา
- กด Save

ขั้นตอนเหมือนกับ การสร้าง Flask API Request Duration Dashboard ซึ่งจะได้
กราฟลักษณะนี้



-เป็นอันเสร็จเรียบร้อย กราฟที่ได้ทั้งหมดจะขึ้นดังนี้



- กราฟแรก จะเป็น MySQL Connections,Uptime,Query per second

ซึ่งสามารถเลือกดูแต่ละกราฟได้ โดยคลิกที่ชื่อ ที่มีแถบสี ในกรอบสีแดงตามรูป

คำสั่ง Query : `mysql_global_status_threads_connected:`

- **วัตถุประสงค์:** ใช้สำหรับตรวจสอบจำนวนการเชื่อมต่อที่เปิดใช้งานอยู่ในปัจจุบันกับเซิร์ฟเวอร์ MySQL
- **คำอธิบาย:** คำสั่งนี้จะดึงข้อมูลเกี่ยวกับจำนวน threads ที่กำลังเชื่อมต่ออยู่ในขณะนี้ ซึ่งมีประโยชน์ในการตรวจสอบว่ามีการใช้งานเซิร์ฟเวอร์มากน้อยเพียงใดและสามารถใช้ในการวางแผนขยายระบบหรือแก้ไขปัญหาที่เกี่ยวข้องกับการเชื่อมต่อ

คำสั่ง Query : `mysql_global_status_uptime:`

- **วัตถุประสงค์:** ใช้สำหรับตรวจสอบระยะเวลาที่เซิร์ฟเวอร์ MySQL ทำงานตั้งแต่ครั้งล่าสุดที่รีสตาร์ท
- **คำอธิบาย:** คำสั่งนี้จะดึงข้อมูล uptime ของเซิร์ฟเวอร์ MySQL ซึ่งสามารถช่วยให้ผู้ดูแลระบบรู้ว่าเซิร์ฟเวอร์ทำงานต่อเนื่องนานแค่ไหนแล้ว และใช้เป็นตัวบ่งชี้ในการวางแผนการบำรุงรักษาหรือรีสตาร์ทเซิร์ฟเวอร์

คำสั่ง Query : `rate(mysql_global_status_queries[1m]):`

- **วัตถุประสงค์:** ใช้สำหรับตรวจสอบอัตราการทำงานของคำสั่ง queries ต่อวินาทีในช่วงเวลาที่กำหนด
- **คำอธิบาย:** คำสั่งนี้จะดึงข้อมูลเกี่ยวกับจำนวนคำสั่ง queries ที่ถูกประมวลผลในช่วงเวลาหนึ่ง (1 นาที) และคำนวณอัตราการทำงานของ queries ต่อวินาที การใช้คำสั่งนี้ช่วยให้ผู้ดูแลระบบสามารถติดตามประสิทธิภาพการทำงานของเซิร์ฟเวอร์ MySQL และตรวจสอบว่าเซิร์ฟเวอร์มีการทำงานอย่างมีประสิทธิภาพหรือไม่

- กราฟที่ 2 Database item count

คำสั่ง Query : item_count

- **วัตถุประสงค์:** ใช้เพื่อดึงจำนวนรายการทั้งหมดในระบบหรือฐานข้อมูล โดยสามารถใช้เพื่อตรวจสอบการเปลี่ยนแปลงของจำนวนรายการตลอดเวลา
- **คำอธิบาย:** คำสั่งนี้จะดึงข้อมูลจำนวนรายการทั้งหมดที่มีอยู่ในระบบหรือฐานข้อมูลปัจจุบัน ซึ่งสามารถใช้เพื่อการมอนิเตอร์การเติบโตของข้อมูล การตรวจสอบปัญหาที่อาจเกิดขึ้นเมื่อมีการเพิ่มหรือลดของรายการในระบบ

- กราฟที่ 3 Flask API Request Duration

คำสั่ง Query : $\text{rate}(\text{request_processing_seconds_sum}[5m]) / \text{rate}(\text{request_processing_seconds_count}[5m])$

วัตถุประสงค์: ใช้เพื่อคำนวณค่าเฉลี่ยของเวลาในการประมวลผลคำขอ (request) ของระบบในช่วงเวลาที่กำหนด (ในที่นี้คือ 5 นาที)

คำอธิบาย:

- คำสั่งนี้ประกอบด้วยสองส่วน:
 - $\text{rate}(\text{request_processing_seconds_sum}[5m])$: คำนวณอัตราการเพิ่มขึ้นของผลรวมเวลาที่ใช้ในการประมวลผลคำขอทั้งหมดในช่วง 5 นาที
 - $\text{rate}(\text{request_processing_seconds_count}[5m])$: คำนวณอัตราการเพิ่มขึ้นของจำนวนคำขอที่ประมวลผลในช่วง 5 นาที
- ผลลัพธ์ที่ได้จากการหารค่าของทั้งสองส่วนจะให้ค่าเฉลี่ยของเวลาในการประมวลผลคำขอในช่วง 5 นาที ซึ่งสามารถใช้เพื่อตรวจสอบประสิทธิภาพของระบบและการตอบสนองของเซิร์ฟเวอร์

สามารถอ่าน วิธีการเพิ่มติมได้ที่ ไฟล์ REAME.md ได้ที่ Link GitHub :

<https://github.com/YannawutRoumsuk/Prometheus-and-Grfana.git>

อ้างอิง

Docker Documentation. (n.d.). Retrieved from <https://docs.docker.com>

Prometheus Documentation. (n.d.). Getting Started. Retrieved from https://prometheus.io/docs/prometheus/latest/getting_started

Grafana Documentation. (n.d.). Getting Started. Retrieved from <https://grafana.com/docs/grafana/latest/getting-started>

Docker Forums. (n.d.). Retrieved from <https://docs.docker.com>