

การออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน และกู้คืนเว็บแอปพลิเคชัน
ภายในระบบการประมวลผลแบบกลุ่มเมฆ ด้วยซอฟต์แวร์โอเพนซอร์ซ

ณัฐพล ศรีรอดบาง

TNI

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
บัณฑิตวิทยาลัย สาขาวิชาเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีไทย-ญี่ปุ่น
ปีการศึกษา 2563

A DESIGN OF VIRTUAL SERVER STATUS MONITORING SYSTEM AND WEB APPLICATION
RECOVERY IN CLOUD COMPUTING SYSTEM USING OPEN SOURCE SOFTWARE

Nutthapon Srirodbang

TNI

A Term Paper Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Information Technology

Graduate School

Thai-Nichi Institute of Technology

Academic Year 2020

หัวข้อสารนิพนธ์

การออกแบบระบบติดตามสถานภาพเครื่องแม่ข่าย
เสมือน และกู้คืนเว็บแอปพลิเคชัน ภายในระบบการ
ประมวลผลแบบกลุ่มเมฆ ด้วยซอฟต์แวร์โอเพนซอร์ซ

โดย

ณัฐพล ศรีรอดบาง

สาขาวิชา

เทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาสารนิพนธ์

รองศาสตราจารย์ ดร. อรรถณพ หมั่นสกุล

บัณฑิตวิทยาลัย สถาบันเทคโนโลยีไทย-ญี่ปุ่น อนุมัติให้รับสารนิพนธ์ฉบับนี้เป็นส่วนหนึ่งของ
การศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต

.....คณบดีบัณฑิตวิทยาลัย

(รองศาสตราจารย์ ดร. พิชิต สุขเจริญพงษ์)

วันที่.....เดือน.....พ.ศ.....

คณะกรรมการสอบสารนิพนธ์

.....ประธานกรรมการ

(ดร. สะพรั่งสิทธิ์ มฤตสาร)

.....กรรมการ

(ดร. สรмыพร เจริญพิทย์)

.....อาจารย์ที่ปรึกษาสารนิพนธ์

(รองศาสตราจารย์ ดร. อรรถณพ หมั่นสกุล)

ณัฐพล ศรีรอดบาง : การออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน และกู้คืนเว็บแอปพลิเคชัน ภายในระบบการประมวลผลแบบกลุ่มเมฆ ด้วยซอฟต์แวร์โอเพนซอร์ซ.
อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร. อรรณพ หมั่นสกุล, 115 หน้า.

การออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน และกู้คืนเว็บแอปพลิเคชัน ภายในระบบการประมวลผลแบบกลุ่มเมฆ ด้วยซอฟต์แวร์โอเพนซอร์ซ มีวัตถุประสงค์ในการศึกษา ได้แก่ 1) ออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน 2) เพื่อให้มีระบบแจ้งเตือนผู้เกี่ยวข้องให้ทราบ เมื่อเครื่องแม่ข่ายเสมือนทำงานผิดปกติ 3) เพื่อให้มีระบบตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันภายในเครื่องแม่ข่ายเสมือน โดยระบบที่ออกแบบประกอบด้วย 4 ส่วนคือ 1) ส่วนฐานข้อมูลอนุกรมเวลาโดยใช้ซอฟต์แวร์ Prometheus 2) ส่วนชุดคำสั่งตรวจสอบสถานภาพเครื่องแม่ข่าย และเว็บแอปพลิเคชันโดยใช้โพรโทคอล SNMP, ICPM, TCP และ HTTP ร่วมกับชุดคำสั่งที่พัฒนาด้วยภาษาไพทอน 3) ส่วนแสดงผล และตรวจสอบสถานภาพโดยใช้ซอฟต์แวร์ Grafana 4) ส่วนชุดคำสั่งตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชัน ที่พัฒนาด้วยภาษาไพทอน

การออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน และกู้คืนเว็บแอปพลิเคชัน ภายในระบบการประมวลผลแบบกลุ่มเมฆ ด้วยซอฟต์แวร์โอเพนซอร์ซในครั้งนี้ จากการทดสอบส่วนติดตามสถานภาพ โดยการจำลองสถานการณ์ที่เครื่องแม่ข่ายเสมือน และเว็บแอปพลิเคชันมีสถานภาพผิดปกติ ระบบสามารถตรวจสอบพบความผิดปกติ แล้วแจ้งเตือนไปยังผู้ดูแลระบบผ่านแอปพลิเคชัน LINE ได้ทุกครั้งที่ทดสอบ ในการทดสอบส่วนตรวจสอบ และกู้คืนเว็บแอปพลิเคชัน จากการจำลองสถานการณ์ความผิดปกติ พบว่าสามารถตรวจสอบปัญหา แล้วกู้คืนส่วนประกอบของเว็บแอปพลิเคชันให้กลับมาทำงานได้ปกติ และมีการแจ้งผลลัพธ์ไปยังผู้ดูแลระบบผ่านแอปพลิเคชัน LINE ได้ทุกครั้งที่ทดสอบ

TNI

NUTTHAPON SRIRODBANG: A Design of Virtual Server Status Monitoring System and Web Application Recovery in Cloud Computing System Using Open Source Software. ADVISOR: ASSOC.PROF. DR. ANNOP MONSAKUL, 115 PP.

The objectives of this study were 1) to design a virtual server status monitoring system 2) to create a system to notify relevant persons in case of the malfunctioned virtual server, and 3) to develop a virtual server status monitoring and web application recovery system. The designed system consisted of 4 parts: 1) a time series database using Prometheus software, 2) scripts for virtual server status and web application monitoring included SNMP, ICPM, TCP, and HTTP protocols with Python programming language, 3) monitoring status display using Grafana software, and 4) scripts for web application diagnosis and recovery developed with Python programming language.

The results of testing monitoring system indicated that the situation simulation at the virtual server and the web application displayed an abnormal status. The designed system could detect abnormalities and the results were notified to the system administrator via the LINE application every time when testing. In terms of web application diagnosis and recovery system, from simulating the anomaly situation, the system could detect the problem and recover the web application components to normal operation. In addition, the results could be notified to the system administrator through the LINE application every time.

TNI

Graduate School

Field of Study Information Technology

Academic Year 2020

Student's Signature.....

Advisor's Signature.....

กิตติกรรมประกาศ

การทำสารนิพนธ์ในครั้งนี้สำเร็จลงได้นั้น ผู้ศึกษาขอกราบขอบพระคุณในความกรุณาของ รองศาสตราจารย์ ดร. อรรถนพ หมั่นสกุล ที่ได้เสียสละเวลาในการให้คำปรึกษา และคำแนะนำตลอดระยะเวลาในการดำเนินการศึกษา อีกทั้งยังขอกราบขอบพระคุณ ดร. บุษราพร เหลืองมาลาวัฒน์ ในการชี้แนะ และให้คำปรึกษาในการเริ่มต้นการทำสารนิพนธ์

ขอขอบพระคุณคณะกรรมการทุกท่าน ได้แก่ อาจารย์ ดร. สะพรั่งสิทธิ์ มฤตุสาธร์ และอาจารย์ ดร. สรสมัยพร เจริญพิทย์ ที่ได้กรุณาให้ข้อเสนอแนะที่เป็นประโยชน์ และช่วยชี้แนะการปรับปรุงข้อบกพร่องของสารนิพนธ์ฉบับนี้ให้สมบูรณ์ และครบถ้วน

ขอกราบขอบพระคุณบิดามารดา และครอบครัวที่คอยส่งเสริมให้กำลังใจผู้ศึกษาตลอดระยะเวลาการศึกษาจนสามารถจัดทำสารนิพนธ์นี้ให้สำเร็จลุล่วง

ขอขอบคุณ คุณพรไพลิน พี่ๆ เพื่อนๆ น้องๆ และทีม Monitor ที่ให้คำแนะนำ ความรู้ และโอกาสจนสามารถจัดทำสารนิพนธ์นี้ให้สำเร็จลุล่วง

ณัฐพล ศรีรอดบาง

TNI

THAI - NICHI INSTITUTE OF TECHNOLOGY

สารบัญ

	หน้า
บทคัดย่อภาษาไทย.....	ง
บทคัดย่อภาษาอังกฤษ.....	จ
กิตติกรรมประกาศ.....	ฉ
สารบัญ.....	ช
สารบัญตาราง.....	ฌ
สารบัญรูป.....	ญ
บทที่	
1 บทนำ.....	1
1.1 ความเป็นมาและความสำคัญของปัญหา.....	1
1.2 วัตถุประสงค์การศึกษา.....	2
1.3 ขอบเขตของการศึกษา.....	3
1.4 ประโยชน์ที่คาดว่าจะได้รับ.....	3
1.5 นิยามคำศัพท์.....	3
2 ทฤษฎี เทคโนโลยี และงานวิจัยที่เกี่ยวข้อง.....	5
2.1 เทคโนโลยีการประมวลผลแบบกลุ่มเมฆ.....	5
2.2 โพรโทคอล Simple Network Management Protocol.....	6
2.3 ภาษาโปรแกรมไพทอน.....	8
2.4 ฐานข้อมูลอนุกรมเวลา Prometheus.....	8
2.5 เครื่องมือเฝ้าติดตาม Grafana.....	12
2.6 งานวิจัยที่เกี่ยวข้อง.....	15
3 วิธีดำเนินการศึกษา.....	18
3.1 แผนการดำเนินงาน.....	18
3.2 ภาพรวมการทำงานของระบบ.....	19
3.3 รายละเอียดการดำเนินงาน.....	20
3.4 การออกแบบการทดสอบ.....	60

สารบัญ (ต่อ)

บทที่	หน้า
4 ผลการทดลอง	71
4.1 ความถูกต้องแม่นยำของค่าข้อมูลสถานภาพ.....	71
4.2 ปริมาณการใช้ทรัพยากร	77
4.3 การแจ้งเตือนความผิดปกติ	81
4.4 เวลาในการตอบสนองของส่วนติดตามสถานภาพ	82
4.5 การตรวจสอบปัญหา และกู้คืนระบบ	82
5 สรุป อภิปรายผล และข้อเสนอแนะ	85
5.1 สรุปผล และอภิปรายผล	85
5.2 ข้อเสนอแนะ	86
5.3 ประโยชน์ที่ได้รับจากการทำสารนิพนธ์.....	86
บรรณานุกรม	88
ภาคผนวก.....	91
ภาคผนวก ก ชุดคำสั่งตรวจสอบสถานภาพเครื่องแม่ข่ายเสมือน	92
ภาคผนวก ข ชุดคำสั่งตรวจสอบ และกู้คืนเว็บแอปพลิเคชัน.....	101
ประวัติย่อผู้เขียนสารนิพนธ์.....	115

สารบัญตาราง

ตาราง	หน้า
1.1 รายการกลุ่มสินทรัพย์ และการประเมินความเสี่ยง	2
2.1 การเปรียบเทียบคุณสมบัติ และประสิทธิภาพเครื่องมือฐานข้อมูลอนุกรมเวลา	10
2.2 การเปรียบเทียบคะแนนเครื่องมือฐานข้อมูลอนุกรมเวลา.....	11
2.3 การเปรียบเทียบคุณสมบัติ และประสิทธิภาพเครื่องมือติดตามสถานภาพ	13
2.4 การเปรียบเทียบคะแนนเครื่องมือติดตามสถานภาพ.....	14
2.5 การเปรียบเทียบประสิทธิภาพระบบของงานวิจัยที่เกี่ยวข้อง.....	17
3.1 แผนการดำเนินงาน	18
3.2 ค่า OID ของข้อมูลสถานภาพ สำหรับระบบปฏิบัติการ Ubuntu 18.04.....	25
3.3 การประมวลผลค่าข้อมูลสถานภาพ	25
3.4 การแสดงผลสถานภาพ และเงื่อนไขการแจ้งเตือน.....	36
3.5 รายการปัญหาที่สามารถใช้ชุดคำสั่งช่วยแก้ไขปัญหา หรือตรวจสอบเบื้องต้น.....	45
3.6 การจำแนกค่าความผิดปกติ ส่วนที่เกี่ยวข้อง และการดำเนินการ.....	46
4.1 ตารางเปรียบเทียบค่าเปอร์เซ็นต์ซีพียู	72
4.2 ตารางเปรียบเทียบค่าหน่วยความจำหลัก	73
4.3 ตารางเปรียบเทียบค่าหน่วยความจำสำรอง	74
4.4 ตารางเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่าย	75
4.5 ตารางเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่าย	76
4.6 ตารางเปรียบเทียบค่าเปอร์เซ็นต์ซีพียูของฐานข้อมูลอนุกรมเวลา.....	78
4.7 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้หน่วยความจำสำรองของฐานข้อมูลอนุกรมเวลา.....	78
4.8 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของเครื่องมือติดตามสถานภาพ.....	80
4.9 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้หน่วยความจำหลักของเครื่องมือติดตามสถานภาพ...	81
4.10 ผลการทดสอบประสิทธิภาพการตรวจสอบสถานภาพ และการแจ้งเตือน	81
4.11 เวลาในการตอบสนองของส่วนติดตามสถานภาพ	82
4.12 ผลการทดสอบอัตราความสำเร็จในการกู้คืนระบบ.....	83
4.13 ผลการเปรียบเทียบเวลาในการกู้คืนระบบ	84

สารบัญรูป

รูป	หน้า
2.1 รูปแบบการให้บริการการประมวลผลแบบกลุ่มเมฆ	6
2.2 องค์ประกอบของโพรโทคอล SNMP.....	7
2.3 โครงสร้างการทำงานของ Prometheus	9
2.4 ตัวอย่างการแสดงผลของ Grafana.....	12
3.1 ภาพรวมการทำงานของระบบ	19
3.2 หน้าเว็บ Prometheus	21
3.3 หน้าเว็บลงชื่อเข้าใช้ Grafana.....	22
3.4 การตั้งค่าเชื่อมต่อ Grafana กับ Prometheus	23
3.5 ขั้นตอนการทำงานของชุดคำสั่งตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือน	24
3.6 ชุดคำสั่งฟังก์ชัน Web APIs.....	26
3.7 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน ping	27
3.8 ชุดคำสั่งฟังก์ชัน ping.....	28
3.9 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน SNMP.....	29
3.10 ชุดคำสั่งฟังก์ชัน SNMP	30
3.11 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน port	32
3.12 ชุดคำสั่งฟังก์ชัน port.....	32
3.13 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน http	33
3.14 ชุดคำสั่งฟังก์ชัน http.....	34
3.15 ตัวอย่างการแสดงผลของส่วน Status ค่าปกติ และค่าผิดปกติ	37
3.16 ตัวอย่างการแสดงผลของส่วน Status History.....	37
3.17 ตัวอย่างการแสดงผลของส่วน Resource	38
3.18 ตัวอย่างการแสดงผลของส่วน Resource Utilization ค่าปกติ และค่าผิดปกติ	38
3.19 ตัวอย่างการแสดงผลของส่วน Resource Utilization History.....	38
3.20 ตัวอย่างการแสดงผลของส่วน Network	38
3.21 ตัวอย่างการแสดงผลกราฟข้อมูลใน Grafana	39
3.22 การตั้งค่าแจ้งเตือนไปยัง LINE.....	40
3.23 การตั้งค่าแจ้งเตือนไปยังส่วน Diagnostic and Recovery	40
3.24 การกำหนดเงื่อนไขการแจ้งเตือน.....	41

สารบัญรูป (ต่อ)

รูป	หน้า
3.25 โครงสร้างภายในของเครื่องแม่ข่ายเสมือนที่ให้บริการเว็บแอปพลิเคชัน	42
3.26 Root Cause Analysis ของเว็บแอปพลิเคชัน.....	43
3.27 สับเซตของปัญหา.....	44
3.28 ความสัมพันธ์ระหว่างส่วนประกอบ กับค่าสถานะภาพ.....	46
3.29 ขั้นตอนการทำงานของชุดคำสั่งตรวจสอบความผิดปกติ และกู้คืน	47
3.30 ชุดคำสั่งฟังก์ชัน Webhook.....	48
3.31 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Anomaly Classification	49
3.32 ชุดคำสั่งฟังก์ชัน Anomaly Classification.....	50
3.33 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker).....	51
3.34 ชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker) บรรทัดที่ 1-54.....	52
3.35 ชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker) บรรทัดที่ 55-111	53
3.36 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_service).....	55
3.37 ชุดคำสั่งฟังก์ชัน fix_service.....	56
3.38 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (check_firewall).....	57
3.39 ชุดคำสั่งฟังก์ชัน check_firewall	58
3.40 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (check_usage).....	59
3.41 ชุดคำสั่งฟังก์ชัน check_usage	59
3.42 การแจ้งเตือนปริมาณการใช้งานซีพียู มากกว่า 80 เปอร์เซ็นต์.....	62
3.43 การแจ้งเตือนปริมาณการใช้งานหน่วยความจำหลัก มากกว่า 80 เปอร์เซ็นต์.....	62
3.44 การแจ้งเตือนปริมาณการใช้งานหน่วยความจำสำรอง มากกว่า 80 เปอร์เซ็นต์.....	62
3.45 การแจ้งเตือนไม่มีการตอบสนองต่อการ Ping.....	63

สารบัญรูป (ต่อ)

รูป	หน้า
3.46 การแจ้งเตือน TCP พอร์ต 80 หรือ 443 ไม่ตอบสนอง	63
3.47 การแจ้งเตือนรหัสสถานภาพของ HTTP ไม่เท่ากับ 200	63
3.48 การติดต่อสื่อสารของส่วนประกอบระบบติดตามสถานภาพ.....	64
3.49 Prometheus ส่วน Targets.....	64
3.50 Prometheus ส่วน Scrape Duration.....	64
3.51 Grafana ส่วน Inspect	65
3.52 Prometheus ส่วน Total request time.....	65
3.53 การแจ้งเตือนเมื่อ Nginx ไม่ทำงาน.....	66
3.54 การแจ้งเตือนเมื่อ Nginx ไม่ทำงานเนื่องจาก Port ถูกใช้งานอยู่.....	67
3.55 การแจ้งเตือนเมื่อ Container Web Application ไม่ทำงาน	67
3.56 การแจ้งเตือนเมื่อ Container Web Application ไม่ทำงาน เนื่องจาก Port ถูกใช้งานอยู่	68
3.57 การแจ้งเตือนเมื่อ Container Web Database ไม่ทำงาน	68
3.58 การแจ้งเตือนเมื่อ Container Web Database ไม่ทำงาน เนื่องจาก Port ถูกใช้งานอยู่....	69
3.59 การแจ้งเตือนเมื่อ Docker ไม่ทำงาน.....	69
3.60 การแจ้งเตือนเมื่อปริมาณการใช้ซีพียูมากกว่า 80 เปอร์เซ็นต์.....	70
3.61 การแจ้งเตือนเมื่อปริมาณการใช้หน่วยความจำหลักมากกว่า 80 เปอร์เซ็นต์.....	70
4.1 การเปรียบเทียบค่าเปอร์เซ็นต์ซีพียู	72
4.2 การเปรียบเทียบค่าหน่วยความจำหลัก	73
4.3 การเปรียบเทียบค่าหน่วยความจำสำรอง	74
4.4 การเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่าย	75
4.5 การเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่าย	76
4.6 การเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของฐานข้อมูลอนุกรมเวลา.....	77
4.7 การเปรียบเทียบค่าปริมาณการใช้หน่วยความจำหลักของฐานข้อมูลอนุกรมเวลา.....	78
4.8 การเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของเครื่องมือติดตามสถานภาพ.....	79
4.9 การเปรียบเทียบค่าเปอร์เซ็นต์การใช้หน่วยความจำหลักของเครื่องมือติดตามสถานภาพ	80

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันเทคโนโลยีการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) ได้ถูกนำไปใช้อย่างแพร่หลาย เนื่องจากมีข้อดีหลายอย่างเช่น การประมวลผลร่วมกันของเครื่องแม่ข่ายจำนวนหลายเครื่องทำให้ได้การประมวลผลที่รวดเร็ว รองรับการะการทำงานได้ปริมาณมาก และเพิ่มลดทรัพยากรได้ตามความต้องการของผู้ใช้ ทำให้ภาคธุรกิจขนาดเล็ก กลาง และใหญ่เริ่มปรับเปลี่ยนโครงสร้างระบบเทคโนโลยีสารสนเทศ โดยการหันมาเช่าใช้บริการเครื่องแม่ข่ายเสมือน (Virtual Server) ที่อยู่บนโครงสร้างการประมวลผลแบบกลุ่มเมฆ ของผู้ให้บริการ มาใช้เพิ่มประสิทธิภาพให้ระบบเทคโนโลยีสารสนเทศ รองรับการขยายตัวของธุรกิจ และลดต้นทุน โดยในปี 2018 ตลาดการให้บริการคลาวด์เติบโตขึ้นประมาณ 31.3% เมื่อเทียบกับปีที่แล้ว [1]

จากการที่ตลาดการให้บริการคลาวด์เติบโตขึ้นเรื่อยๆ พร้อมกับจำนวนเครื่องแม่ข่ายเสมือนที่มีจำนวนมากขึ้น ผู้ให้บริการคลาวด์จึงมีภาระในการดูแลระบบเพิ่มขึ้นตามไปด้วย ผู้ให้บริการคลาวด์ต้องมีเจ้าหน้าที่คอยติดตามสถานภาพเครื่องแม่ข่ายเสมือนของผู้ใช้บริการอยู่ตลอดเวลา และต้องแก้ไขปัญหาให้ได้โดยเร็วที่สุดเมื่อพบปัญหา เพราะในการให้บริการคลาวด์ จะมีตัวชี้วัดมาตรฐานการให้บริการ (Service Level Agreement : SLA) คือข้อตกลงระหว่าง ผู้ให้บริการคลาวด์ กับ ผู้ใช้บริการ ในเรื่องสถานภาพของระบบที่ต้องสามารถทำงานได้ปกติตามเวลาที่ได้ตกลงไว้ เพื่อให้การดำเนินธุรกิจของผู้ใช้บริการมีความมั่นคง

เมื่อจำนวนเครื่องแม่ข่ายเสมือนของผู้ใช้บริการ บนระบบคลาวด์ของผู้ให้บริการเพิ่มขึ้น ยิ่งส่งผลให้การติดตามสถานภาพเครื่องแม่ข่ายเสมือน เกิดความล่าช้าในการรับรู้ความผิดปกติของเครื่องแม่ข่ายเสมือน และทำให้การรักษา SLA มีปัญหาซึ่งส่งผลเสียต่อทั้งผู้ให้บริการคลาวด์ และผู้ให้บริการ

นอกจากนี้ระบบงานที่อยู่ภายในเครื่องแม่ข่ายเสมือนเป็นสิ่งสำคัญเช่นกัน เนื่องจากเครื่องแม่ข่ายเสมือนมักถูกใช้ติดตั้งระบบงานที่ให้บริการผู้ใช้ ยกตัวอย่างเช่น ระบบเว็บแอปพลิเคชันที่ทำงานอยู่ภายในเครื่องแม่ข่ายเสมือน โดยในกรณีที่ระบบงานภายในมีปัญหา ผู้ดูแลระบบมีหน้าที่ตรวจสอบ และแก้ไขปัญหาส่วนต่างๆของระบบงานให้ทำงานปกติ ซึ่งใช้เวลาพอสมควรในการที่ผู้ดูแลระบบจะรับรู้ถึงปัญหา และยังต้องตรวจสอบส่วนประกอบของระบบงานที่มีหลายส่วนเพื่อหาสาเหตุของปัญหาแล้วจึงทำการกู้คืน โดยในบางกรณีการกู้คืนระบบงานมีขั้นตอนที่ชัดเจน สามารถนำเครื่องมือมาประยุกต์ใช้เพื่อแก้ไขปัญหาได้โดยอัตโนมัติ เพื่อให้ระบบงานสามารถกลับมาทำงานเป็นปกติอย่างรวดเร็ว

ตารางที่ 1.1 รายการกลุ่มทรัพย์สิน และการประเมินความเสี่ยง

ลำดับ	รายการกลุ่มทรัพย์สิน	ผลกระทบต่อการทำงาน	ระดับโอกาสเกิดความเสี่ยง	ความรุนแรง
1	Operating System	การให้บริการหยุดชะงัก เกิน 1 ชั่วโมง	ค่อนข้างบ่อย ปีละ 6-10 ครั้ง	สูง
2	Antivirus	การให้บริการหยุดชะงัก เกิน 1 ชั่วโมง	ปานกลาง ปีละ 3-5 ครั้ง	ปานกลาง
3	Application	การให้บริการหยุดชะงัก เกิน 1 ชั่วโมง	ค่อนข้างบ่อย ปีละ 6-10 ครั้ง	สูง
4	Virtual Machine	การให้บริการหยุดชะงัก เกิน 1 ชั่วโมง	ปานกลาง ปีละ 3-5 ครั้ง	ปานกลาง
5	Configuration	การให้บริการหยุดชะงัก เกิน 1 ชั่วโมง	ปานกลาง ปีละ 3-5 ครั้ง	ปานกลาง

จากตารางที่ 1.1 รายการกลุ่มทรัพย์สิน และการประเมินความเสี่ยง ของระบบงานที่ใช้ในการศึกษาครั้งนี้ โดยจากการประเมินความเสี่ยงองค์ประกอบของระบบพบว่า ระบบปฏิบัติการ (Operating System) และแอปพลิเคชัน (Application) เป็นองค์ประกอบที่มีความเสี่ยงสูง ถ้าเกิดปัญหาจะส่งผลกระทบต่อที่รุนแรง จึงควรให้ความสำคัญในการติดตามอย่างใกล้ชิด และหาวิธีลดผลกระทบให้น้อยลง

จากความเป็นมาของปัญหาดังที่กล่าวมาข้างต้น ผู้ศึกษาจึงสนใจออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน และแจ้งเตือนไปยังผู้ที่เกี่ยวข้อง เมื่อเครื่องแม่ข่ายเสมือนหรือระบบเว็บแอปพลิเคชันทำงานผิดปกติ เพื่อให้ผู้เกี่ยวข้องสามารถตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันให้กลับมาทำงานเป็นปกติได้ทันการณ์

1.2 วัตถุประสงค์การศึกษา

1. ออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน
2. เพื่อให้มีระบบแจ้งเตือนผู้เกี่ยวข้องให้ทราบ เมื่อเครื่องแม่ข่ายเสมือนทำงานผิดปกติ
3. เพื่อให้มีระบบตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันภายในเครื่องแม่ข่ายเสมือน

1.3 ขอบเขตของการศึกษา

ลักษณะของระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน เป็นเว็บแอปพลิเคชัน โดยการนำซอฟต์แวร์โอเพนซอร์สมาประยุกต์ใช้ โดยมีขอบเขตของการศึกษาดังนี้

1. ระบบสามารถติดตามสถานภาพเครื่องแม่ข่ายเสมือน ที่มีรูปแบบการให้บริการแบบ Infrastructure as a Service (IaaS)
2. ระบบสามารถติดตามสถานภาพเครื่องแม่ข่ายเสมือน และเว็บแอปพลิเคชันได้ที่อยู่ภายในเครื่อง โดยใช้โพรโทคอล Simple Network Management Protocol ,Internet Control Message Protocol, Hypertext Transfer Protocol และ Transmission Control Protocol
3. ระบบสามารถติดตามสถานภาพเครื่องแม่ข่ายเสมือนในระดับระบบปฏิบัติการ ได้แก่ ปริมาณการใช้งานซีพียู หน่วยความจำหลัก หน่วยความจำสำรอง และปริมาณการรับ-ส่งข้อมูลจราจรเครือข่าย
4. ระบบสามารถติดตามสถานภาพเครื่องแม่ข่ายเสมือนในระดับแอปพลิเคชัน ได้แก่ พอร์ต (Port) ของแอปพลิเคชัน และสถานภาพ HTTP ของเว็บแอปพลิเคชันที่ทำงานอยู่ในเครื่องแม่ข่ายเสมือน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ผู้เกี่ยวข้องทราบถึงปัญหาเมื่อเครื่องแม่ข่ายเสมือนทำงานผิดปกติ และแก้ไขได้ทันการณ์
2. ผู้เกี่ยวข้องทราบถึงแนวโน้มการใช้ทรัพยากรของเครื่องแม่ข่ายเสมือน เพื่อนำไปวางแผนจัดสรรทรัพยากรได้
3. ระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน ช่วยให้ผู้ใช้ดูแลระบบสามารถทำการแก้ปัญหาในเชิงรุกได้

1.5 นิยามคำศัพท์

1. **Infrastructure as a Service (IaaS)** คือ การให้บริการคลาวด์เฉพาะโครงสร้างพื้นฐาน ที่ผู้ใช้บริการสามารถนำไปใช้งานได้เหมือนคอมพิวเตอร์เครื่องหนึ่ง
2. **Simple Network Management Protocol (SNMP)** คือ โพรโทคอลสำหรับแลกเปลี่ยนข้อมูลสถานภาพของอุปกรณ์ที่เชื่อมต่อกับเครือข่าย เช่น ปริมาณการใช้งานซีพียู หน่วยความจำหลัก และหน่วยความจำสำรองของอุปกรณ์ เป็นต้น โดยสามารถนำมาใช้ติดตามสถานภาพเครื่องแม่ข่ายเสมือนได้
3. **Internet Control Message Protocol (ICMP)** คือ โพรโทคอลสำหรับตรวจสอบการเชื่อมต่อของอุปกรณ์ที่เชื่อมต่อกับเครือข่าย โดยการ Ping จากเครื่องต้นทางไปยังอุปกรณ์

ปลายทาง กรณีที่สามารถเชื่อมต่อกันได้ อุปกรณ์ปลายทางจะตอบกลับมายังเครื่องต้นทาง โดยสามารถนำมาใช้ติดตามสถานภาพการเชื่อมต่อเครือข่ายของเครื่องแม่ข่ายเสมือนได้

4. Hypertext Transfer Protocol (HTTP) คือ โพรโทคอลสำหรับติดต่อสื่อสารระหว่าง Client กับ Web Server โดย Client จะร้องขอข้อมูลโดยการส่งข้อมูล HTTP Request แล้ว Web Server จะส่งข้อมูล HTTP Response กลับมายัง Client โดย HTTP Response จะประกอบไปด้วยรหัสซึ่งเป็นค่าบ่งชี้สถานภาพการทำงานของเว็บแอปพลิเคชัน โดยสามารถนำมาใช้ติดตามสถานภาพเว็บแอปพลิเคชันได้ และรหัสที่ได้จาก HTTP Response สามารถบ่งชี้สาเหตุความผิดปกติของเว็บแอปพลิเคชันได้ ว่าเกิดจากส่วนประกอบส่วนใด ช่วยให้รู้สาเหตุ และแก้ไขปัญหาได้ตรงจุดมากขึ้น

5. Transmission Control Protocol (TCP) คือ โพรโทคอลสำหรับควบคุมการรับ-ส่งข้อมูลผ่านเครือข่าย โดยก่อนการรับ-ส่งข้อมูล TCP จะมีกระบวนการ 3-Way Handshake เพื่อตรวจสอบ และสร้างช่องทางรับ-ส่งข้อมูล โดยสามารถนำมาตรวจสอบการทำงานของพอร์ตเว็บแอปพลิเคชันที่ทำงานอยู่ในเครื่องแม่ข่ายเสมือน

TNI

บทที่ 2

ทฤษฎี เทคโนโลยี และงานวิจัยที่เกี่ยวข้อง

ในการศึกษาการออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน ผู้ศึกษาได้สืบค้นข้อมูล ทฤษฎี และงานวิจัย เพื่อเป็นแนวทางในการออกแบบ โดยมีหัวข้อดังต่อไปนี้

- 2.1 เทคโนโลยีการประมวลผลแบบกลุ่มเมฆ
- 2.2 โพรโทคอล Simple Network Management Protocol
- 2.3 ภาษาโปรแกรมไพทอน (Python Programming Language)
- 2.4 ฐานข้อมูลอนุกรมเวลา Prometheus
- 2.5 เครื่องมือติดตามสถานภาพ Grafana
- 2.6 งานวิจัยที่เกี่ยวข้อง

2.1 เทคโนโลยีการประมวลผลแบบกลุ่มเมฆ [2]

การประมวลผลแบบกลุ่มเมฆ เป็นการนำทรัพยากรด้านระบบเทคโนโลยีสารสนเทศของผู้ให้บริการ เช่น ฮาร์ดแวร์ ซอฟต์แวร์ และระบบเครือข่าย มาแบ่งให้ผู้ใช้งานหลายๆรายสามารถเข้าถึงใช้งานผ่านระบบเครือข่าย เพื่อให้การใช้ทรัพยากรทางด้านคอมพิวเตอร์มีประสิทธิภาพสูงสุด ช่วยให้ผู้ใช้บริการสามารถลดค่าใช้จ่ายในการจัดหาระบบเทคโนโลยีสารสนเทศได้

ลักษณะเฉพาะของการประมวลผลแบบกลุ่มเมฆมีดังนี้

1. On-Demand Self-Service ผู้ใช้บริการ สามารถบริหารจัดการทรัพยากรได้ด้วยตัวเอง ผ่านช่องทางการบริหารจัดการของผู้ให้บริการ
2. Broad network access สามารถเข้าถึงบริการ ผ่านเครือข่ายอินเทอร์เน็ต และอุปกรณ์พื้นฐานเช่น โทรศัพท์มือถือ คอมพิวเตอร์ และอื่นๆ
3. Resource pooling มีแหล่งทรัพยากรปริมาณมาก เพื่อรองรับความต้องการของผู้ใช้บริการหลายๆราย
4. Rapid elasticity สามารถเพิ่ม-ลดปริมาณทรัพยากรได้ตามความต้องการของผู้ใช้
5. Measured service สามารถวัดปริมาณการใช้ทรัพยากรได้ เพื่อนำไปคิดค่าใช้จ่าย

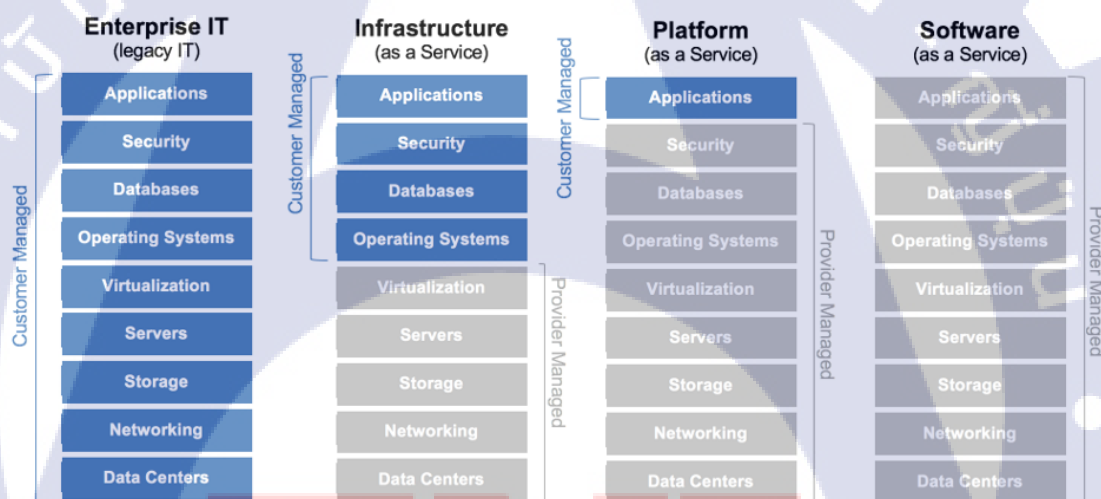
รูปแบบการให้บริการการประมวลผลแบบกลุ่มเมฆมี 3 รูปแบบดังนี้

1. Infrastructure as a Service รูปแบบบริการที่ผู้ให้บริการ ให้บริการเฉพาะโครงสร้างพื้นฐาน ได้แก่ พื้นที่จัดเก็บข้อมูล การประมวลผล และเครือข่าย ยกตัวอย่างเช่น ผู้ให้บริการสร้าง

Virtual Machine (VM) จากระบบประมวลผลแบบกลุ่มเมฆของตนเอง ให้ผู้ใช้บริการเช่าเพื่อนำไปใช้เป็นเครื่องแม่ข่ายเสมือน เป็นต้น

2. Platform as a Service รูปแบบที่ผู้ให้บริการให้บริการเครื่องมือที่ใช้ในการพัฒนาซอฟต์แวร์ เครื่องมือที่ใช้ในการเก็บข้อมูล โดยที่ผู้รับบริการสามารถพัฒนาระบบ ภายใต้ทรัพยากรของผู้ให้บริการ ซึ่งการให้บริการลักษณะนี้ เหมาะสำหรับองค์กรที่ต้องการพัฒนาระบบเองเพื่อให้ตรงกับความต้องการขององค์กรผู้ใช้บริการ ได้แก่ Google Cloud, Amazon Web Application และ Microsoft Azure เป็นต้น

3. Software as a Service รูปแบบการให้บริการโปรแกรมประยุกต์บนโครงสร้างของการประมวลผลแบบกลุ่มเมฆ ซึ่งผู้ให้บริการจะดูแลส่วนของซอฟต์แวร์ ฮาร์ดแวร์ ข้อมูลของผู้ใช้ การเชื่อมต่อเครือข่ายโปรแกรมของผู้ให้บริการ ยกตัวอย่างบริการรูปแบบนี้เช่น Google Apps และ Microsoft Office 365 เป็นต้น



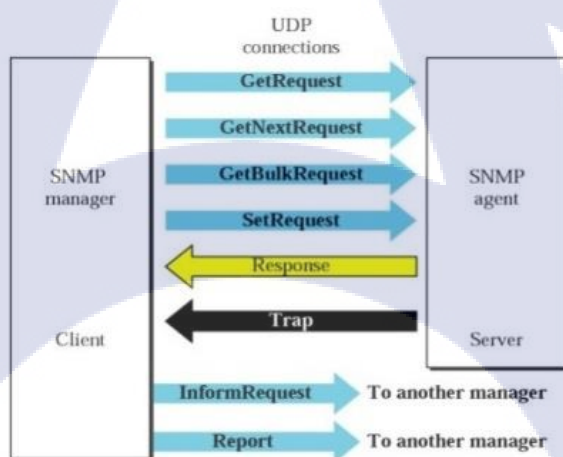
รูปที่ 2.1 รูปแบบการให้บริการการประมวลผลแบบกลุ่มเมฆ [3]

2.2 โพรโทคอล Simple Network Management Protocol [4]

Simple Network Management Protocol หรือ SNMP เป็นโพรโทคอลที่ใช้สำหรับการบริหารจัดการระบบเครือข่าย ทำงานอยู่ในแอปพลิเคชันเลเยอร์ ใช้สำหรับบริหารจัดการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์เครือข่าย ถูกใช้อย่างแพร่หลายในการจัดการ และเฝ้าติดตามสถานะภาพของระบบเครือข่าย

องค์ประกอบของโปรโตคอล SNMP ประกอบด้วย 4 ส่วนดังนี้

1. SNMP Manager ทำงานอยู่ในฝั่งระบบบริหารจัดการเครือข่าย ทำหน้าที่ส่งการร้องขอข้อมูลไปยัง Agent แล้วนำข้อมูลที่ได้รับไปประมวลผลต่อไป
2. อุปกรณ์ที่ต้องการบริหารจัดการ คืออุปกรณ์ที่เชื่อมต่อกับเครือข่าย ที่ต้องการติดตามสถานภาพการทำงาน เช่น คอมพิวเตอร์ สวิตช์ และเราเตอร์ เป็นต้น
3. SNMP Agent ทำงานอยู่ในฝั่งอุปกรณ์ที่ต้องการบริหารจัดการ ทำหน้าที่ค้นหาข้อมูลที่ Manager ร้องขอ จากฐานข้อมูลของอุปกรณ์แล้วส่งข้อมูลกลับไป
4. Management Information Base (MIB) คือฐานข้อมูลที่เก็บค่าข้อมูลต่างๆ ของอุปกรณ์ โดยแต่ละค่าข้อมูลจะมีชุดตัวเลขกำกับไว้ใช้อ้างอิงถึงค่าข้อมูลต่างๆ ของอุปกรณ์ เรียกว่า Object Identifier (OID) เช่น ค่าเปอร์เซ็นต์การใช้งานซีพียูของระบบปฏิบัติการวินโดวส์ มี OID คือ .1.3.6.1.2.1.25.3.3.1.2 เป็นต้น



รูปที่ 2.2 องค์ประกอบของโปรโตคอล SNMP

ข้อความที่ใช้ในการทำงานของโปรโตคอล SNMP

1. GET คือข้อความที่ SNMP Manager ใช้ส่งคำร้องขอข้อมูลจากอุปกรณ์
2. GETNEXT ต่างจากข้อความ GET ตรงที่จะส่งคำร้องขอข้อมูล OID ตัวถัดไป
3. GETBULK คือข้อความเพื่อร้องขอข้อมูลจำนวนมาก แทนการใช้ GET และ GET NEXT หลายครั้ง
4. SET คือข้อความใช้เปลี่ยนแปลงการตั้งค่าข้อมูลใน MIB
5. TRAP คือข้อความที่ Agent จะคอยแจ้งเตือนความผิดปกติของอุปกรณ์ ไปยัง SNMP Manager

6. INFORM คล้ายกับ TRAP แตต่างกันตรงที่เมื่อ Manager ได้รับข้อความจาก Agent แล้วจะตอบยืนยันกลับไปว่าได้รับข้อความแล้ว

7. RESPONSE คือข้อความที่ใช้ส่งข้อมูลกลับไปให้ SNMP Manager

หลักการทำงานของโพรโทคอล SNMP แบ่งเป็น 3 รูปแบบดังนี้

1. Read ทำงานผ่าน UDP พอร์ต 161 ในการติดต่อสื่อสารกันระหว่าง SNMP Manager กับ SNMP Agent โดย Manager จะส่งการร้องขอไปยัง Agent ที่อยู่ในแต่ละอุปกรณ์ แล้ว Agent จะทำการค้นหาข้อมูลใน MIB แล้วส่งค่าข้อมูลกลับไปให้ Manager ประมวลผลต่อไป ซึ่งข้อความที่ใช้ในการทำงานได้แก่ GET GETNEXT และ GETBULK

2. Write คือการที่ SNMP Manager จะส่งคำสั่งไปยัง Agent เพื่อตั้งค่าข้อมูลใน MIB ซึ่งข้อความที่ใช้ในการทำงานได้แก่ SET

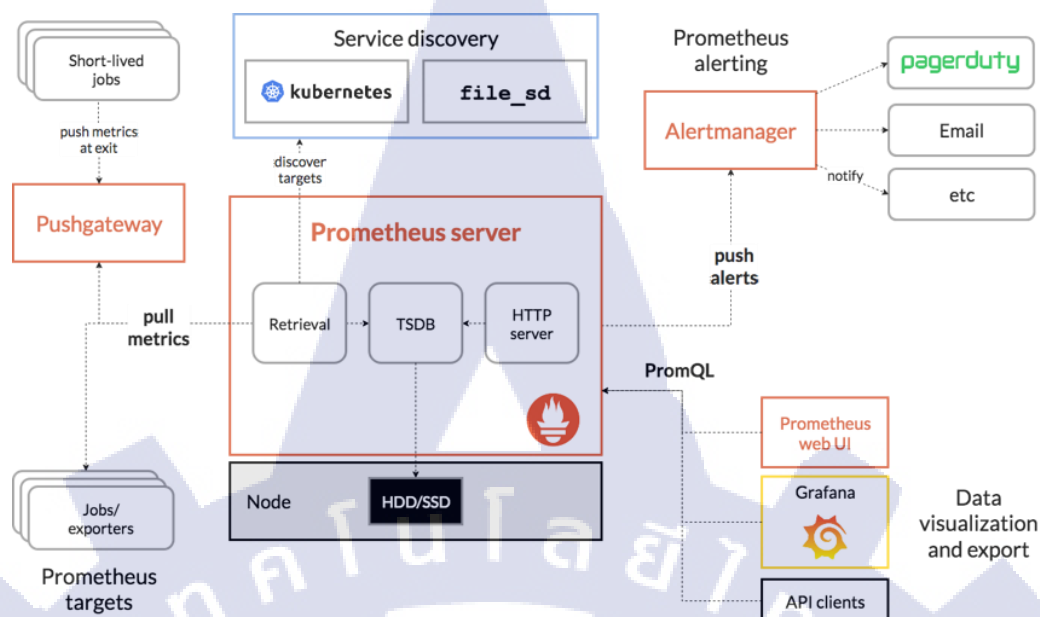
3. TRAP ทำงานผ่าน UDP พอร์ต 162 โดย SNMP Agent จะคอยส่งการแจ้งเตือนไปยัง Manager เมื่อพบความผิดปกติบนอุปกรณ์ตามที่ตั้งค่าไว้ ซึ่งข้อความที่ใช้ในการทำงานได้แก่ TRAP และ INFORM

2.3 ภาษาโปรแกรมไพทอน (Python Programing Language) [5]

ไพทอน (Python) เป็นภาษาการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming : OOP) ระดับสูง และเป็นโอเพนซอร์ส ถูกสร้างขึ้นโดย Guido van Rossum ในช่วงปี ค.ศ.1980-1990 ได้รับความนิยมเนื่องจากมีหลักการเขียนที่ชัดเจน ยืดหยุ่น เรียบง่าย และใช้คำศัพท์ภาษาอังกฤษที่พบเจอได้ในชีวิตประจำวัน จึงทำให้การเรียนรู้ภาษาไพทอนค่อนข้างง่าย และเข้าใจได้อย่างรวดเร็ว สามารถทำงานได้ในหลายอุปกรณ์ และหลายวัตถุประสงค์เช่น ใช้พัฒนาเว็บ แอปพลิเคชัน ฐานข้อมูล และวิเคราะห์ข้อมูลด้วยหลักการทางวิทยาศาสตร์ข้อมูล เป็นต้น

2.4 ฐานข้อมูลอนุกรมเวลา Prometheus [6]

Prometheus เป็นเครื่องมือโอเพนซอร์สใช้สำหรับ ฝ้าติดตามสถานะภาพระบบ และแจ้งเตือน ถูกพัฒนาขึ้นที่ SoundCloud ในปี ค.ศ. 2012 ถูกนำไปใช้ในหลายองค์กร อีกทั้งยังมีผู้พัฒนาอยู่ตลอด มีแหล่งชุมชนเพื่อแลกเปลี่ยนความรู้บนอินเทอร์เน็ต โดยในงานวิจัยนี้ Prometheus จะถูกนำมาประยุกต์ใช้เป็นฐานข้อมูลอนุกรมเวลา (Time-series Database) สำหรับเก็บค่าข้อมูลสถานะภาพของเครื่องแม่ข่ายเสมือน



รูปที่ 2.3 โครงสร้างการทำงานของ Prometheus

จากรูปที่ 2.3 โครงสร้างของ Prometheus ประกอบด้วย 6 ส่วน ได้แก่

1. Prometheus server คือจุดศูนย์กลางของ Prometheus ทำหน้าที่รวบรวมข้อมูล และเก็บข้อมูลอนุกรมเวลา
2. Prometheus targets คือเป้าหมายแหล่งข้อมูลที่ Prometheus server ต้องไปรวบรวมข้อมูล
3. Pushgateway คือ Gateway ที่เก็บรวบรวมข้อมูลที่ได้รับจากการ Push ข้อมูลมาจากแหล่งข้อมูล เพื่อให้ Prometheus server นำข้อมูลไปเก็บเข้าฐานข้อมูล
4. Service discovery คือส่วนที่มีหน้าที่ติดตามสถานภาพการทำงานของซอฟต์แวร์ที่รองรับการทำงานร่วมกับ Prometheus เช่น ฐานข้อมูล
5. Alertmanager คือส่วนการแจ้งเตือนความผิดปกติที่ตั้งค่าไว้
6. Data visualization and export คือ ส่วนส่งออกข้อมูล และแสดงกราฟข้อมูล

Prometheus มีการเก็บข้อมูลทั้งหมดเป็นข้อมูลอนุกรมเวลา โดยการประทับเวลา (Time stamp) ให้กับทุกข้อมูลที่เก็บรวบรวม โดยมีการใช้ชื่อ Metric และ Label ในการระบุข้อมูลไม่ให้ซ้ำกัน ชื่อ Metric คือชื่อที่บ่งบอกว่าค่าข้อมูลนั้น เป็นค่าของอะไร เช่น `cpu_usage_percent` คือ ค่าเปอร์เซ็นต์การใช้งาน CPU เป็นต้น ส่วน Labels เป็นค่าช่วยเพิ่มมิติข้อมูลให้กับ Metric เช่น `cpu_usage_percent {type="notebook", name="A"}` จากในตัวอย่าง ชื่อของ Metric คือ

cpu_usage_percent ประกอบด้วย Labels 2 ค่าคือ type ระบุประเภทอุปกรณ์ และ name ระบุชื่อของอุปกรณ์ เป็นต้น

ตารางที่ 2.1 การเปรียบเทียบคุณสมบัติ และประสิทธิภาพเครื่องมือฐานข้อมูลอนุกรมเวลา

ข้อเปรียบเทียบ	Prometheus	InfluxDB	Graphite
1. Opensource	ใช่	ใช่	ใช่
2. Data Collect Model	Pull และ Push	Push	Push
3. Data Type Supports	ตัวเลข	ตัวเลข, บูลีน, สตริง	ตัวเลข
4. Query Language	PromQL	SQL	GraphiteQL
5. Data Analysis Function	หลากหลาย	น้อย	น้อย
6. Timestamp resolution	มิลลิวินาที	นาโนวินาที	มิลลิวินาที
7. CPU Consumption*	ต่ำ	สูงมาก	สูง
8. Memory Consumption*	ต่ำ	สูง	สูงมาก
9. Number of Metrics Process on Average 50% of CPU Utilization*	20,000	2,000	2,400

* ทดสอบบนเครื่องแม่ข่ายเสมือนจำนวนซีพียู 16 คอร์ และหน่วยความจำหลัก 16 กิกะไบต์

จากตารางที่ 2.1 เครื่องมือฐานข้อมูลอนุกรมที่นำมาเปรียบเทียบเป็น Opensource ทั้งหมด รองรับการรวบรวมข้อมูลแบบ Push ยกเว้น Prometheus ที่รองรับทั้ง Push และ Pull ในด้านชนิดข้อมูลที่รองรับ InfluxDB รองรับข้อมูลได้หลายประเภท แต่ในทางปฏิบัติข้อมูลตัวเลขก็เพียงพอต่อการใช้งานด้าน Monitoring ในส่วนภาษาที่ใช้เรียกข้อมูล Prometheus และ Graphite มีรูปแบบที่เข้าใจง่ายไม่ซับซ้อน ในส่วนความละเอียดของการประทับเวลา InfluxDB มีความละเอียดมากที่สุดคือ นาโนวินาทีซึ่งเหมาะกับการนำไปใช้กับข้อมูล Transaction ที่ต้องการความละเอียดสูง ในส่วนปริมาณการใช้ซีพียู และหน่วยความจำหลัก Prometheus ใช้ทรัพยากรน้อยที่สุด

ตารางที่ 2.2 การเปรียบเทียบคะแนนเครื่องมือฐานข้อมูลอนุกรมเวลา

ข้อเปรียบเทียบ	คะแนนเต็ม	คะแนนที่ได้		
		Prometheus	InfluxDB	Graphite
1. Opensource	10.00	10.00	10.00	10.00
2. Data Collect Model	10.00	9.00	5.00	5.00
3. Data Type Supports	5.00	2.00	5.00	2.00
4. Query Language	10.00	10.00	5.00	10.00
5. Data Analysis Function	10.00	10.00	7.00	5.00
6. Timestamp resolution	5.00	2.00	5.00	2.00
7. CPU Consumption	15.00	15.00	5.00	10.00
8. Memory Consumption	15.00	15.00	10.00	5.00
9. Number of Metrics Process on Average 50% of CPU Utilization	20.00	20.00	10.00	12.00
รวม	100.00	93.00	62.00	61.00

จากตารางที่ 2.2 ผู้ศึกษาได้ทำการให้คะแนนแต่ละเครื่องมือตามข้อเปรียบเทียบ โดยเครื่องมือที่มีคุณสมบัติ และประสิทธิภาพมากจะได้คะแนนที่มาก และเครื่องมือที่มีคุณสมบัติ และประสิทธิภาพน้อยจะได้คะแนนที่น้อย ซึ่งจากการให้คะแนน เครื่องมือ Prometheus มีคะแนนมากที่สุด รองลงมาคือ InfluxDB และ Graphite ตามลำดับ

ผู้ศึกษาได้ตัดสินใจเลือกใช้ Promethues เพราะมีคุณสมบัติ และประสิทธิภาพมากกว่าเครื่องมือตัวอื่น ทั้งในด้านฟังก์ชันการทำงาน และด้านปริมาณการใช้ทรัพยากรทั้งซีพียู และหน่วยความจำหลักที่ใช้ในปริมาณน้อย แต่สามารถรองรับปริมาณข้อมูลได้จำนวนมาก ทำให้สามารถรองรับการขยายตัวของระบบติดตามสถานะภาพเครื่องแม่ข่ายเสมือนในอนาคตได้มาก และใช้ทรัพยากรได้คุ้มค่ามากที่สุด

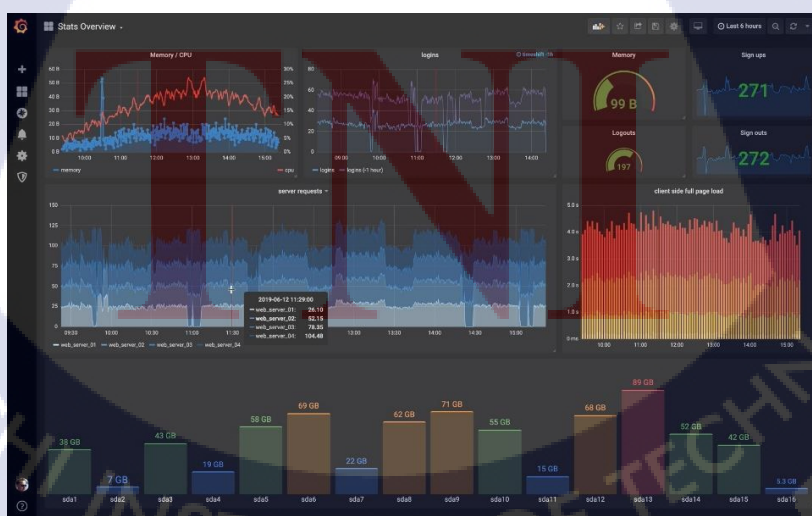
2.5 เครื่องมือติดตามสถานภาพ Grafana [7] [8]

Grafana คือซอฟต์แวร์โอเพนซอร์สสำหรับการวิเคราะห์ และสร้างกราฟข้อมูลที่สามารถโต้ตอบกับผู้ใช้ได้ ถูกพัฒนาขึ้นในปี ค.ศ. 2014 โดย Torkel Ödegaard ชาวสวีเดน เป็นเว็บแอปพลิเคชันที่รองรับการนำเข้าข้อมูลจากหลายแหล่งข้อมูลมาสร้างกราฟข้อมูล สร้างแดชบอร์ดสำหรับการเฝ้าติดตาม และการแจ้งเตือนผ่านทางอีเมล LINE และอื่นๆ

Grafana ถูกนิยมนำมาใช้งานร่วมกับ ฐานข้อมูลอนุกรมเวลาเช่น Prometheus และ InfluxDB หรือซอฟต์แวร์เฝ้าติดตามอื่นๆ เช่น Zabbix และ PRTG และอีกมากมาย

คุณสมบัติของ Grafana ได้แก่

1. การสร้างกราฟข้อมูล สามารถสร้างกราฟข้อมูลได้หลายรูปแบบเพื่อช่วยให้ผู้ใช้เข้าใจ และสร้างความสวยงามให้การแสดงข้อมูล
2. การแจ้งเตือน สามารถแจ้งเตือนไปยังผู้ใช้งาน เมื่อข้อมูลถึงขีดจำกัดที่ตั้งค่าไว้ ผ่านหลากหลายช่องทางเช่น อีเมล LINE และอื่นๆ
3. การนำเข้าข้อมูล สามารถนำเข้าข้อมูลจากหลายแหล่งข้อมูล เพื่อนำมาวิเคราะห์ร่วมกัน
4. การติดตั้ง สามารถนำไปติดตั้งใช้งานได้กับระบบปฏิบัติการ Windows, Linux และ Mac นอกจากนี้ยังรองรับการติดตั้งใน Docker Container ด้วย
5. การเพิ่มความสามารถ การอนุญาตให้นักพัฒนาทุกคนสามารถพัฒนา Plugin เพื่อมาทำงานร่วมกับ Grafana ทำให้มี Plugin มากมายให้เลือกใช้
5. การทำงานร่วมกัน สามารถเข้าใช้ได้พร้อมๆกัน ใส่คำอธิบาย (Annotation) ให้กราฟ ตั้งสิทธิ์การเข้าถึงแดชบอร์ด (Permission) แบ่งแดชบอร์ดตามกลุ่มผู้ใช้ได้



รูปที่ 2.4 ตัวอย่างการแสดงผลของ Grafana

ตารางที่ 2.3 การเปรียบเทียบคุณสมบัติ และประสิทธิภาพเครื่องมือติดตามสถานะภาพ

ข้อเปรียบเทียบ	Grafana	Chronograf	Zabbix
1. Alerting	รองรับ	รองรับ	รองรับ
2. Data source	หลากหลาย	InfluxDB เท่านั้น	MySQL หรือ PostgreSQL
3. Chart Type	Graph, Stat, Gauge, Bar Chart, Table, Text, Heatmap	Graph, Stat, Gauge, Bar Chart, Table, Text	Graph, Text
4. Annotation	รองรับ	รองรับ	ไม่รองรับ
5. Value Mapping	รองรับ	ไม่รองรับ	รองรับ
6. Interactive Visualization	Filtering Tooltip Relate	Filtering Tooltip Relate	Filtering Tooltip
7. CPU Consumption*	สูง	ต่ำ	สูงมาก
8. Memory Consumption*	ต่ำ	สูง	สูงมาก

* ทดสอบบนเครื่องแม่ข่ายเสมือนจำนวนซีพียู 16 คอร์ และหน่วยความจำหลัก 16 กิกะไบต์

จากตารางที่ 2.3 เครื่องมือติดตามสถานะภาพที่นำมาเปรียบเทียบ สามารถแจ้งเตือนเมื่อพบค่าสถานะภาพผิดปกติได้ แต่มีแค่ Grafana เท่านั้นที่รองรับการแจ้งเตือนผ่านทาง LINE Application ในส่วนการเชื่อมต่อกับฐานข้อมูล Grafana รองรับฐานข้อมูลหลายประเภท แต่ Chronograf กับ Zabbix ค่อนข้างจำกัด ในส่วนรูปแบบการแสดงผล และฟังก์ชันการแสดงผล Grafana กับ Chronograf สามารถแสดงผลได้หลากหลาย แต่ Grafana สามารถปรับแต่งได้มากกว่า ในขณะที่ Zabbix มีการแสดงผลที่จำกัด ในส่วนปริมาณการใช้งานซีพียู Chronograf ใช้ปริมาณซีพียูต่ำที่สุด ในส่วนปริมาณการใช้หน่วยความจำหลัก Grafana ใช้ปริมาณหน่วยความจำหลักต่ำที่สุด

ตารางที่ 2.4 การเปรียบเทียบคะแนนเครื่องมือติดตามสถานภาพ

ข้อเปรียบเทียบ	คะแนนเต็ม	คะแนนที่ได้		
		Grafana	Chronograf	Zabbix
1. Alerting	15.00	15.00	13.00	10.00
2. Data source	20.00	20.00	10.00	10.00
3. Chart Type	20.00	20.00	17.00	10.00
4. Annotation	5.00	5.00	5.00	0.00
5. Value Mapping	10.00	10.00	0.00	10.00
6. Interactive Visualization	10.00	10.00	10.00	4.00
7. CPU Consumption	10.00	7.00	10.00	4.00
8. Memory Consumption	10.00	10.00	7.00	4.00
รวม	100.00	97.00	72.00	52.00

จากตารางที่ 2.4 ผู้ศึกษาได้ทำการให้คะแนนแต่ละเครื่องมือตามข้อเปรียบเทียบ โดยเครื่องมือที่มีคุณสมบัติ และประสิทธิภาพมากจะได้คะแนนที่มาก และเครื่องมือที่มีคุณสมบัติ และประสิทธิภาพน้อยจะได้คะแนนที่น้อย ซึ่งจากการให้คะแนน เครื่องมือ Grafana มีคะแนนมากที่สุด รองลงมาคือ Chronograf และ Zabbix ตามลำดับ

ผู้ศึกษาตัดสินใจเลือกใช้ Grafana เพราะมีคุณสมบัติ และประสิทธิภาพมากกว่าเครื่องมือตัวอื่น ในด้านฟังก์ชันการทำงาน Grafana รองรับการทำงานร่วมกับฐานข้อมูลได้หลากหลายประเภท ทำให้มีความยืดหยุ่นในการแสดงข้อมูลหลายประเภท ด้านการแสดงผลข้อมูลของ Grafana มีกราฟข้อมูลหลายรูปแบบ ผู้ใช้สามารถปฏิสัมพันธ์กับกราฟข้อมูลได้ สามารถตั้งค่าการแสดงผลได้หลากหลายกว่าเครื่องมือตัวอื่น และด้านการแจ้งเตือนสามารถทำได้หลายช่องทาง โดยเฉพาะแอปพลิเคชัน LINE ที่นิยมใช้ในประเทศไทย ซึ่งเครื่องมือตัวอื่นไม่สามารถทำได้

2.6 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน มีดังนี้

K. Anshul [9] ได้ศึกษาแนวทางการใช้ซอฟต์แวร์เฝ้าสังเกตแบบโอเพนซอร์ส เช่น Nagios, Zabbix และ Hyperic Hq มาประยุกต์ใช้เพื่อการติดตามสถานภาพเครื่องแม่ข่ายแบบอัตโนมัติ ลดการสิ้นเปลืองทรัพยากรบุคคล และเวลา จากการวิเคราะห์ความต้องการของระบบ ทักษะของผู้ใช้งาน และความสามารถของซอฟต์แวร์ทั้งในเรื่องความง่ายในการติดตั้ง การตั้งค่า ฟังก์ชัน และคู่มือของซอฟต์แวร์ ผู้ศึกษาได้เลือกใช้ Hyperic Hq มาประยุกต์ใช้กับระบบติดตามสถานภาพระบบเครื่องแม่ข่าย ผลลัพธ์คือ สามารถลดเวลาในการตรวจสอบเครื่องแม่ข่าย ได้ข้อมูลที่แม่นยำและยังช่วยให้ผู้ดูแลระบบสามารถแก้ไขปัญหาเชิงรุกได้

จากการนำเสนอระบบของ Anshul Kaushik โดยการนำซอฟต์แวร์โอเพนซอร์สมาประยุกต์ใช้ซึ่งต้องมีค่าถึงหลายปัจจัย โดยปัจจัยที่สำคัญคือ ฟังก์ชัน และคู่มือของซอฟต์แวร์ที่จะนำมาใช้ ซึ่งการนำซอฟต์แวร์โอเพนซอร์สมาใช้สามารถช่วยให้ติดตั้งระบบได้รวดเร็ว

G. Nidhishree และ S. Manimala [10] ได้นำเสนอการออกแบบ ระบบติดตามสถานภาพอุปกรณ์เครือข่ายด้วยโพรโทคอล SNMP โดยแบ่งโครงสร้างระบบออกเป็น 3 ส่วนคือ 1). ส่วนรวบรวมข้อมูลจากอุปกรณ์เครือข่ายด้วยโพรโทคอล SNMP 2). ส่วนประมวลผลข้อมูล และ 3). ส่วนต่อประสานกับผู้ใช้ โดยแสดงผลข้อมูลผ่านการประมวลผลให้ผู้ใช้เห็นในรูปแบบกราฟข้อมูล ผลลัพธ์การออกแบบโครงสร้างระบบติดตามสถานภาพอุปกรณ์เครือข่าย ระบบสามารถรวบรวมข้อมูลสถานภาพของอุปกรณ์เครือข่ายด้วย SNMP ได้ อีกทั้งยังสามารถวิเคราะห์ และแสดงข้อมูลผ่านเว็บ และแจ้งเตือนความผิดปกติได้ตามเวลาจริง

จากการนำเสนอ Framework ระบบของ G. Nidhishree และ S. Manimala เป็นการติดตามสถานภาพแบบไม่ใช้โปรแกรมตัวแทนในการรวบรวมข้อมูลสถานภาพอุปกรณ์เครือข่าย แต่จะใช้โพรโทคอล SNMP แทน ซึ่งจะต้องพัฒนาร่วมกับภาษาโปรแกรมเพื่อทำหน้าที่รวบรวมข้อมูลสถานภาพ ทำให้สามารถเลือกเฉพาะข้อมูลที่ต้องการได้ ส่งผลกระทบต่อประสิทธิภาพของอุปกรณ์ และระบบเครือข่ายน้อยลง เมื่อได้ข้อมูลสถานภาพแล้ว จึงนำไปประมวลผลข้อมูล แล้วแสดงผลกราฟข้อมูล

M. Brattstrom และ P. Morreale [11] ได้นำเสนอระบบติดตามสถานภาพระบบการประมวลผลแบบคลาวด์ที่สามารถรองรับการขยายระบบได้ และไม่ใช้โปรแกรมตัวแทนในการติดตามสถานภาพของระบบ โดยใช้ซอฟต์แวร์โอเพนซอร์สเพื่อให้สามารถนำไปประยุกต์ใช้กับระบบคลาวด์ได้อย่างรวดเร็ว โดยระบบแบ่งได้ 3 ส่วนคือ ส่วนรวบรวม และเก็บข้อมูล, ส่วนแสดงผล และส่วนแจ้งเตือนปัญหา ในส่วนรวบรวม และเก็บข้อมูล ผู้ศึกษาเลือกใช้โพรโทคอล SNMP ในการรวบรวมข้อมูลจากอุปกรณ์เครือข่าย และใช้ Prometheus เป็นฐานข้อมูลอนุกรมเวลา ในส่วนแสดงผลใช้ Grafana

เป็นที่แสดงข้อมูลสถานะภาพของระบบในรูปแบบกราฟแดชบอร์ด และในส่วนแจ้งเตือนปัญหาใช้ Alert Manager ซึ่งเป็นส่วนหนึ่งของ Prometheus ในการแจ้งเตือนไปยังผู้ดูแลระบบเมื่อระบบคลาวด์มีสถานะภาพผิดปกติ โดยงานนำเสนอนี้ สามารถพิสูจน์การนำซอฟต์แวร์โอเพนซอร์สช่วยประหยัดต้นทุน และผู้ใช้ไม่จำเป็นต้องมีทักษะคอมพิวเตอร์ที่สูงในการนำไปประยุกต์ใช้

จากการนำเสนอระบบของ Morgan Brattstrom และ Patricia Morreale เป็นการนำระบบติดตามสถานะภาพอุปกรณ์ แบบไม่ใช่โปรแกรมตัวแทน แต่ใช้โปรโตคอล SNMP ในการติดตามข้อมูลสถานะภาพอุปกรณ์เครือข่าย ในส่วนฐานข้อมูล และส่วนแสดงผลเป็นการนำซอฟต์แวร์โอเพนซอร์สประเภท Monitoring มาประยุกต์ใช้ ทำให้การติดตั้งระบบมีความรวดเร็ว

เชษฐ ศรีแย้ม [12] ได้พัฒนาระบบติดตามสถานะภาพอุปกรณ์เครือข่าย บนระบบปฏิบัติการลินุกซ์ อุ-บุนตุ โดยใช้โปรโตคอล SNMP กับ ICMP ติดตามสถานะภาพอุปกรณ์เครือข่ายได้แก่ เครื่องแม่ข่าย, เราเตอร์ และสวิตช์ โดยมีฐานข้อมูล MySQL Server เก็บข้อมูลอุปกรณ์เครือข่าย และแสดงผลข้อมูลบนเว็บไซต์ที่พัฒนาด้วยภาษา PHP ผลลัพธ์ของการพัฒนาเป็นเว็บไซต์ที่ผู้ใช้สามารถดูข้อมูลในรูปแบบตารางข้อมูล และสามารถจัดการเพิ่ม หรือลบอุปกรณ์เครือข่ายในระบบติดตามสถานะภาพได้

จากการนำเสนอระบบของ เชษฐ ศรีแย้ม เป็นการนำระบบติดตามสถานะภาพอุปกรณ์ แบบไม่ใช่โปรแกรมตัวแทน โดยใช้โปรโตคอล SNMP ร่วมกับโปรโท ICMP สามารถติดตามสถานะภาพอุปกรณ์เครือข่ายที่หลากหลายได้ โดยโปรโตคอล ICMP จะช่วยเสริมการตรวจสอบการสถานะภาพการเชื่อมต่อของอุปกรณ์กับเครือข่าย ซึ่งต่างกับโปรโตคอล SNMP ที่ใช้เพื่อตรวจสอบสถานะภาพภายในอุปกรณ์

จักรินทร์ พิมพ์ดี [13] ได้พัฒนาระบบตรวจสอบอุปกรณ์เครือข่าย และเครื่องแม่ข่าย โดยระบบแบ่งออกเป็น 2 ส่วนคือ ส่วนตรวจสอบระบบ กับส่วนแจ้งเตือน โดยในส่วนตรวจสอบระบบใช้โปรโตคอล ICMP ตรวจสอบสถานะภาพอุปกรณ์เครือข่าย และใช้โปรโตคอล SNMP ที่กำหนดค่า OID ไปรวบรวมข้อมูล การใช้งานซีพียู,หน่วยความจำหลัก หรือหน่วยความจำสำรอง แล้วเก็บข้อมูลพื้นฐานข้อมูล LiteDB แล้วนำข้อมูลไปแสดงผลที่หน้าโปรแกรมที่พัฒนาด้วยภาษา .NET ผลลัพธ์ของการพัฒนาคือโปรแกรมที่สามารถดูข้อมูลการทำงานของอุปกรณ์เครือข่ายในรูปแบบกราฟข้อมูลย้อนหลังได้ และสามารถแจ้งเตือนเมื่อค่าการทำงานของอุปกรณ์ ผิดปกติตามที่ตั้งเงื่อนไข โดยระบบจะส่งข้อความแจ้งเตือนไปยังผู้ดูแลระบบ ผ่านทางแอปพลิเคชัน LINE ทันที

จากการนำเสนอระบบของ จักรินทร์ พิมพ์ดี เป็นการนำระบบติดตามสถานะภาพอุปกรณ์ แบบไม่ใช่โปรแกรมตัวแทน โดยใช้โปรโตคอล SNMP ที่กำหนดค่า OID เพื่อระบุเฉพาะค่าข้อมูลที่ต้องการตรวจสอบ ซึ่งช่วยลดผลกระทบต่อปริมาณข้อมูลภายในเครือข่าย และลดผลกระทบต่อการประมวลผลของอุปกรณ์ที่ถูกตรวจสอบสถานะภาพ ในส่วนของการแจ้งเตือนเมื่อสถานะภาพอุปกรณ์มี

ความผิดปกติ จะส่งการแจ้งเตือนผ่านแอปพลิเคชันไลน์ ซึ่งเพิ่มความสะดวกให้ผู้ดูแลระบบได้ทราบความผิดปกติได้รวดเร็ว

ตารางที่ 2.5 การเปรียบเทียบประสิทธิภาพระบบของงานวิจัยที่เกี่ยวข้อง

ผู้วิจัย	เครื่องมือ			อุปกรณ์ที่รองรับ		เวลาการติดตั้ง	การแสดงผลกราฟข้อมูล
	Data Collection	Database	Monitoring	เครื่องแม่ข่าย	อุปกรณ์เครือข่าย		
1. Anshul Kaushik	Agent Based	ไม่ระบุ	Hyperic HQ	รองรับ	ไม่รองรับ	เร็วที่สุด	ดี
2. Nidhishree & Manimala	SNMP	ไม่ระบุ	Java	รองรับ	รองรับ	ช้า	ปานกลาง
3. Morgan และ Patricia	SNMP	Prometheus	Grafana	รองรับ	รองรับ	เร็ว	ดีที่สุด
4. เชษฐ ศรีแย้ม	SNMP	MySQL	PHP	รองรับ	รองรับ	ช้า	ปานกลาง
5. จักรินทร์ พิมพ์ดี	SNMP	LiteDB	.Net	รองรับ	รองรับ	ช้า	ปานกลาง

จากตารางที่ 2.5 การเปรียบเทียบประสิทธิภาพระบบของงานวิจัยที่เกี่ยวข้อง สามารถสรุปได้ว่าการใช้โปรโตคอล SNMP ในการตรวจสอบสถานะภาพอุปกรณ์มีความยืดหยุ่นมากกว่าการใช้โปรแกรมตัวแทน เนื่องจากอุปกรณ์ที่สามารถเชื่อมต่อกับเครือข่ายได้นั้น ส่วนมากรองรับโปรโตคอล SNMP ในส่วนเวลาที่ใช้ในการติดตั้ง ระบบที่ใช้โปรแกรมตัวแทนสามารถติดตั้งระบบได้อย่างรวดเร็วเนื่องจากเป็นโปรแกรมสำเร็จรูป แต่ระบบที่ใช้โปรโตคอล SNMP ผู้ใช้ต้องมีการพัฒนาชุดคำสั่งขึ้นมาเพื่อทำงานร่วมกับโปรโตคอล SNMP นอกจากนี้ยังต้องประกอบไปด้วยส่วนฐานข้อมูล และส่วนติดตามสถานะภาพที่ผู้ใช้ต้องออกแบบติดตั้งเอง ทำให้การติดตั้งระบบค่อนข้างใช้เวลามากกว่าแบบโปรแกรมตัวแทน แต่ในกรณีการนำเสนอระบบของ Morgan Brattstrom และ Patricia Morreale ที่นำซอฟต์แวร์โอเพนซอร์สมา Prometheus และ Grafana มาประยุกต์ใช้ทำให้เวลาการติดตั้งน้อยลง เนื่องจาก Prometheus และ Grafana ถูกออกแบบมาเพื่อใช้งานด้านการ Monitoring โดยเฉพาะ จึงมีฟังก์ชันการทำงานที่พร้อมใช้งาน และมีประสิทธิภาพ

จากข้อดี และข้อสังเกตในแต่ละงานวิจัยที่เกี่ยวข้อง ผู้ศึกษาสนใจออกแบบระบบตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือนด้วยโปรโตคอล SNMP ที่มีความยืดหยุ่นรองรับการขยายขอบเขตของการติดตามสถานะภาพของอุปกรณ์ ร่วมกับการนำซอฟต์แวร์โอเพนซอร์สมาประยุกต์ใช้งาน เพื่อให้ระบบมีประสิทธิภาพทั้งในด้านการรองรับการขยายตัวของระบบ ด้านการแสดงผลกราฟข้อมูล และด้านการแจ้งเตือนความผิดปกติ ผู้ศึกษาได้เล็งเห็นประโยชน์จากระบบติดตามสถานะภาพ และสนใจต่อยอดพัฒนาส่วนกึ่งเว็บแอปพลิเคชัน ที่อยู่ภายในเครื่องแม่ข่ายเสมือนให้กลับมาทำงานได้ปกติ ในกรณีที่ระบบตรวจสอบสถานะภาพพบความผิดปกติ

บทที่ 3

วิธีดำเนินการศึกษา

จากการศึกษาแนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องเพื่อเป็นความรู้สำหรับการศึกษาในครั้งนี้ ผู้ศึกษาได้นำความรู้ที่ได้รับมาดำเนินการออกแบบระบบ โดยมีหัวข้ออธิบายรายละเอียดการดำเนินการศึกษาดังนี้

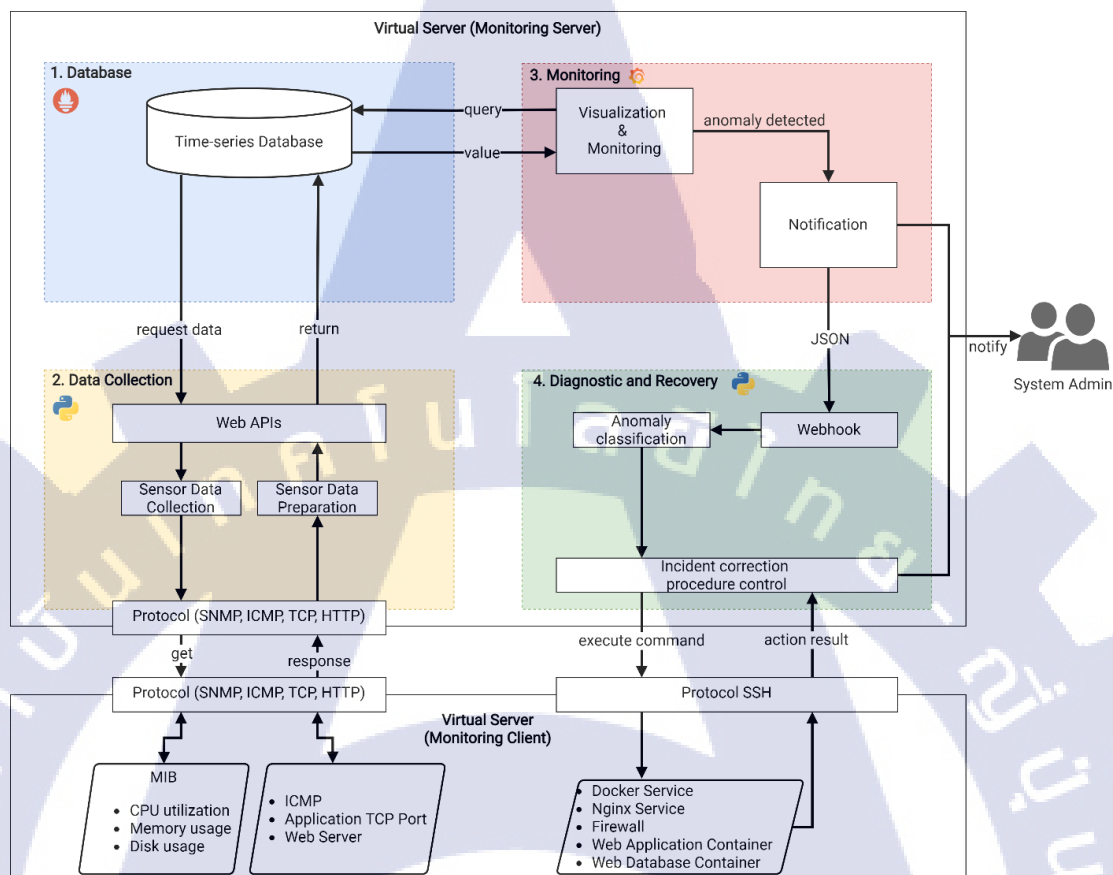
- 3.1 แผนการดำเนินงาน
- 3.2 ภาพรวมการทำงานของระบบ
- 3.3 รายละเอียดการดำเนินงาน
- 3.4 การออกแบบการทดสอบ

3.1 แผนการดำเนินงาน

ตารางที่ 3.1 แผนการดำเนินงาน

หัวข้องาน	ปี 2563						ปี 2564	
	ม.ค.	ก.พ.	มี.ค.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.
1. ศึกษาความต้องการของระบบ								
2. ศึกษาทฤษฎี และแนวคิด								
3. ศึกษาเครื่องมือ และทดลอง								
4. ร่าง และเสนอเค้าโครงสารนิพนธ์								
4. ออกแบบการทำงานของระบบ								
5. ติดตั้งระบบ และพัฒนาชุดคำสั่ง								
6. สร้างกราฟข้อมูล และการแจ้งเตือน								
7. ทดสอบ และปรับปรุง								
8. ประเมินผล								

3.2 ภาพรวมการทำงานของระบบ



รูปที่ 3.1 ภาพรวมการทำงานของระบบ

จากรูปที่ 3.1 ภาพรวมการทำงานของระบบ แบ่งออกเป็น 4 ส่วนดังนี้

1. Database คือฐานข้อมูลอนุกรมเวลา Prometheus มีหน้าที่ร้องขอข้อมูลจากส่วน Data Collection แล้วแปลงข้อมูลที่ได้รับ เป็นข้อมูลอนุกรมเวลาโดยการประทับเวลาข้อมูล และบันทึกไว้ในฐานข้อมูล

2. **Data Collection** คือชุดคำสั่งไพทอน มีหน้าที่รวบรวมข้อมูลสถานะภาพของเครื่องแม่ข่ายเสมือน โดยการติดต่อสื่อสารกับเครื่องแม่ข่ายเสมือน ผ่านโปรโตคอล SNMP, ICPM, TCP และ HTTP โดยทั้ง 4 โปรโตคอล สามารถใช้ติดตามสถานะภาพทั้งเครื่องแม่ข่ายเสมือน และเว็บแอปพลิเคชันได้อย่างครอบคลุม ตามวัตถุประสงค์ของการศึกษานี้

3. Monitoring คือเครื่องมือติดตามสถานะภาพ Grafana ทำหน้าที่นำข้อมูลจากส่วน Database มาแสดงผลเป็นกราฟข้อมูล ตรวจสอบความผิดปกติของค่าข้อมูลตามที่ใช้กำหนดเงื่อนไข

(Threshold) และแจ้งเตือนไปยังผู้เกี่ยวข้องผ่านแอปพลิเคชัน LINE และส่งข้อมูลไปยังส่วน Diagnostic and Recovery เพื่อประมวลผลต่อไป

4. Diagnostic and Recovery คือชุดคำสั่งไพทอน มีหน้าที่ตรวจสอบปัญหา และกู้คืนเว็บแอปพลิเคชัน โดยรับข้อมูลแจ้งความผิดปกติของระบบมาจากส่วน Monitoring แล้วทำการตรวจสอบส่วนประกอบของเว็บแอปพลิเคชันในเครื่องแม่ข่ายเบื้องต้นผ่านช่องทาง Secure Shell (SSH) เพื่อหาสาเหตุของความผิดปกติ และแก้ไขในกรณีที่สามารถดำเนินการได้

จากรูปที่ 3.1 ขั้นตอนการทำงานของระบบมีดังนี้

1. ส่วน Database ร้องขอข้อมูลสถานะภาพจากเครื่องแม่ข่ายเสมือน จากส่วน Data Collection ผ่านโปรโตคอล HTTP ทุก 5 วินาที
2. ส่วน Data Collection รับการร้องขอข้อมูลจากส่วน Database แล้วทำการติดต่อสื่อสารกับเครื่องแม่ข่ายเสมือน ผ่านโปรโตคอล SNMP, ICMP, TCP และ HTTP เพื่อตรวจสอบสถานะภาพ เมื่อได้รับข้อมูลจากเครื่องแม่ข่ายเสมือนแล้วนำประมวลผลข้อมูล ให้อยู่ในรูปแบบข้อมูลที่ Database รองรับ และส่งข้อมูลกลับไปยังส่วน Database
3. ส่วน Database ทำการประทับเวลาข้อมูลที่ได้รับ แล้วบันทึกข้อมูล
4. ส่วน Monitoring ทำการ Query ข้อมูลจากส่วน Database มาแสดงผลให้ผู้ใช้ในรูปแบบกราฟข้อมูล พร้อมทั้งติดตามความผิดปกติของค่าข้อมูล ตามที่ผู้ใช้งานกำหนดเงื่อนไข เมื่อพบความผิดปกติของค่าข้อมูล จะส่งการแจ้งเตือนไปยังผู้ดูแลระบบผ่านแอปพลิเคชัน LINE และส่งข้อมูลไปยังส่วน Diagnostic and Recovery ในรูปแบบ JSON ผ่านโปรโตคอล HTTP
5. ส่วน Diagnostic and Recovery รับและประมวลผลข้อมูล JSON เพื่อจำแนกประเภทของความผิดปกติ แล้วส่งคำสั่งไปยังเครื่องแม่ข่ายเสมือนที่ผิดปกติ ผ่าน SSH เพื่อตรวจสอบปัญหา และแก้ไขในกรณีที่สามารถทำได้ และส่งผลลัพธ์ไปยังผู้ดูแลระบบผ่านแอปพลิเคชัน LINE

3.3 รายละเอียดการดำเนินงาน

3.3.1 เตรียมเครื่องแม่ข่ายเสมือน

ในขั้นตอนการเตรียมเครื่องแม่ข่ายเสมือนสำหรับติดตั้งระบบติดตามสถานะภาพเครื่องแม่ข่ายเสมือน (Monitoring Server) มีข้อมูลจำเพาะดังนี้

- ระบบปฏิบัติการ Ubuntu 18.04
- ติดตั้ง Docker
- ติดตั้ง Python 3.6.9 และ Pip3

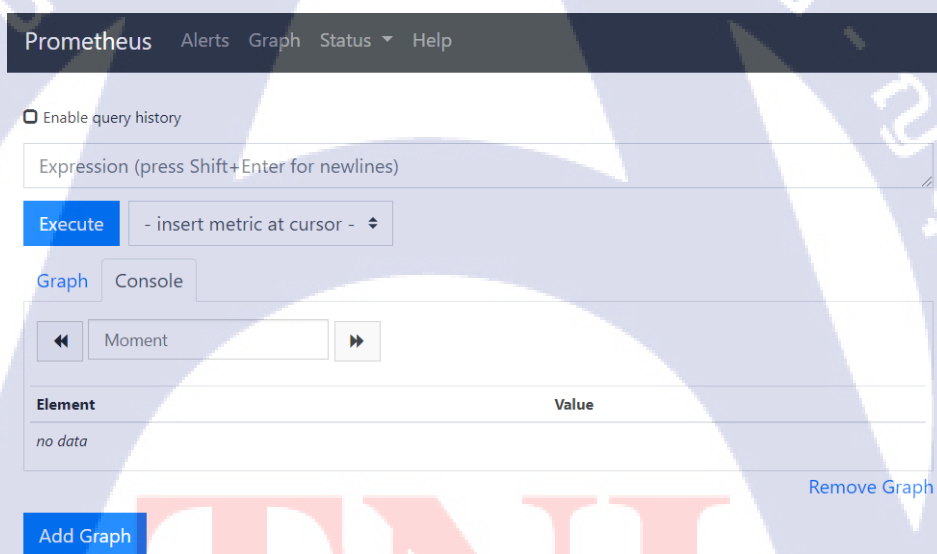
- ติดตั้ง SNMP Daemon

3.3.2 ติดตั้งฐานข้อมูลอนุกรมเวลา Prometheus

ในขั้นตอนการติดตั้งฐานข้อมูลอนุกรมเวลา Prometheus บนเครื่องแม่ข่ายเสมือนที่ได้เตรียมไว้ในข้อที่ 3.3.1 เป็นการติดตั้งแบบ Container ที่ทำงานด้วย Docker โดยโดยใช้คำสั่งดังนี้

```
docker run -d \
  -p 9090:9090 \
  -v /etc/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \
  --name prometheus prom/prometheus
```

เมื่อทำการติดตั้งเสร็จแล้ว สามารถทดสอบการเข้าใช้งาน Prometheus ผ่านทางเว็บเบราว์เซอร์ ผ่านทางพอร์ต 9090 เช่น <http://localhost:9090> จะปรากฏหน้าเว็บตามรูปที่ 3.2



รูปที่ 3.2 หน้าเว็บ Prometheus

ทำการตั้งค่า Prometheus ให้คอยร้องขอข้อมูลจากชุดคำสั่งตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือนที่ทำงานอยู่ที่พอร์ต 8001 ทุกๆ 5 วินาทีตามความต้องการของระบบ ทำให้ไม่ส่งผลกระทบต่อประสิทธิภาพของเครื่องแม่ข่ายเสมือนจนเกินไป และยังได้ค่าข้อมูลที่ต่อเนื่อง ในด้านผลกระทบต่อการแจ้งเตือนของการดึงข้อมูลทุกๆ 5 วินาทีคือ ผู้ใช้งานจะรับรู้ความผิดปกติช้ากว่าเวลาจริงไม่เกิน 5 วินาที โดยการตั้งค่าพอร์ต และช่วงเวลาดึงข้อมูลสามารถตั้งค่าด้วยการแก้ไขไฟล์ตั้งค่า

ของ Prometheus ที่ Path \etc\prometheus\prometheus.yml แล้วเพิ่มข้อความที่ด้านล่างสุด ดังนี้

```
- job_name: 'prometheus'
  scrape_interval: 5s
  static_configs:
  - targets: ['localhost:8001']
```

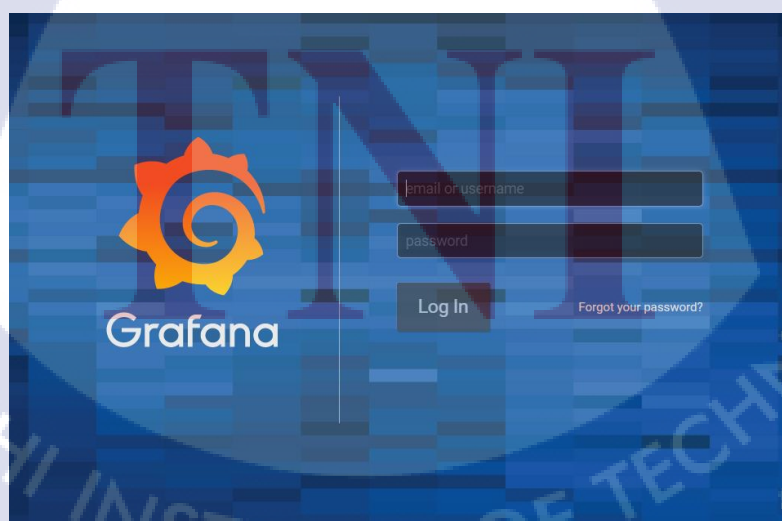
เมื่อตั้งค่าเสร็จแล้วให้ทำการ Restart Docker Container Prometheus โดยใช้คำสั่ง docker restart Prometheus

3.3.3 ติดตั้งเครื่องมือติดตามสถานะภาพ Grafana

ในขั้นตอนการติดตั้งฐานข้อมูลอนุกรมเวลา Grafana บนเครื่องแม่ข่ายเสมือนที่ได้เตรียมไว้ในข้อที่ 3.3.1 เป็นการติดตั้งแบบ Container ที่ทำงานด้วย Docker โดยใช้คำสั่งดังนี้

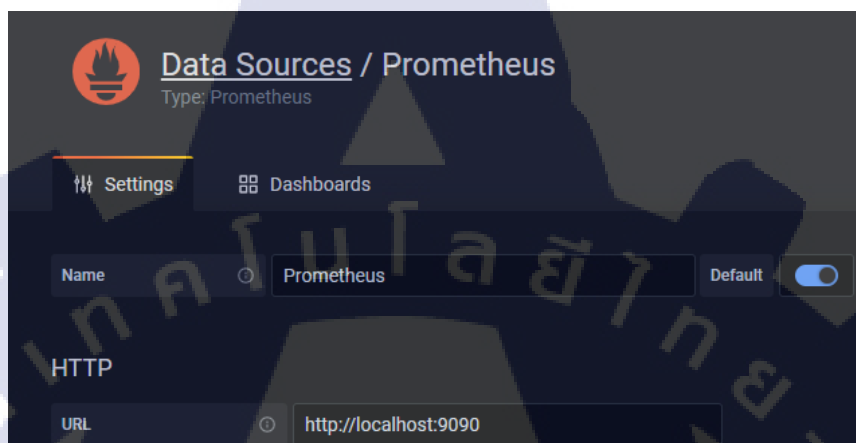
```
docker run -d \
  -p 3000:3000 \
  --name=grafana grafana/grafana
```

เมื่อทำการติดตั้งเสร็จแล้ว สามารถทดสอบการเข้าใช้งาน Grafana ผ่านทางเว็บเบราว์เซอร์ ผ่านทางพอร์ต 3000 เช่น <http://localhost:3000> จะปรากฏหน้าเว็บตามรูปที่ 3.3 ให้ Log In โดยใช้ username: admin และ password: admin ซึ่งเป็น user ค่าเริ่มต้นของ Grafana เมื่อเข้าไปแล้วให้ดำเนินการเปลี่ยน password เพื่อความปลอดภัย



รูปที่ 3.3 หน้าเว็บลงชื่อเข้าใช้ Grafana

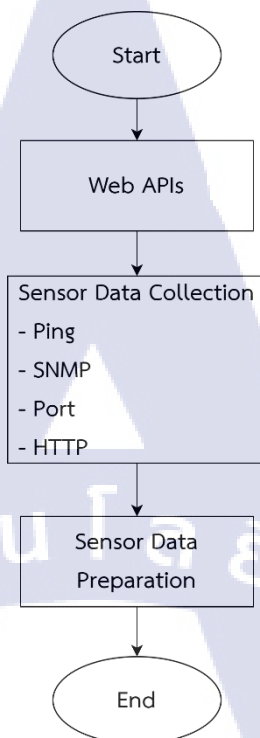
เชื่อมต่อ Grafana กับ Prometheus โดยไปที่เมนู Configuration > Datasource > Add data source แล้วเลือก Prometheus โดยในส่วนการตั้งค่า HTTP ในช่องของ URL ใส่ลิงก์ของ Prometheus ที่ติดตั้งไว้ก่อนหน้านี้เช่น <http://localhost:9090> แล้วเลือก Default ดังรูปที่ 3.4 แล้วกด Save & Test



รูปที่ 3.4 การตั้งค่าเชื่อมต่อ Grafana กับ Prometheus

3.3.4 พัฒนาชุดคำสั่งตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือน

พัฒนาชุดคำสั่งด้วยภาษาไพทอน ร่วมกับไลบรารีโปรโตคอล SNMP, ICMP, TCP และ HTTP ในลักษณะ Web Application เพื่อให้สามารถติดต่อสื่อสารกับ Prometheus ผ่านโปรโตคอล HTTP ได้โดยมีขั้นตอนการทำงานดังรูปที่ 3.5



รูปที่ 3.5 ขั้นตอนการทำงานของชุดคำสั่งตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือน

จากรูปที่ 3.5 มีรายละเอียดขั้นตอนการทำงานดังนี้

1. รับ Request จาก Prometheus ผ่าน Web APIs
2. รวบรวมข้อมูลสถานะภาพ โดยติดต่อสื่อสารกับเครื่องแม่ข่ายเสมือนที่ต้องการตรวจสอบสถานะภาพ (Monitoring Client) โดยใช้โปรโตคอล SNMP กำหนดค่า OID เพื่อระบุเฉพาะค่าข้อมูลที่ต้องการตามตารางที่ 3.2 ใช้โปรโตคอล ICMP เพื่อ Ping ตรวจสอบสถานะภาพการเชื่อมต่อเครือข่าย ใช้โปรโตคอล TCP ตรวจสอบสถานะภาพพอร์ตของแอปพลิเคชัน และใช้โปรโตคอล HTTP ตรวจสอบสถานะภาพของเว็บแอปพลิเคชันในเครื่องแม่ข่ายเสมือน
3. เตรียมข้อมูลที่ได้รับจากเครื่องแม่ข่ายเสมือน แล้วส่งข้อมูลกลับโดยจัดข้อมูลให้อยู่ในรูปแบบที่เหมาะสม ดังนี้
 - สกัดค่าข้อมูลจาก Text ให้เป็นตัวเลขเพื่อนำไปประมวลผล
 - ประมวลผลค่าข้อมูลสถานะภาพต่างๆ ตามตารางที่ 3.3
 - จัดข้อมูลให้อยู่ในรูปแบบ metric{label}value โดยกำหนดชื่อ metric และ label ให้กับค่าข้อมูล เช่น cpu_usage{IP="192.168.1.10"}12

ตารางที่ 3.2 ค่า OID ของข้อมูลสถานะภาพ สำหรับระบบปฏิบัติการ Ubuntu 18.04

OID	Name	คำอธิบาย
1.3.6.1.4.1.2021.11.11.0	ssCpuDle	% ของ CPU ที่ไม่ได้ถูกใช้งาน
1.3.6.1.4.1.2021.11.67	ssCpuNumCpus	จำนวนแกนของ CPU
1.3.6.1.4.1.2021.4.6.0	memAvailReal	ปริมาณ Memory ที่ว่าง
1.3.6.1.4.1.2021.4.5.0	memTotalReal	ปริมาณ Memory ทั้งหมด
1.3.6.1.4.1.2021.4.14.0	memBuffer	ปริมาณ Memory ที่ถูกจอง
1.3.6.1.4.1.2021.4.15.0	memCached	ปริมาณ Memory ที่ถูกใช้ชั่วคราว
1.3.6.1.4.1.2021.9.1.7.1	dskAvail	ปริมาณ Disk ที่ว่าง
1.3.6.1.4.1.2021.9.1.8.1	dskUsed	ปริมาณ Disk ที่ถูกใช้
1.3.6.1.2.1.31.1.1.1	ifHCInOctets	ปริมาณการรับข้อมูลจราจรเครือข่าย
1.3.6.1.2.1.31.1.1.1	ifHCOctets	ปริมาณการส่งข้อมูลจราจรเครือข่าย
1.3.6.1.2.1.31.1.1.1.1	ifName	รายชื่ออินเตอร์เฟซเครือข่าย

ตารางที่ 3.3 การประมวลผลค่าข้อมูลสถานะภาพ

ชื่อตัวแปร	คำอธิบาย	วิธีประมวลผล	ผลลัพธ์ค่าข้อมูล
cpu_percent	% ของซีพียูที่ถูกใช้งาน	$100 - \text{ssCpuDle}$	0 ถึง 100 %
mem_used	ปริมาณ Memory ที่ถูกใช้งาน	$\text{memTotalReal} - (\text{memAvailReal} + \text{memBuffer} + \text{memCached})$	มากกว่า 0 Byte
mem_percent	% ของ Memory ที่ถูกใช้งาน	$(\text{mem_used} / \text{memTotalReal}) \times 100$	0 ถึง 100 %
disk_total	ปริมาณ Disk ทั้งหมด	$\text{dskAvail} + \text{dskUsed}$	มากกว่า 0 Byte
disk_percent	ปริมาณ Disk ที่ถูกใช้งาน	$\text{dskUsed} / (\text{dskAvail} + \text{dskUsed}) \times 100$	0 ถึง 100 %
bits_received	ปริมาณการรับข้อมูลจราจรเครือข่าย	$\text{ifHCInOctets} \times 8$	มากกว่า 0 bits
bits_sent	ปริมาณการส่งข้อมูลจราจรเครือข่าย	$\text{ifHCOctets} \times 8$	มากกว่า 0 bits

ตารางที่ 3.3 การประมวลผลค่าข้อมูลสถานะภาพ (ต่อ)

ชื่อตัวแปร	คำอธิบาย	วิธีประมวลผล	ผลลัพธ์ค่าข้อมูล
ping	ผลลัพธ์จาก ICMP ping	Ping สำเร็จ กำหนดค่าเท่ากับ 1 Ping ล้มเหลว กำหนดค่าเท่ากับ 0	0 หรือ 1
port	ผลลัพธ์จาก TCP ping พอร์ต	Ping สำเร็จ กำหนดค่าเท่ากับ 1 Ping ล้มเหลว กำหนดค่าเท่ากับ 0	0 หรือ 1
http_code	ผลลัพธ์จาก HTTP Response	นอกเหนือจาก code 100-599 กำหนดค่าเท่ากับ 0	0 ถึง 599

```

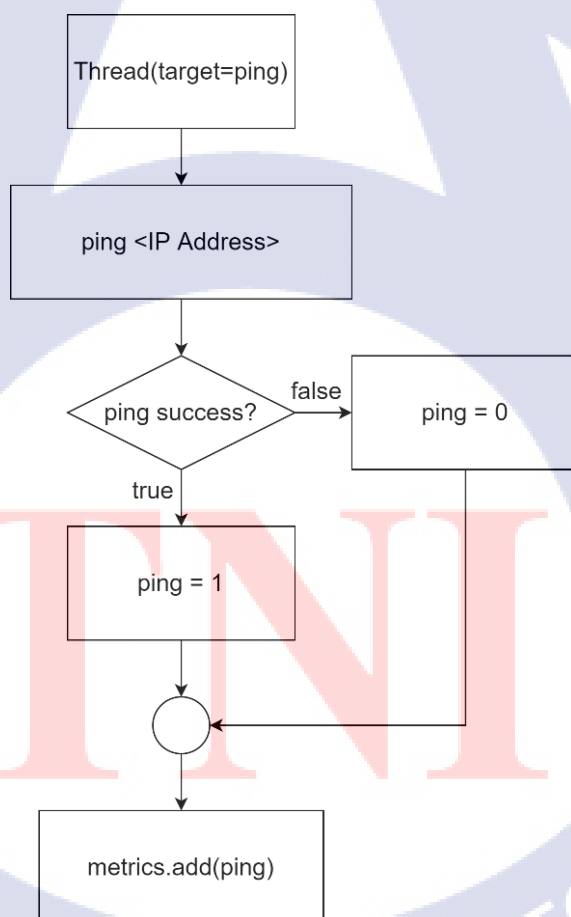
1  @app.route("/metrics", methods=["GET"])
2  def main():
3      global threads
4      threads = []
5      global metric
6      metric = ""
7
8      for i in host: # Thread Group 1 : Ping Status + SNMP
9          t = threading.Thread(target=ping, args=(i['ip'], i['url']))
10         threads.append(t)
11         t.start()
12
13     for i in host: # Thread Group 2 : Port Status
14         for j in range(len(i['port'])):
15             t = threading.Thread(target=port, args=(i['ip'], i['url'], i['port'][j]))
16             threads.append(t)
17             t.start()
18
19     for i in host: # Thread Group 3 : HTTP Status
20         t = threading.Thread(target=http, args=(i['ip'], i['url']))
21         threads.append(t)
22         t.start()
23
24     for t in threads:
25         t.join()
26
27     return metric
28
29 if __name__ == '__main__':
30     app.run(host="127.0.0.1", port=8001)

```

รูปที่ 3.6 ชุดคำสั่งฟังก์ชัน Web APIs

คำอธิบายรูปที่ 3.6

- บรรทัดที่ 1 และ 29-30 เป็นการสร้าง Web APIs ให้ทำงานที่พอร์ต 8001 โดยรอรับ Request จาก Prometheus ด้วย HTTP Method GET
- บรรทัดที่ 3-4 เป็นการสร้างตัวแปรแบบ List ชื่อ threads เพื่อเก็บข้อมูล thread
- บรรทัดที่ 5-6 เป็นการสร้างตัวแปรชื่อ metric เพื่อเก็บข้อมูลสถานะภาพต่างๆ
- บรรทัดที่ 8-25 เป็นการเรียกใช้ฟังก์ชันที่ทำหน้าที่เป็น Sensor Data Collection ต่างๆ แบบ Thread เพื่อความรวดเร็วในการรวบรวมข้อมูล ได้แก่ ping, port และ http
- บรรทัดที่ 27 เป็นการส่งค่าข้อมูลสถานะภาพทั้งหมดที่อยู่ใน metric กลับไปให้ Prometheus



รูปที่ 3.7 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน ping

คำอธิบายรูปที่ 3.7

- เรียกใช้งานฟังก์ชัน ping
- Ping ไปยังหมายเลข IP Address เครื่องแม่ข่ายเสมือน ตามที่ได้รับค่า
- ตรวจสอบค่าที่ได้รับจากการ Ping
- กรณี Ping สำเร็จ กำหนดค่าเท่ากับ 1 แต่กรณีไม่สำเร็จ กำหนดค่าเท่ากับ 0
- เพิ่มค่าข้อมูล ping เข้าไปใน List ชื่อ metrics เพื่อส่งข้อมูลกลับไปยัง Prometheus

```

1 def ping(ip, url):
2     if subprocess.call(["ping", "-c", "1", "-W", "1", ip]) == 0:
3         metric += "%s[ip=\"%s\",url=\"%s\"] %s \n" % ('ping', ip, url, '1')
4         try:
5             t = threading.Thread(target=cpu, args=(ip, url))
6             threads.append(t)
7             t.start()
8
9             t = threading.Thread(target=memory, args=(ip, url))
10            threads.append(t)
11            t.start()
12
13            t = threading.Thread(target=disk, args=(ip, url))
14            threads.append(t)
15            t.start()
16
17            t = threading.Thread(target=traffic, args=(ip, url))
18            threads.append(t)
19            t.start()
20
21        except:
22            pass
23    else:
24        metric += "%s[ip=\"%s\",url=\"%s\"] %s \n" % ('ping', ip, url, '0')

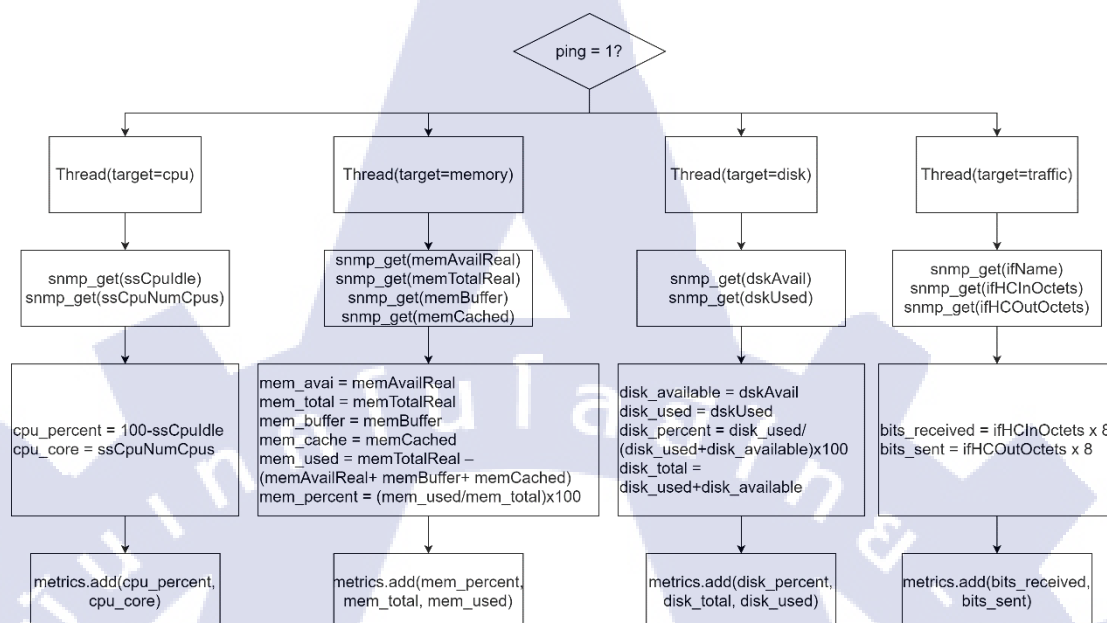
```

รูปที่ 3.8 ชุดคำสั่งฟังก์ชัน ping

คำอธิบายรูปที่ 3.8

- บรรทัดที่ 2 เป็นการใช้คำสั่ง Ping ไปยัง IP Address ของเครื่อง Monitoring Client ถ้าได้ค่าเท่ากับ 1 หมายถึง Ping สำเร็จยังสามารถติดต่อกับเครื่องปลายทางได้
- บรรทัดที่ 3 ในกรณีที่ Ping สำเร็จจะจัดรูปแบบข้อมูลและกำหนดค่า ping เท่ากับ 1
- บรรทัดที่ 4-22 ในกรณีที่ Ping สำเร็จ จะเรียกใช้ฟังก์ชัน cpu, memory, disk และ traffic เพื่อขอค่าสถานะภาพจากเครื่อง Monitoring Client

- บรรทัดที่ 23-24 ในกรณีที่ Ping ไม่สำเร็จจะจัดรูปแบบข้อมูลและกำหนดค่า ping เท่ากับ 0



รูปที่ 3.9 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน SNMP

คำอธิบายรูปที่ 3.9

- ในกรณีที่ ping มีค่าเท่ากับ 1 จะดำเนินการเรียกใช้ฟังก์ชัน cpu, memory, disk และ traffic
- ใช้โปรโตคอล SNMP ขอข้อมูลจากเครื่องแม่ข่ายเสมือน
- รับค่าที่ได้จากเครื่องแม่ข่ายเสมือน แล้วเตรียมข้อมูลให้อยู่ในรูปแบบที่เหมาะสม
- เพิ่มค่าข้อมูลเข้าไปใน List ชื่อ metrics เพื่อส่งข้อมูลกลับไปยัง Prometheus

```

1 def cpu(ip, url):
2     try:
3         cpu_core = snmp_walk('1.3.6.1.4.1.2021.11.67', hostname=ip, community=cms, version=2)
4         cpu_core = int(cpu_core[0].value)
5         cpu_percent = snmp_get('1.3.6.1.4.1.2021.11.11.0', hostname=ip, community=cms, version=2)
6         cpu_percent = str(cpu_percent).split()[1]
7         cpu_percent = 100-int(cpu_percent[7:-1])
8         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('cpu_percent', ip, url, cpu_percent)
9         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('cpu_core', ip, url, cpu_core)
10    except:
11        pass
12
13 def memory(ip, url):
14     try:
15         mem_avai = snmp_get('1.3.6.1.4.1.2021.4.6.0', hostname=ip, community=cms, version=2)
16         mem_total = snmp_get('1.3.6.1.4.1.2021.4.5.0', hostname=ip, community=cms, version=2)
17         mem_buffer = snmp_get('1.3.6.1.4.1.2021.4.14.0', hostname=ip, community=cms, version=2)
18         mem_cache = snmp_get('1.3.6.1.4.1.2021.4.15.0', hostname=ip, community=cms, version=2)
19         mem_avai = str(mem_avai).split()[1]
20         mem_avai = float(mem_avai[7:-1])
21         mem_total = str(mem_total).split()[1]
22         mem_total = float(mem_total[7:-1])
23         mem_buffer = str(mem_buffer).split()[1]
24         mem_buffer = float(mem_buffer[7:-1])
25         mem_cache = str(mem_cache).split()[1]
26         mem_cache = float(mem_cache[7:-1])
27         mem_used = mem_total-(mem_avai+mem_buffer+mem_cache)
28         mem_percent = (mem_used/mem_total)*100.0
29         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('mem_percent', ip, url, mem_percent)
30         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('mem_total', ip, url, mem_total)
31         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('mem_used', ip, url, mem_used)
32    except:
33        pass
34
35 def disk(ip, url):
36     try:
37         disk_available = snmp_get('1.3.6.1.4.1.2021.9.1.7.1', hostname=ip, community=cms, version=2)
38         disk_available = str(disk_available).split()[1]
39         disk_available = float(disk_available[7:-1])
40         disk_used = snmp_get('1.3.6.1.4.1.2021.9.1.8.1', hostname=ip, community=cms, version=2)
41         disk_used = str(disk_used).split()[1]
42         disk_used = float(disk_used[7:-1])
43         disk_percent = disk_used/(disk_used+disk_available)*100.0
44         disk_total = disk_used+disk_available
45         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('disk_total', ip, url, disk_total)
46         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('disk_percent', ip, url, disk_percent)
47         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('disk_available', ip, url, disk_available)
48         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('disk_used', ip, url, disk_used)
49    except:
50        pass
51
52 def traffic(ip, url):
53     try:
54         ifHCInOctets = snmp_walk('1.3.6.1.2.1.31.1.1.1.6', hostname=ip, community=cms, version=2)
55         ifHCOutOctets = snmp_walk('1.3.6.1.2.1.31.1.1.1.10', hostname=ip, community=cms, version=2)
56         IfName = snmp_walk('1.3.6.1.2.1.31.1.1.1.1', hostname=ip, community=cms, version=2)
57         for i in range(len(IfName)):
58             metric += "%s{ip=\"%s\",url=\"%s\",interface=\"%s\"} %s\n"
59             % ('bits_received', ip, url, str(IfName[i].value), str(int(ifHCInOctets[i].value)*8))
60             metric += "%s{ip=\"%s\",url=\"%s\",interface=\"%s\"} %s\n"
61             % ('bits_sent', ip, url, str(IfName[i].value), str(int(ifHCOutOctets[i].value)*8))
62    except:
63        pass

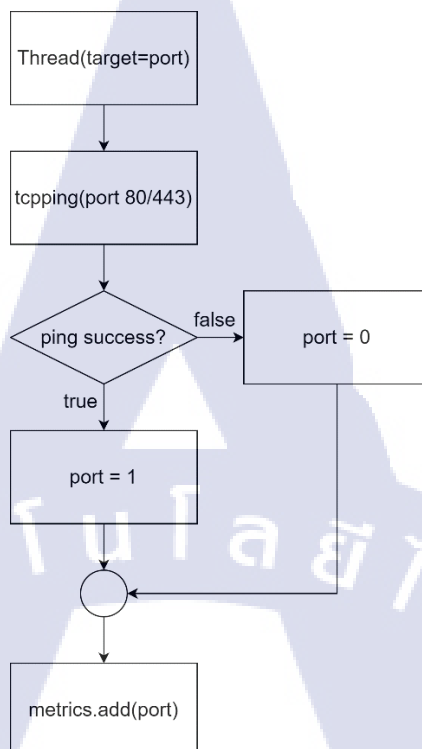
```

รูปที่ 3.10 ชุดคำสั่งฟังก์ชัน SNMP

คำอธิบายรูปที่ 3.10

- บรรทัดที่ 3 และ 5 เป็นการใช้ Library easySNMP เพื่อขอข้อมูลสถานะภาพของซีพียู
- บรรทัดที่ 4 และ 6-7 เป็นการสกัดค่าข้อมูลสถานะภาพที่ได้รับจากเครื่อง Monitoring Client

- บรรทัดที่ 8-9 เป็นการจัดรูปแบบข้อมูลและกำหนดค่า `cpu_percent` และ `cpu_core` เท่ากับข้อมูลที่ได้รับ
- บรรทัดที่ 15-18 เป็นการใช้ Library `easySNMP` เพื่อขอข้อมูลสถานะภาพของหน่วยความจำหลัก
- บรรทัดที่ 19-28 เป็นการสกัดค่า และประมวลผลข้อมูลสถานะภาพหน่วยความจำหลัก
- บรรทัดที่ 29-31 เป็นการจัดรูปแบบข้อมูลและกำหนดค่า `mem_percent`, `mem_total` และ `mem_used`
- บรรทัดที่ 37 และ 40 เป็นการใช้ Library `easySNMP` เพื่อขอข้อมูลสถานะภาพของหน่วยความจำสำรอง
- บรรทัดที่ 38-39 และ 42-44 เป็นการสกัดค่า และประมวลผลข้อมูลสถานะภาพหน่วยความจำสำรอง
- บรรทัดที่ 45-48 เป็นการจัดรูปแบบข้อมูลและกำหนดค่า `disk_percent`, `disk_total`, `disk_available` และ `disk_used`
- บรรทัดที่ 54-56 เป็นการใช้ Library `easySNMP` เพื่อขอข้อมูลสถานะภาพของปริมาณการรับ-ส่งข้อมูลจราจรเครือข่าย
- บรรทัดที่ 57-61 เป็นการจัดรูปแบบข้อมูลและกำหนดค่า `bits_received` และ `bits_sent`



รูปที่ 3.11 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน port

คำอธิบายรูปที่ 3.11

- เรียกใช้งานฟังก์ชัน port
- Ping ไปยังหมายเลข Port 80 และ 443 ของเครื่องแม่ข่ายเสมือน
- ตรวจสอบค่าที่ได้รับจากการ Ping
- กำหนดค่าข้อมูล port ตามเงื่อนไขการตรวจสอบ
- เพิ่มค่าข้อมูล port เข้าไปใน List ชื่อ metrics เพื่อส่งข้อมูลกลับไปยัง Prometheus

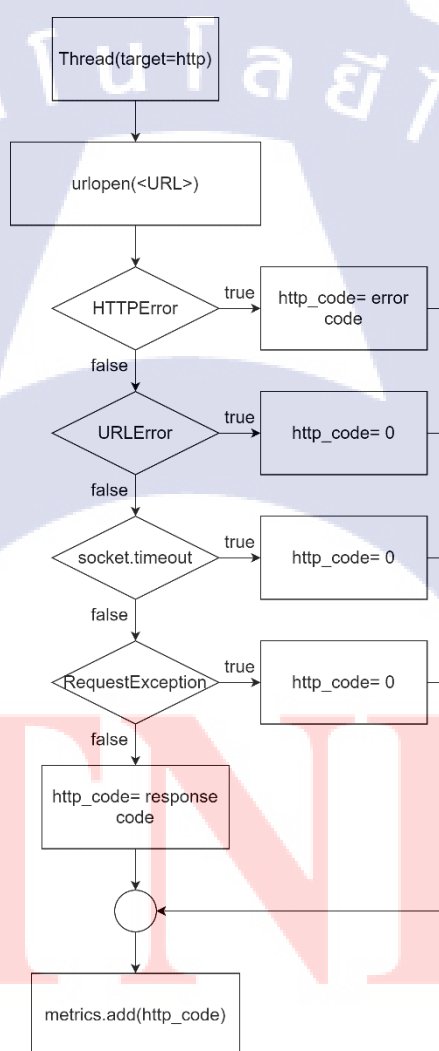
```

1 def port(ip, url, port):
2     global metric
3     try:
4         if tcpping(ip, port=port, timeout=1):
5             metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url, port, '1')
6         else:
7             metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url, port, '0')
8     except:
9         metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url, port, '0')
  
```

รูปที่ 3.12 ชุดคำสั่งฟังก์ชัน port

คำอธิบายรูปที่ 3.12

- บรรทัดที่ 4 เป็นการใช้ Library tcpping เพื่อ ping ไปยังพอร์ตของ เครื่อง Monitoring Client
- บรรทัดที่ 5 ในกรณีที่ Ping สำเร็จจะจัดรูปแบบข้อมูลและกำหนดค่า port เท่ากับ 1
- บรรทัดที่ 6-9 ในกรณีที่ Ping ไม่สำเร็จ หรือกรณีอื่นๆจะจัดรูปแบบข้อมูลและกำหนดค่า port เท่ากับ 0



รูปที่ 3.13 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน http

คำอธิบายรูปที่ 3.13

- เรียกใช้งานฟังก์ชัน http
- เข้าถึงเว็บไซต์ที่ต้องการตรวจสอบสถานะ จาก URL ที่ได้รับค่า
- ตรวจสอบค่าข้อมูลเว็บไซต์ที่ได้รับ
- กำหนดค่าให้ตัวแปร http_code ตามเงื่อนไขการตรวจสอบค่า
- เพิ่มค่าข้อมูล http_code เข้าไปใน List ชื่อ metrics เพื่อส่งข้อมูลกลับไปยัง Prometheus

```

1 def http(ip, url):
2     global metric
3     try:
4         response = urllib.request.urlopen('http://'+url, timeout = 1)
5     except HTTPError as e:
6         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('http_code', ip, url, e.code)
7     except URLError as e:
8         metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('http_code', ip, url, '0')
9     except socket.timeout:
10        metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('http_code', ip, url, '0')
11    except requests.exceptions.RequestException:
12        metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('http_code', ip, url, '0')
13        raise Exception('Failed to connect to %s' % url) from None
14    else:
15        metric += "%s{ip=\"%s\",url=\"%s\"} %s\n" % ('http_code', ip, url, response.code)

```

รูปที่ 3.14 ชุดคำสั่งฟังก์ชัน HTTP

คำอธิบายรูปที่ 3.14

- บรรทัดที่ 4 เป็นการใช้ Library urllib ส่ง request ไปยัง URL ที่กำหนดเพื่อตรวจสอบสถานะภาพ HTTP Response Code
- บรรทัดที่ 5-6 ในกรณีที่รับค่าประเภท HTTPError จัดรูปแบบข้อมูลและกำหนดค่า http_code เท่ากับ Error Code ที่ได้รับ
- บรรทัดที่ 7-8 ในกรณีที่รับค่าประเภท URLError จัดรูปแบบข้อมูลและกำหนดค่า http_code เท่ากับ 0
- บรรทัดที่ 9-10 ในกรณีที่ Timeout จัดรูปแบบข้อมูลและกำหนดค่า http_code เท่ากับ 0
- บรรทัดที่ 11-13 ในกรณีอื่นๆ จัดรูปแบบข้อมูลและกำหนดค่า http_code เท่ากับ 0

- บรรทัดที่ 14-15 ในกรณีที่ได้รับค่าปกติ จัดรูปแบบข้อมูลและกำหนดค่า http_code เท่ากับ HTTP Response Code ที่ได้รับ

หลังจากพัฒนาชุดคำสั่งเสร็จแล้ว จึงดำเนินการสร้าง Service ชื่อ monitor เพื่อให้ชุดคำสั่งทำงานตลอดเวลา โดยใช้คำสั่ง vi /etc/systemd/system/monitor.service แล้วแก้ไขไฟล์โดยใส่ข้อความดังนี้

```
[Unit]
Description=Monitoring Service
After=multi-user.target
[Service]
Type=simple
ExecStart=/usr/bin/python3 /path/to/script/python.py
[Install]
WantedBy=multi-user.target
```

หลังจากนั้นใช้คำสั่ง systemctl daemon-reload และคำสั่ง systemctl enable monitor.service เพื่อเพิ่ม Service เข้าไปในระบบ Ubuntu หลังจากนั้นเริ่มการทำงานของ Service โดยใช้คำสั่ง systemctl start monitor.service ผลลัพธ์คือ ชุดคำสั่งทำงานที่พอร์ต 8001 พร้อมรับการเรียกใช้งานจาก Prometheus ตลอดเวลา

3.3.5 สร้างกราฟข้อมูล และการแจ้งเตือน

นำค่าข้อมูลจาก Prometheus ที่ได้เชื่อมต่อกันไว้จากข้อ 3.3.3 มาสร้างกราฟข้อมูลใน Grafana โดยแสดงค่าข้อมูล และกำหนดเงื่อนไขการแจ้งเตือนตามมาตรฐานขององค์กรที่ผู้ศึกษาทำงานอยู่ เพื่อส่งการแจ้งเตือนไปยังผู้เกี่ยวข้องผ่านทาง LINE และส่วน Diagnostic and Recovery ตามตารางที่ 3.4 และท้ายที่สุดจะได้ผลลัพธ์การแสดงผลภาพดังรูปที่ 3.21

ตารางที่ 3.4 การแสดงผลสถานะภาพ และเงื่อนไขการแจ้งเตือน

ชื่อ Metric	ค่าข้อมูล	เงื่อนไขการแจ้งเตือน
ping	สถานะภาพของการเชื่อมต่อเครือข่าย	ค่าข้อมูล != 1
http_code	รหัสสถานะภาพ HTTP ของเว็บแอปพลิเคชัน	ค่าข้อมูล != 200
port	สถานะภาพของพอร์ต	ค่าข้อมูล != 1
cpu_percent	เปอร์เซ็นต์การใช้งานซีพียู	ค่าข้อมูล > 80
mem_percent	เปอร์เซ็นต์การใช้งานหน่วยความจำหลัก	ค่าข้อมูล > 80
disk_percent	เปอร์เซ็นต์การใช้งานหน่วยความจำสำรอง	ค่าข้อมูล > 80
cpu_core	จำนวนแกนซีพียู	-
mem_total	ปริมาณหน่วยความจำหลัก	-
disk_total	ปริมาณหน่วยความจำสำรอง	-
mem_used	ปริมาณหน่วยความจำหลักที่ถูกใช้	-
disk_used	ปริมาณหน่วยความจำสำรองที่ถูกใช้	-
bits_received	ปริมาณการรับข้อมูลจราจรเครือข่าย	-
bits_sent	ปริมาณการส่งข้อมูลจราจรเครือข่าย	-

โดยในส่วนของการสร้างกราฟ ผู้ศึกษาได้แบ่งกราฟข้อมูลออกเป็น 6 ส่วน ซึ่งในแต่ละส่วนจะแสดงค่าสถานะภาพแตกต่างกันไปดังนี้

1. ส่วน Status แสดงค่าข้อมูล ping, http_code และ port ในรูปแบบของตัวอักษร (Stat) โดยตั้งค่า Threshold ตามเงื่อนไขในตารางที่ 3.4 ถ้าค่าข้อมูลปกติจะแสดงสีพื้นหลังสีเขียวเพราะเป็นสีที่สื่อถึงความปกติเป็นมิตร แต่กรณีค่าผิดปกติจะแสดงสีพื้นหลังสีแดงเพราะเป็นสีที่สังเกตได้ง่ายและสื่อถึงอันตราย และในกราฟค่าข้อมูล ping กับ port ให้ทำการ Mapping value โดยกำหนดค่าข้อมูล 1 เท่ากับ UP และค่าข้อมูล 0 เท่ากับ Down ดังรูปที่ 3.15

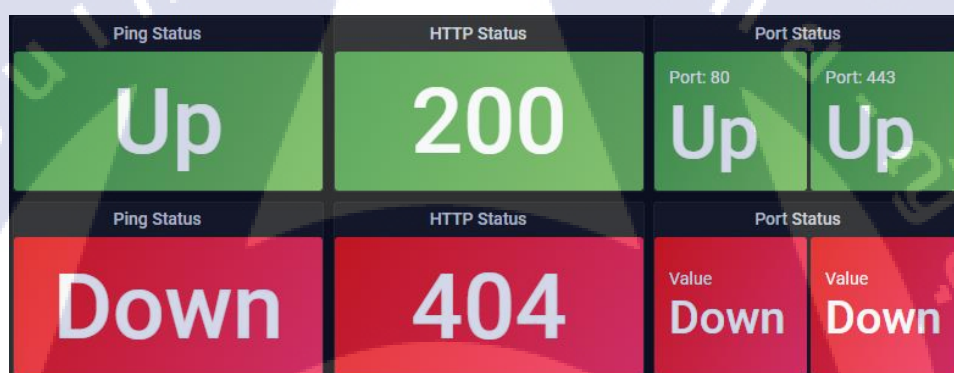
2. ส่วน Status History แสดงค่าข้อมูล ping, http_code และ port ในรูปแบบของ Graph เพื่อแสดงให้เห็นการเปลี่ยนแปลงของค่าข้อมูลตามช่วงเวลาทำให้สามารถระบุได้ว่ามีความผิดปกติในช่วงเวลาใด ดังรูปที่ 3.16 และตั้งค่าเงื่อนไขการแจ้งเตือนตามตารางที่ 3.4

3. ส่วน Resource แสดงค่าข้อมูล cpu_core, mem_total, disk_total, mem_used และ disk_used ในรูปแบบของตัวอักษร (Stat) เพื่อแสดงให้เห็นทรัพยากรทั้งหมดที่มีของเครื่องแม่ข่ายเสมือน และทรัพยากรที่ถูกใช้ไป ดังรูปที่ 3.17

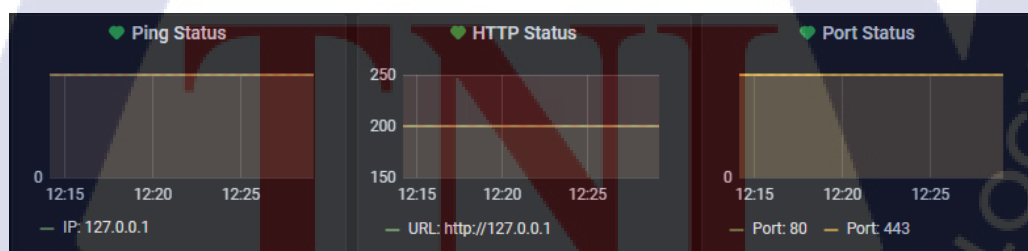
4. ส่วน Resource Utilization แสดงค่าข้อมูล `cpu_percent`, `mem_percent` และ `disk_percent` ในรูปแบบของตัวอักษร (Stat) โดยตั้งค่า Threshold ตามเงื่อนไขในตารางที่ 3.3 ถ้าค่าข้อมูลปกติจะแสดงสีพื้นหลังสีเขียว เพราะเป็นสีที่สื่อถึงความปกติดูเป็นมิตร แต่กรณีค่าผิดปกติจะแสดงสีพื้นหลังสีแดงเพราะเป็นสีที่สังเกตได้ง่ายและสื่อถึงอันตราย ดังรูปที่ 3.18

5. ส่วน Resource Utilization History แสดงค่าข้อมูล `cpu_percent`, `mem_percent` และ `disk_percent` ในรูปแบบของ Graph เพื่อแสดงให้เห็นการเปลี่ยนแปลงของค่าข้อมูลตามช่วงเวลาทำให้สามารถระบุได้ว่ามีความผิดปกติในช่วงเวลาใด ดังรูปที่ 3.19 และตั้งค่าเงื่อนไขการแจ้งเตือนตามตารางที่ 3.4

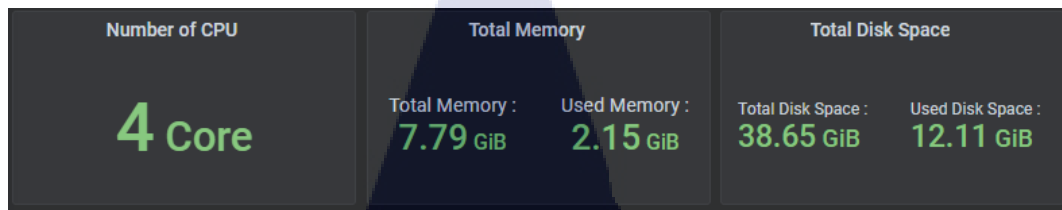
6. ส่วน Network แสดงค่าข้อมูล `bits_received` และ `bits_sent` ในรูปแบบของตัวอักษร (Stat) และ Graph ดังรูปที่ 3.20



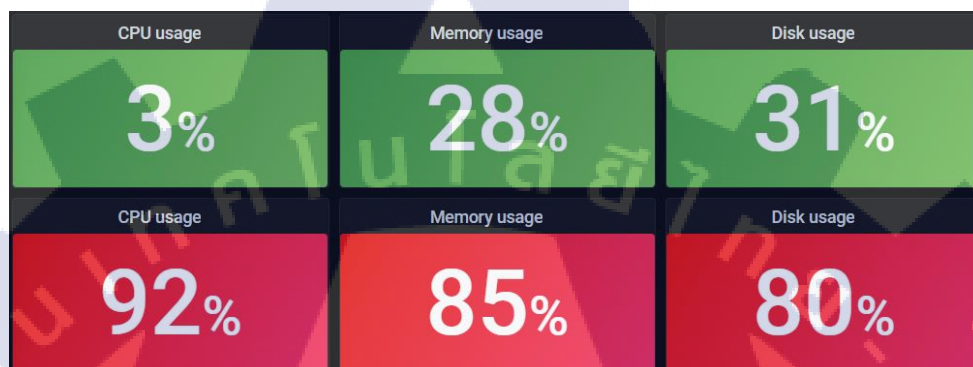
รูปที่ 3.15 ตัวอย่างการแสดงผลของส่วน Status ค่าปกติ และค่าผิดปกติ



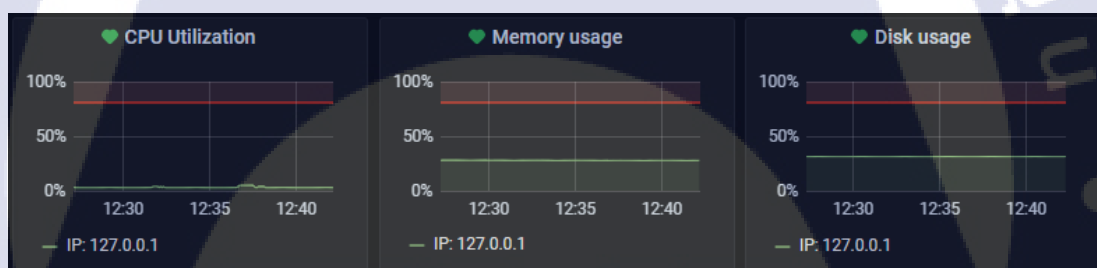
รูปที่ 3.16 ตัวอย่างการแสดงผลของส่วน Status History



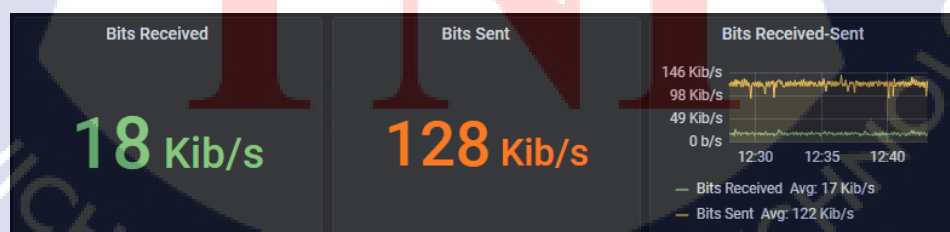
รูปที่ 3.17 ตัวอย่างการแสดงผลของส่วน Resource



รูปที่ 3.18 ตัวอย่างการแสดงผลของส่วน Resource Utilization ค่าปกติ และค่าผิดปกติ



รูปที่ 3.19 ตัวอย่างการแสดงผลของส่วน Resource Utilization History

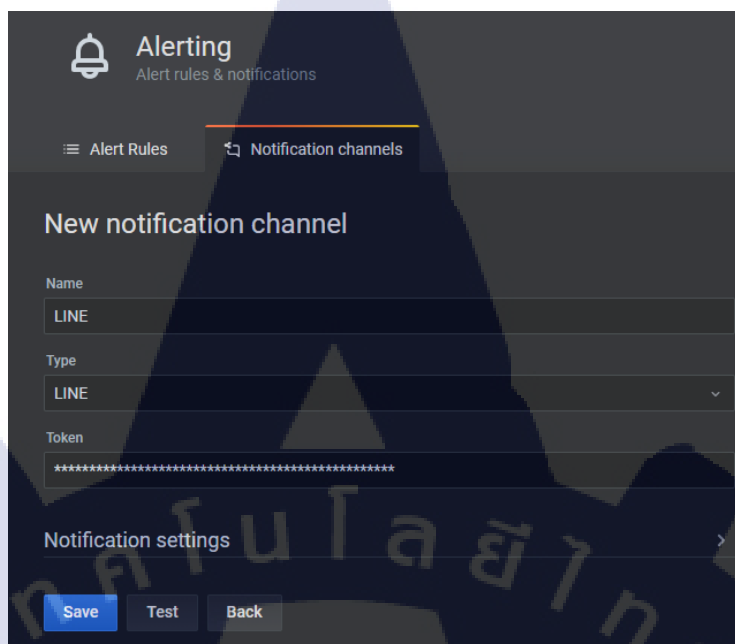


รูปที่ 3.20 ตัวอย่างการแสดงผลของส่วน Network



รูปที่ 3.21 ตัวอย่างการแสดงผลกราฟข้อมูลใน Grafana

ตั้งค่าการแจ้งเตือนไปยัง LINE ซึ่งต้องสมัครใช้บริการ LINE Notify ก่อน เมื่อได้รับ Token แล้ว นำไปตั้งค่าที่เมนู Alerting > Notification channels แล้วคลิก New channel แล้วดำเนินการตั้งค่าดังรูปที่ 3.22 โดยในส่วนของ Token ให้นำ Token ที่ได้จากการสมัคร LINE Notify มาใส่ เมื่อตั้งค่าเสร็จแล้วกด Save



Alerting
Alert rules & notifications

Alert Rules Notification channels

New notification channel

Name
LINE

Type
LINE

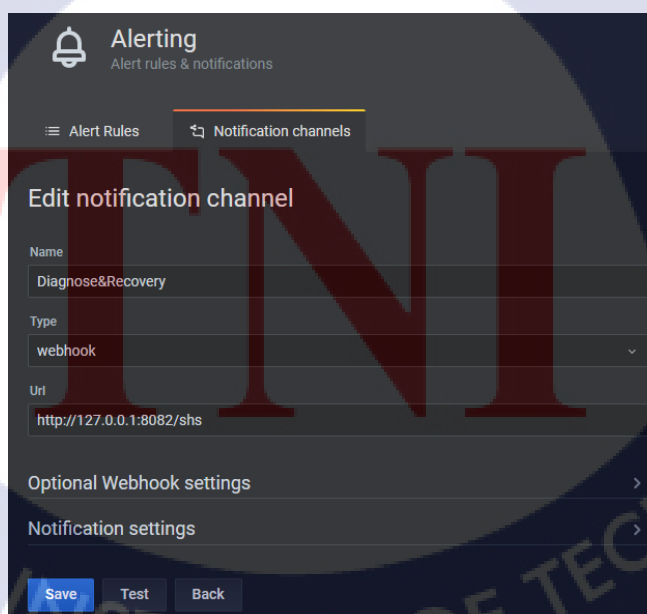
Token

Notification settings

Save Test Back

รูปที่ 3.22 การตั้งค่าแจ้งเตือนไปยัง LINE

ตั้งค่าการแจ้งเตือนไปยังส่วน Diagnostic and Recovery คลิก New channel แล้วดำเนินการตั้งค่าดังรูปที่ 3.23 โดยในส่วนของ URL กำหนด URL ของส่วน Diagnostic and Recovery ที่พัฒนาขึ้น <http://127.0.0.1:8082/shs> เมื่อตั้งค่าเสร็จแล้วกด Save



Alerting
Alert rules & notifications

Alert Rules Notification channels

Edit notification channel

Name
Diagnose&Recovery

Type
webhook

Uri
<http://127.0.0.1:8082/shs>

Optional Webhook settings

Notification settings

Save Test Back

รูปที่ 3.23 การตั้งค่าแจ้งเตือนไปยังส่วน Diagnostic and Recovery

ดำเนินการกำหนดเงื่อนไขการแจ้งเตือนที่กราฟส่วน Status History และ Resource Utilization History ดังรูปที่ 3.16 และ 3.19 โดยตั้งค่าการแจ้งเตือนที่ Alert ดังรูปที่ 3.24 กำหนดข้อความที่ต้องการแจ้งเตือนที่ส่วน Name ในส่วน Conditions กำหนดเงื่อนไขตามตารางที่ 3.3 และในส่วน Notifications เพิ่ม Notification channels ที่ได้ทำการตั้งค่าไว้แล้วทั้ง LINE และ Diagnose&Recovery เมื่อเสร็จแล้วจะปรากฏเส้นสีแดงที่กราฟข้อมูลคือ Threshold ที่ตั้งค่าไว้ เมื่อค่าข้อมูลเกินเส้นสีแดง Grafana จะส่งการแจ้งเตือนไปยังช่องทางที่กำหนด

The screenshot displays the Grafana Alerting Rule configuration interface. At the top, there are tabs for 'Query' (1), 'Transform' (0), and 'Alert' (1). The 'Rule' section includes a 'Name' field with the value 'ข้อความแจ้งเตือน', an 'Evaluate every' field set to '5s', and a 'For' field set to '10s'. The 'Conditions' section shows a 'WHEN' clause with 'last ()' followed by 'OF query (A, 10s, now) IS OUTSIDE RANGE' with values '1' and '1'. The 'No Data & Error Handling' section has two rows, both with 'SET STATE TO' and 'Keep Last State' dropdowns. The 'Notifications' section shows 'Send to' with 'LINE' and 'Diagnose&Recovery' channels.

รูปที่ 3.24 การกำหนดเงื่อนไขการแจ้งเตือน

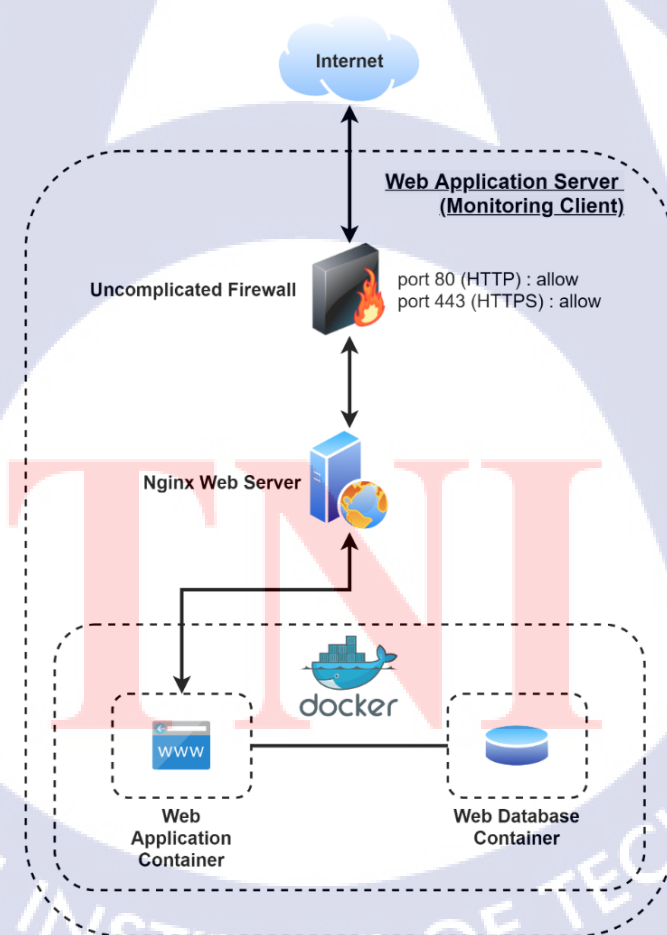
3.3.6 พัฒนาชุดคำสั่ง Diagnostic and Recovery

ในการศึกษานี้เครื่องมือช่วยเสมือนที่ต้องการติดตามสถานะภาพ เป็นเครื่องให้บริการเว็บแอปพลิเคชัน ซึ่งได้มีการติดตามสถานะภาพผ่านโปรโตคอล HTTP อยู่แล้ว ผู้ศึกษาได้สังเกตเห็นประโยชน์จากการแจ้งเตือนความผิดปกติ ที่ได้รับจากระบบติดตามสถานะภาพ มาประยุกต์ต่อยอดโดยการพัฒนาส่วนตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชัน เพื่อช่วยผู้ดูแลระบบให้ทราบถึงสาเหตุของปัญหาเบื้องต้น และกู้คืนระบบอัตโนมัติ โดยมีขั้นตอนการดำเนินงานดังนี้

1. ศึกษาส่วนประกอบของระบบ

ผู้ศึกษาได้ทำการวิเคราะห์ส่วนประกอบสำคัญที่เกี่ยวข้องกับการทำงานของเว็บแอปพลิเคชัน โดยมีโครงสร้างระบบตามรูปที่ 3.25 และมีส่วนประกอบดังนี้

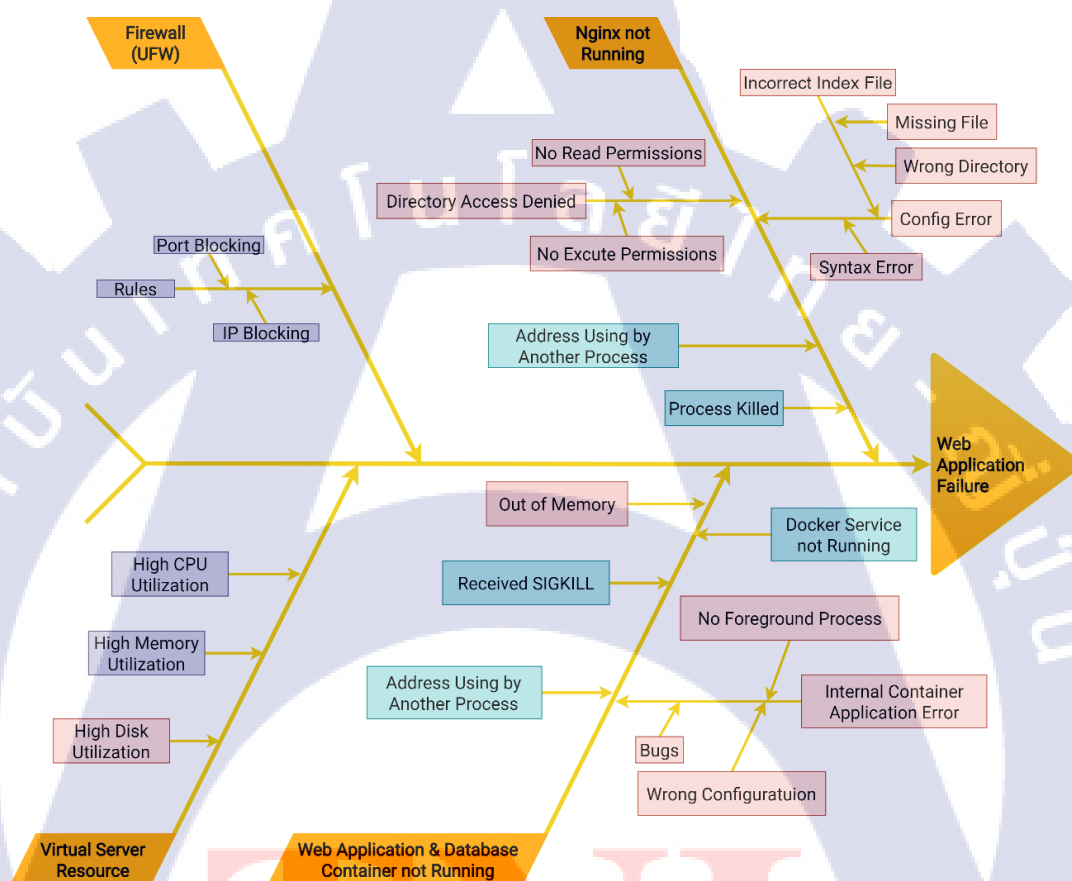
- Nginx เป็น Web Server Software ที่ทำหน้าที่เป็นตัวกลางการเชื่อมต่อสื่อสารระหว่าง Client กับ Web Application
- Docker เป็น Software Container ทำหน้าที่ควบคุมการทำงานของ Container
- Web Application Container ทำหน้าที่เป็นเว็บแอปพลิเคชัน
- Web Database Container ทำหน้าที่เป็นฐานข้อมูลเว็บ
- Uncomplicated Firewall (UFW) เป็น Firewall Software ทำหน้าที่ควบคุมช่องทางการเข้าออกของข้อมูลผ่านเครือข่าย



รูปที่ 3.25 โครงสร้างภายในของเครื่องแม่ข่ายเสมือนที่ให้บริการเว็บแอปพลิเคชัน

2. Root Cause Analysis

ในการทำงานของเว็บแอปพลิเคชัน หากส่วนประกอบที่กล่าวข้างต้นมีปัญหาจะทำให้เว็บแอปพลิเคชันไม่สามารถทำงานได้อย่างปกติ ผู้ศึกษาจึงทำการวิเคราะห์หาสาเหตุที่แท้จริง (Root Cause Analysis) ที่จะส่งผลให้ส่วนประกอบของเว็บแอปพลิเคชันมีปัญหา เพื่อนำมาพัฒนาชุดคำสั่ง Diagnostic and Recovery ต่อไป โดยได้ผลลัพธ์จากการวิเคราะห์ดังรูปที่ 3.26



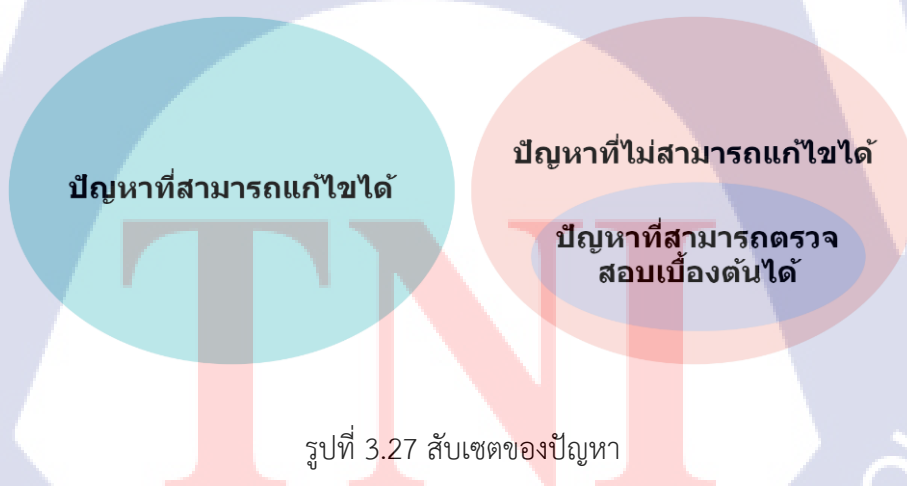
รูปที่ 3.26 Root Cause Analysis ของเว็บแอปพลิเคชัน

จากรูปที่ 3.26 คือส่วนประกอบสำคัญในการทำงานของเว็บแอปพลิเคชันมี 4 ส่วนหลักได้แก่

1. Firewall (UFW)
2. Nginx
3. Web Application & Database Container
4. Virtual Server Resource

จากส่วนประกอบหลักสามารถแบ่งปัญหาออกเป็น 3 ประเภท ดังรูปที่ 3.27 คือ

1. ปัญหาที่สามารถแก้ไขได้ ด้วยชุดคำสั่ง (สียฟ้า) รายละเอียดตามตารางที่ 3.5
2. ปัญหาที่ไม่สามารถแก้ไขได้ ด้วยชุดคำสั่ง (สีแดง)เนื่องจากเป็นปัญหาละเอียดอ่อน เช่น บั๊กของโปรแกรม การตั้งค่า ข้อมูล หรือสิทธิ์การเข้าถึงข้อมูล ซึ่งเป็นสิ่งที่ซับซ้อน และต้องอาศัยความเข้าใจ จึงควรให้ผู้ดูแลระบบตรวจสอบด้วยตัวเอง
3. ปัญหาที่สามารถตรวจสอบเบื้องต้นได้ คือปัญหาที่ไม่สามารถแก้ไข แต่ตรวจสอบเบื้องต้นได้โดยใช้ชุดคำสั่ง (สีม่วง) ซึ่งปัญหานี้ชุดคำสั่งจะทำหน้าที่ตรวจสอบ รวบรวมข้อมูลที่เกี่ยวข้องให้ผู้ดูแลระบบทราบ เพื่อเป็นข้อมูลเบื้องต้นในการนำไปดำเนินการแก้ไขปัญหาต่อไป โดยมีรายละเอียดตามตารางที่ 3.5



รูปที่ 3.27 สับเซตของปัญหา

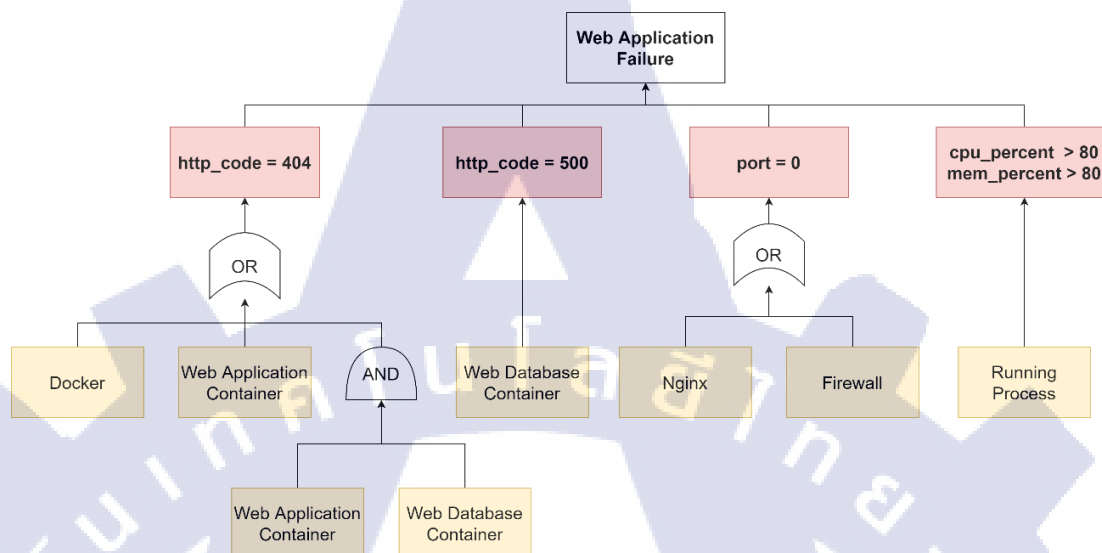
ตารางที่ 3.5 รายการปัญหาที่สามารถใช้ชุดคำสั่งช่วยแก้ไขปัญหา หรือตรวจสอบเบื้องต้น

ส่วนประกอบ	ปัญหา	สาเหตุ
Web Application Container	Container not Running: แก้ไขได้	- Received SIGKILL - Address Using by Another Process - Docker Service not Running
Web Database Container	Container not Running: แก้ไขได้	- Received SIGKILL - Address Using by Another Process - Docker Service not Running
Nginx	Service not Running: แก้ไขได้	- Process killed - Address Using by Another Process
Firewall	- Port Blocking: ตรวจสอบได้ - IP Blocking: ตรวจสอบได้	- Firewall Rule Configuration
Virtual Server Resource	- High CPU Utilization: ตรวจสอบได้ - High Memory Utilization: ตรวจสอบได้	- Running Process

3. หาความสัมพันธ์ระหว่างส่วนประกอบ กับค่าสถานะภาพ

เนื่องจากชุดคำสั่งที่พัฒนามีหน้าที่ตรวจสอบ และกู้คืนระบบเมื่อมีปัญหา รับค่าข้อมูลปัญหาจากส่วนติดตามสถานะภาพซึ่งคอยตรวจสอบสถานะภาพต่างๆของเครื่อง Monitoring Client ตามตารางที่ 3.4 เช่น ค่า http_code, port ,cpu_percent และmem_percent ซึ่งค่า

ข้อมูลเหล่านี้เป็นตัวบ่งชี้ว่าส่วนประกอบภายในเครื่อง Monitoring Client มีปัญหา ดังนั้นจึงต้องทำการวิเคราะห์ความสัมพันธ์ เพื่อนำมาวางแผนพัฒนาชุดคำสั่ง



รูปที่ 3.28 ความสัมพันธ์ระหว่างส่วนประกอบ กับค่าสถานะภาพ

จากรูปที่ 3.28 แสดงความสัมพันธ์ระหว่างส่วนประกอบ กับค่าสถานะภาพ (สีแดง) ที่สามารถบ่งชี้ส่วนประกอบที่มีปัญหา (สีเหลือง) ที่ส่งผลกระทบทำให้เว็บแอปพลิเคชันไม่สามารถใช้งานได้ โดยสามารถสรุปได้ตามตารางที่ 3.6 ดังนี้

ตารางที่ 3.6 การจำแนกค่าสถานะภาพ ส่วนที่เกี่ยวข้อง และการดำเนินการ

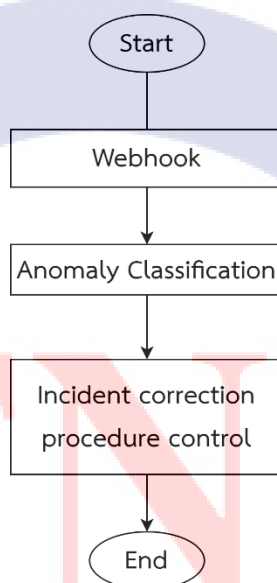
ค่าสถานะภาพ	ส่วนที่เกี่ยวข้อง	การดำเนินการ
http_code = 404	- Docker - Web Application Container - Web Database Container	ตรวจสอบ และกู้คืน - Docker - Web Application container - Web database container
http_code = 500	Web Database Container	ตรวจสอบ และกู้คืน Web Database Container

ตารางที่ 3.6 การจำแนกค่าสถานะภาพ ส่วนที่เกี่ยวข้อง และการดำเนินการ (ต่อ)

ค่าสถานะภาพ	ส่วนที่เกี่ยวข้อง	การดำเนินการ
Port = 0	- Nginx - Firewall	ตรวจสอบ และกู้คืน Nginx ตรวจสอบ Firewall Rule
cpu_percent > 80	Running Process	ตรวจสอบ Running Process ที่ใช้ปริมาณซีพียูมากที่สุด
mem_percent > 80	Running Process	ตรวจสอบ Running Process ที่ใช้ปริมาณหน่วยความจำหลักมากที่สุด

4. พัฒนาชุดคำสั่ง

พัฒนาชุดคำสั่งด้วยภาษาไพทอน ร่วมกับไลบรารีโปรโตคอล SSH สำหรับใช้ในการติดต่อสื่อสารกับเครื่องแม่ข่ายเสมือนที่ผิดปกติ เพื่อตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชัน โดยมีขั้นตอนการทำงานตามรูปที่ 3.29



รูปที่ 3.29 ขั้นตอนการทำงานของชุดคำสั่งตรวจสอบความผิดปกติ และกู้คืน

จากรูปที่ 3.29 มีรายละเอียดขั้นตอนการทำงานดังนี้

1. รับข้อมูลแจ้งความผิดปกติจาก Grafana ในรูปแบบ JSON ด้วย Webhook

2. จำแนกค่าความผิดปกติของค่าสถานะภาพเพื่อระบุส่วนที่เกี่ยวข้องกับความผิดปกติตามตารางที่ 3.6

3. ดำเนินการตรวจสอบ และกู้คืนส่วนที่เกี่ยวข้อง แล้วแจ้งผลลัพธ์ไปยังผู้ดูแลระบบผ่าน LINE Notify

```

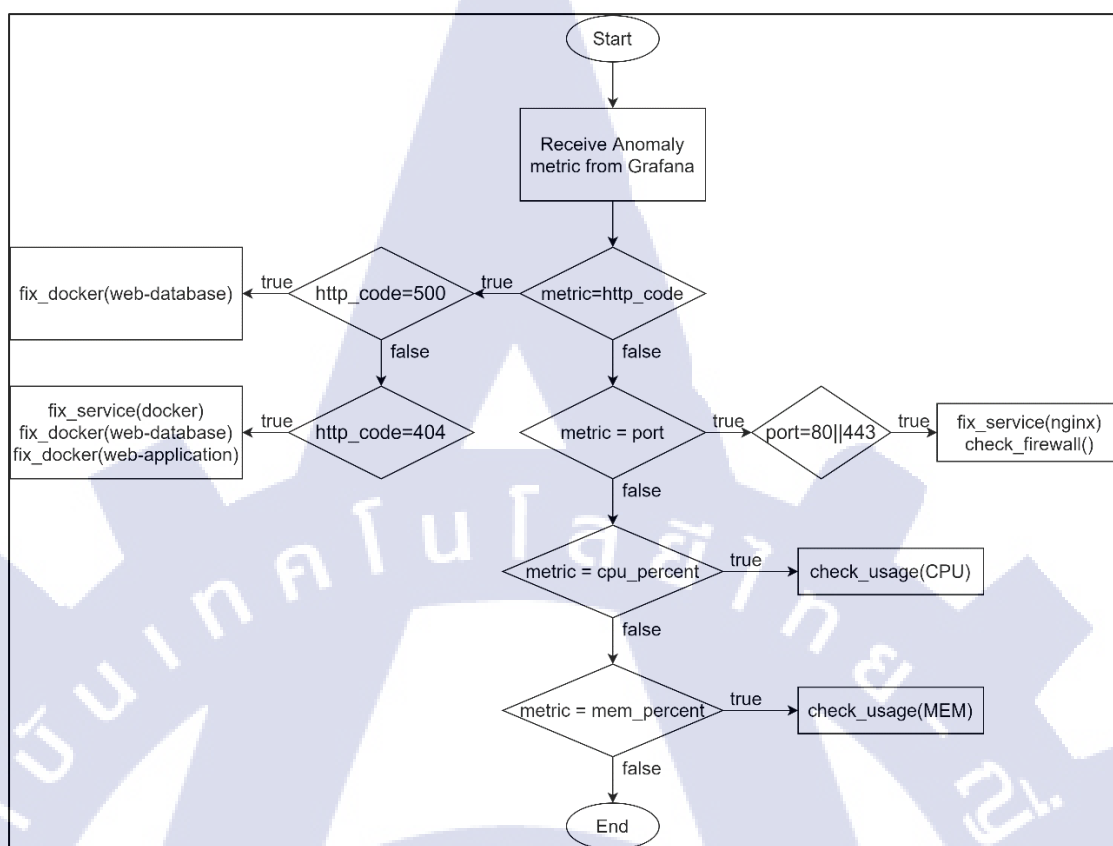
1  @app.route('/', methods=['POST'])
2  def respond():
3
4      incident={
5          "metric": request.json['evalMatches'][0]['tags']['__name__'],
6          "port": request.json['evalMatches'][0]['tags']['port'],
7          "value": str(request.json['evalMatches'][0]['value']),
8          "ip": request.json['evalMatches'][0]['tags']['ip'],
9          "url": request.json['evalMatches'][0]['tags']['url'],
10         "timestamp": datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S')
11     }
12
13     try:
14         return Response(status=200)
15     finally:
16         classify(incident)
17
18 if __name__ == '__main__':
19     app.run(host="127.0.0.1", port=8082)

```

รูปที่ 3.30 ชุดคำสั่งฟังก์ชัน Webhook

คำอธิบายรูปที่ 3.30

- บรรทัดที่ 1 และ 18-19 เป็นการสร้าง Webhook ให้ทำงานที่พอร์ต 8082 โดยรอรับข้อมูลความผิดปกติจาก Grafana ที่ส่งมาด้วย HTTP Method POST
- บรรทัดที่ 4-11 เป็นการนำค่าข้อมูลที่ได้รับจาก Grafana ใส่เข้าไปในตัวแปรแบบ Dictionary ที่ชื่อ incident เพื่อให้ง่ายต่อการนำไปใช้ดำเนินการ
- บรรทัดที่ 14 เป็นการตอบกลับไปยัง Grafana ว่าได้รับข้อมูลเรียบร้อยแล้ว
- บรรทัดที่ 16 เป็นการส่งตัวแปรชื่อ incident ที่มีค่าข้อมูลความผิดปกติไปดำเนินการจำแนกข้อมูลต่อไปในฟังก์ชัน Anomaly Classification



รูปที่ 3.31 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Anomaly Classification

คำอธิบายรูปที่ 3.31

- รับข้อมูลความผิดปกติจาก Grafana
- จำแนกชื่อของค่าข้อมูล
- จำแนกค่าข้อมูล
- เรียกใช้งานฟังก์ชันที่เกี่ยวข้องกับความผิดปกติ

```

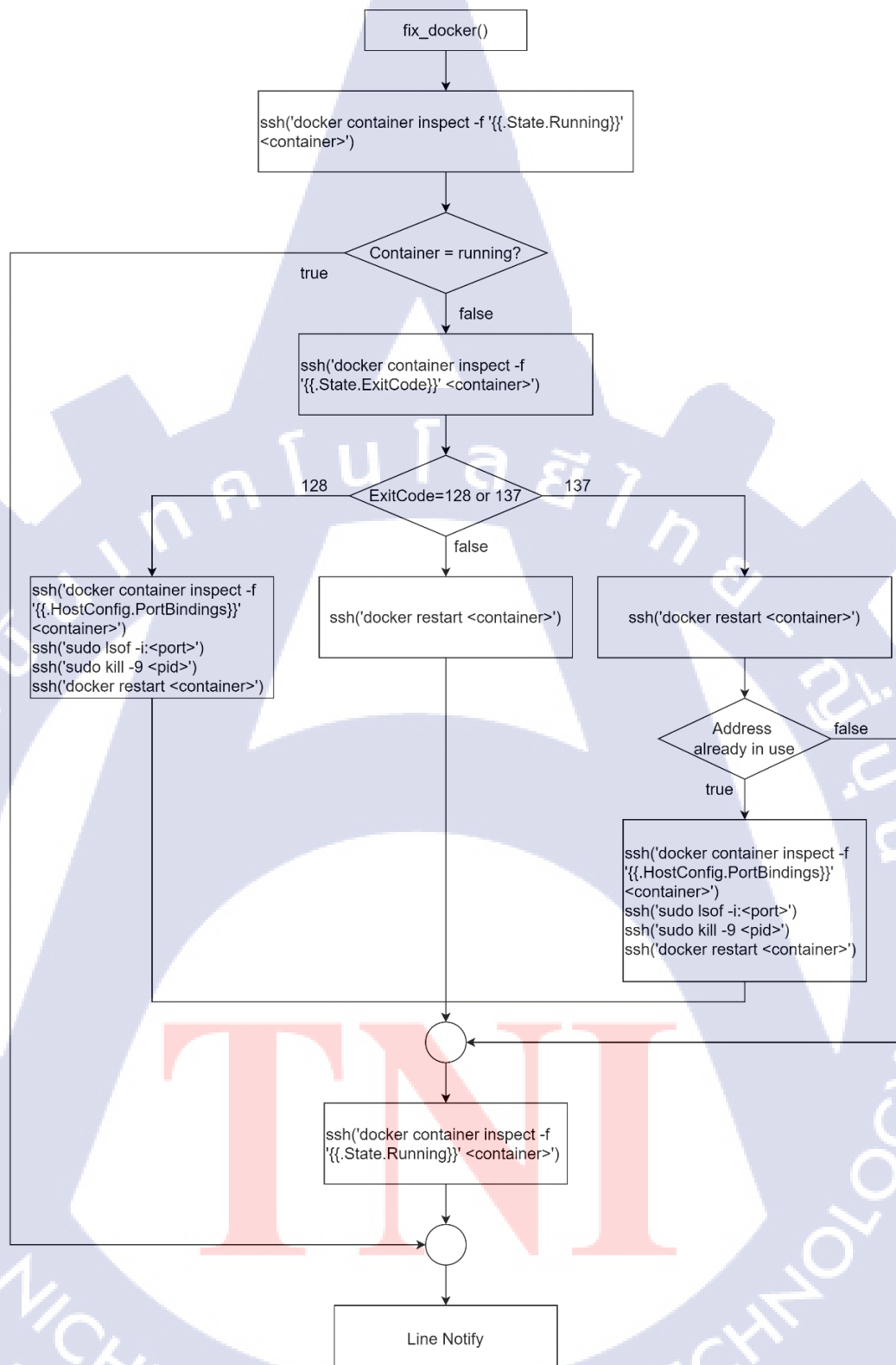
1 def classify(incident):
2     if incident['metric']=='http_code':
3         if incident['value']=='500':
4             fix_docker(incident, 'web-database')
5         elif incident['value']=='404':
6             fix_service(incident, 'docker')
7             fix_docker(incident, 'web-application')
8             fix_docker(incident, 'web-database')
9         else:
10            pass
11    elif incident['metric']=='port':
12        if incident['port']=='80' or incident['port']=='443':
13            fix_service(incident, 'nginx')
14            check_firewall(incident)
15    elif incident['metric']=='cpu_percent':
16        check_usage(incident, 'CPU')
17    elif incident['metric']=='mem_percent':
18        check_usage(incident, 'MEM')
19    else:
20        pass

```

รูปที่ 3.32 ชุดคำสั่งฟังก์ชัน Anomaly Classification

คำอธิบายรูปที่ 3.32

- บรรทัดที่ 2 เป็นการจำแนกค่า http_code
- บรรทัดที่ 3-4 กรณี http_code เท่ากับ 500 จะส่งชื่อ container web-database ไปยังฟังก์ชัน fix_docker เพื่อดำเนินการตรวจสอบ และกู้คืน
- บรรทัดที่ 5-8 กรณี http_code เท่ากับ 404 จะส่งชื่อ service docker ไปยังฟังก์ชัน fix_service และส่งชื่อ container web-application กับ web-database ไปยังฟังก์ชัน fix_docker เพื่อดำเนินการตรวจสอบ และกู้คืน
- บรรทัดที่ 11 เป็นการจำแนกค่า port
- บรรทัดที่ 12-14 กรณีค่า port เท่ากับ 80 หรือ 443 จะส่งชื่อ service nginx ไปยังฟังก์ชัน fix_service เพื่อดำเนินการตรวจสอบ และกู้คืน และส่งหมายเลข port ไปตรวจสอบ Rule ที่ฟังก์ชัน check_firewall
- บรรทัดที่ 15-16 กรณีเป็นค่า cpu_percent หรือ mem_percent จะเรียกใช้ฟังก์ชัน check_usage เพื่อตรวจสอบ Running Process



รูปที่ 3.33 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker)

คำอธิบายรูปที่ 3.33

- เรียกใช้งานฟังก์ชัน fix_docker
- ตรวจสอบสถานะของ Container โดยใช้โปรโตคอล SSH
- กรณีที่ Container ไม่ทำงาน ดำเนินการตรวจสอบ ExitCode ของ Container
- ดำเนินการ Run Command ผ่าน SSH เพื่อกู้คืน Container
- ตรวจสอบสถานะของ Container ครั้งสุดท้าย
- แจ้งผลลัพธ์ไปยังผู้ดูแลระบบ ผ่าน LINE Notify

```

1 def fix_docker(incident, container):
2     ssh.connect(incident['ip'], port=22, username='username', password='password')
3     msg="IP= %s\n" % ('192.168.xxx.xxx')+"Container Name= %s\n" % (container)
4     stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
5     -f '{{.State.Running}}' "+container)
6     running = stdout.readlines()
7     if "No such container" in str(running):
8         running=running[-1].splitlines()
9         error_msg=running[0]
10    else:
11        running=running[0].splitlines()
12        running=str(running[0])
13    if running=='false':
14        stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
15        -f '{{.State.ExitCode}}' "+container)
16        exitcode = stdout.readlines()
17        exitcode=exitcode[0].splitlines()
18        exitcode=str(exitcode[0])
19        msg=msg+"Container Running= %s\n" % (running)
20        msg=msg+"Container Error= %s\n" % (exitcode)
21        if exitcode=='137':
22            stdin, stdout, stderr = ssh.exec_command("sudo sync; echo 2 > /proc/sys/vm/drop_caches")
23            stdin, stdout, stderr = ssh.exec_command("sudo docker restart "+container)
24            errorcheck=stdout.readlines()
25            if "Error" in str(errorcheck):
26                if "address already in use" in str(errorcheck):
27                    stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
28                    -f '{{.HostConfig.PortBindings}}' "+container)
29                    hostport = stdout.readlines()
30                    hostport=hostport[0].splitlines()
31                    hostport=hostport[0].split(' ')
32                    hostport=hostport[1].split(':')
33                    hostport=hostport[0]
34                    stdin, stdout, stderr = ssh.exec_command("sudo lsof -i :"+hostport)
35                    address_check=stdout.readlines()
36                    address_check=address_check[1].split()
37                    msg=msg+"Container Error= %s\n" % ('Address already in use')
38                    msg=msg+"Using Command= %s\n" % (address_check[0])
39                    msg=msg+"Using PID= %s\n" % (address_check[1])
40                    msg=msg+"Using By= %s\n" % (address_check[2])
41                    detail={}
42                    detail['reason']='Address already in use'
43                    detail['using_command']=address_check[0]
44                    detail['using_pid']=address_check[1]
45                    detail['using_by']=address_check[2]
46                    incident['detail']=detail
47                    stdin, stdout, stderr = ssh.exec_command("sudo kill -9 "+address_check[1])
48                    time.sleep(1)
49                    stdin, stdout, stderr = ssh.exec_command("sudo docker restart "+container)
50                    errorcheck=stdout.readlines()
51                else:
52                    pass
53            errorcheck=errorcheck[-1].splitlines()
54            error_msg=errorcheck[0]

```

รูปที่ 3.34 ชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker)

บรรทัดที่ 1-54

```

55 elif exitcode=='128':
56     stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
57     -f '{{.HostConfig.PortBindings}}' "+container)
58     hostport = stdout.readlines()
59     hostport=hostport[0].splitlines()
60     hostport=hostport[0].split(' ')
61     hostport=hostport[1].split('}')
62     hostport=hostport[0]
63     stdin, stdout, stderr = ssh.exec_command("sudo lsof -i :"+hostport)
64     address_check=stdout.readlines()
65     address_check=address_check[1].split()
66     msg=msg+"Container Error= %s\n"%(address_check[0])
67     msg=msg+"Using Command= %s\n"%(address_check[0])
68     msg=msg+"Using PID= %s\n"%(address_check[1])
69     msg=msg+"Using By= %s\n"%(address_check[2])
70     detail={}
71     detail['reason']='Address already in use'
72     detail['using_command']=address_check[0]
73     detail['using_pid']=address_check[1]
74     detail['using_by']=address_check[2]
75     incident['detail']=detail
76     stdin, stdout, stderr = ssh.exec_command("sudo kill -9 "+address_check[1])
77     time.sleep(1)
78     stdin, stdout, stderr = ssh.exec_command("sudo docker restart "+container)
79     errorcheck=stdout.readlines()
80     if "Error" in str(errorcheck):
81         errorcheck=errorcheck[-1].splitlines()
82         error_msg=errorcheck[0]
83     else:
84         stdin, stdout, stderr = ssh.exec_command("sudo docker restart "+container)
85         errorcheck=stdout.readlines()
86         if "Error" in str(errorcheck):
87             errorcheck=errorcheck[-1].splitlines()
88             error_msg=errorcheck[0]
89     time.sleep(1)
90     stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
91     -f '{{.State.Running}}' "+
92     +container)
93     finalcheck = stdout.readlines()
94     finalcheck=finalcheck[0].splitlines()
95     finalcheck=str(finalcheck[0])
96     msg=msg+"---Recovery Result ---\n"
97     if finalcheck=='false':
98         msg=msg+"Container Running= %s\n"%(finalcheck)
99         msg=msg+"Error Msg= %s\n"%(error_msg)
100     elif finalcheck=='true':
101         msg=msg+"Container Running= %s\n"%(finalcheck)
102         msg=msg+"Solve Time= %s\n"%(datetime.fromtimestamp(time.time())
103         .strftime('%d-%m-%Y %H:%M:%S'))
104     else:
105         msg=msg+"Container Running= %s\n"%(finalcheck)
106     elif running=='true':
107         msg=msg+"Container Running= %s\n"%(running)
108     else:
109         msg=msg+"Error Msg= %s\n"%(error_msg)
110     notify(msg)
111     ssh.close()

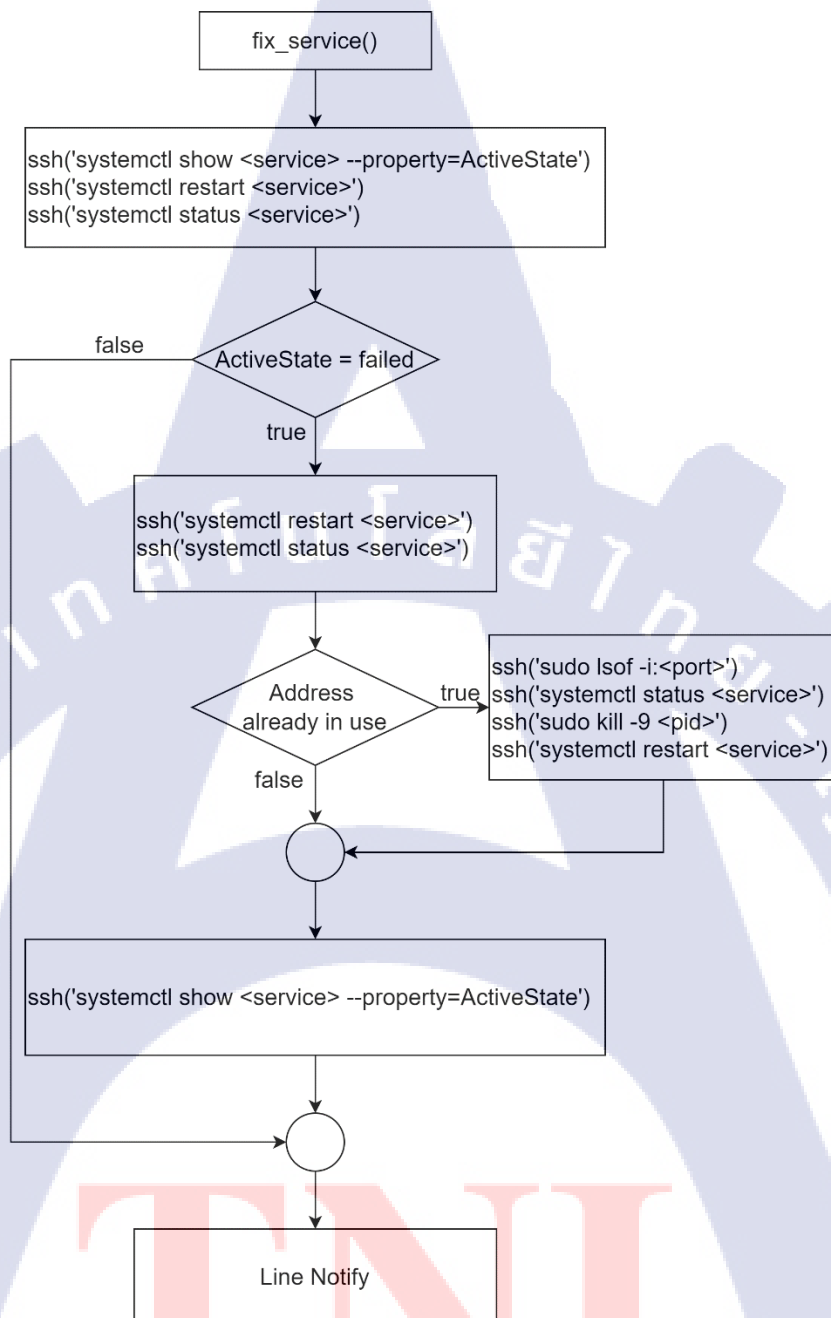
```

รูปที่ 3.35 ชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_docker)
บรรทัดที่ 55-111

คำอธิบายรูปที่ 3.34 และ 3.35

- บรรทัดที่ 2 เป็นการเชื่อมต่อเครื่องแม่ข่ายเสมือนที่พบปัญหาผ่านโปรโตคอล SSH
- บรรทัดที่ 4-12 เป็นการตรวจสอบสถานะภาพของ Container ว่าทำงานอยู่หรือไม่
- บรรทัดที่ 13-18 กรณีที่ Container ไม่ทำงาน จะใช้คำสั่งตรวจสอบ Exitcode ของ Container
- บรรทัดที่ 19-20 เป็นการรวบรวมข้อมูลปัญหาที่พบเพื่อส่งไปยัง LINE

- บรรทัดที่ 21-23 กรณี Exitcode เท่ากับ 137 หมายถึงปัญหาเกิดจาก Received SIGKILL จะดำเนินการใช้คำสั่ง Restart Container ให้กลับมาทำงานอีกครั้ง
- บรรทัดที่ 24 อ่านค่าผลลัพธ์จากการ Restart Container
- บรรทัดที่ 25 และ 53-54กรณีมี Error จะดำเนินการรวบรวมข้อมูลปัญหาที่พบเพื่อส่งไปยัง LINE
- บรรทัดที่ 26-52 กรณีค่าผลลัพธ์มี Error “address already in use” จะดำเนินการตรวจสอบพอร์ตที่ Container ทำงาน แล้วตรวจสอบ และหยุดการทำงานของ Process ที่กำลังใช้งานพอร์ตนั่นอยู่ หลังจากนั้น Restart Container แล้วอ่านค่าผลลัพธ์
- บรรทัดที่ 55-82 กรณี Exitcode เท่ากับ 128 หมายถึงปัญหาเกิดจาก Address Using by Another Process จะดำเนินการตรวจสอบพอร์ตที่ Container ทำงาน แล้วตรวจสอบ และหยุดการทำงานของ Process ที่กำลังใช้งานพอร์ตนั่นอยู่ หลังจากนั้น Restart Container แล้วรวบรวมข้อมูลปัญหาที่พบเพื่อส่งไปยัง LINE
- บรรทัดที่ 83-88 กรณี Exitcode อื่นๆ จะดำเนินการใช้คำสั่ง Restart Container แล้วอ่านค่าผลลัพธ์
- บรรทัดที่ 89-95 เป็นการตรวจสอบสถานะภาพของ Container ว่ากลับมาทำงานปกติหรือไม่
- บรรทัดที่ 96-110 เป็นการนำข้อมูลปัญหาที่รวบรวมไว้ และผลลัพธ์การดำเนินการส่งไปยังผู้ดูแลระบบทาง LINE
- บรรทัดที่ 111 เป็นการปิดการเชื่อมต่อ SSH กับเครื่องแม่ข่ายเสมือน



รูปที่ 3.36 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (fix_service)

คำอธิบายรูปที่ 3.36

- เรียกใช้งานฟังก์ชัน fix_service
- ตรวจสอบสถานะของ Service โดยใช้โปรโตคอล SSH
- กรณีที่ Service ไม่ทำงาน ดำเนินการ Run Command ผ่าน SSH เพื่อกู้คืน
- ตรวจสอบสถานะของ Service ครั้งสุดท้าย
- แจ้งผลลัพธ์ไปยังผู้ดูแลระบบ ผ่าน LINE Notify

```

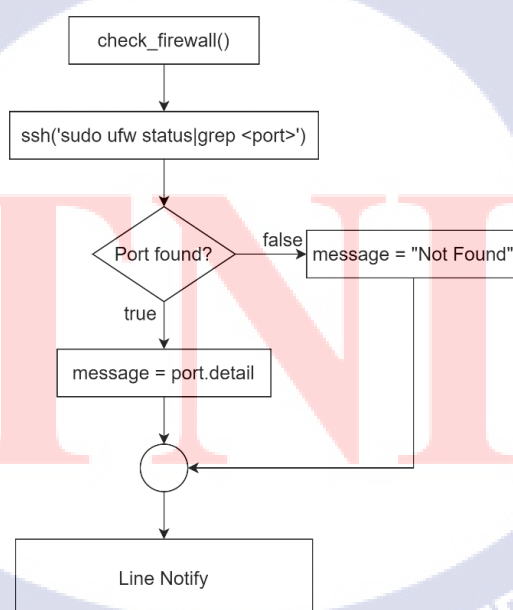
1 def fix_service(incident, service):
2     ssh.connect(incident['ip'], port=22, username='username', password='password')
3     check=[]
4     msg="IP= %s\n"%('192.168.xxx.xxx')+ "Service Name= %s\n"%(service)
5     stdin, stdout, stderr = ssh.exec_command('sudo systemctl show '+service+' --property=ActiveState')
6     running=stdout.readlines()
7     running=running[0].splitlines()
8     running=ansi_escape.sub('', running[1]).split('=')
9     running=running[1]
10
11     if running=='failed' or running=='inactive':
12         msg=msg+"Service State= %s\n"%(running)
13         stdin, stdout, stderr = ssh.exec_command('sudo systemctl restart '+service)
14         running=stdout.readlines()
15         if running==[]:
16             pass
17         else:
18             stdin, stdout, stderr = ssh.exec_command('sudo systemctl status '+service)
19             for line in iter(stdout.readline, ""):
20                 check.append(line)
21             if 'Address already in use' in str(check):
22                 stdin, stdout, stderr = ssh.exec_command('sudo lsof -i:'+incident['port'])
23                 address_check=stdout.readlines()
24                 address_check=address_check[1].split()
25                 msg=msg+"Service Error= %s\n"%(address_check[0])
26                 msg=msg+"Using Command= %s\n"%(address_check[1])
27                 msg=msg+"Using PID= %s\n"%(address_check[2])
28                 msg=msg+"Using By= %s\n"%(address_check[3])
29                 detail={}
30                 detail['reason']='Address already in use'
31                 detail['using_command']=address_check[0]
32                 detail['using_pid']=address_check[1]
33                 detail['using_by']=address_check[2]
34                 incident['detail']=detail
35                 stdin, stdout, stderr = ssh.exec_command("sudo kill -9 "+address_check[1])
36                 stdin, stdout, stderr = ssh.exec_command('sudo systemctl restart '+service)
37             else:
38                 pass
39             time.sleep(1)
40             stdin, stdout, stderr = ssh.exec_command('sudo systemctl show '+service+' --property=ActiveState')
41             finalcheck=stdout.readlines()
42             finalcheck=finalcheck[0].splitlines()
43             finalcheck=ansi_escape.sub('', finalcheck[1]).split('=')
44             finalcheck=finalcheck[1]
45             msg=msg+"---Recovery Result ---\n"
46             if finalcheck=='failed' or finalcheck=='inactive':
47                 msg=msg+"Service State= %s\n"%(finalcheck)
48                 msg=msg+"Error Msg= %s\n"%(str(check))
49             elif finalcheck=='active':
50                 msg=msg+"Service State= %s\n"%(finalcheck)
51                 msg=msg+"Solve Time= %s\n"%(datetime.fromtimestamp(time.time()).strftime('%d-%m-%Y %H:%M:%S'))
52             else:
53                 msg=msg+"Service State= %s\n"%(finalcheck)
54
55     else:
56         msg=msg+"Service State= %s\n"%(running)
57         notify(msg)
58         ssh.close()

```

รูปที่ 3.37 ชุดคำสั่งฟังก์ชัน fix_service

คำอธิบายรูปที่ 3.37

- บรรทัดที่ 2 เป็นการเชื่อมต่อเครื่องแม่ข่ายเสมือนที่พบปัญหาผ่านโปรโตคอล SSH
- บรรทัดที่ 5-9 เป็นการตรวจสอบสถานะภาพของ Service ว่าทำงานอยู่หรือไม่
- บรรทัดที่ 10-13 กรณี Service ไม่ทำงาน จะดำเนินการใช้คำสั่ง Restart Service ให้กลับมาทำงานอีกครั้ง
- บรรทัดที่ 14-15 ถ้า Restart แล้วไม่พบ Error จะดำเนินการในขั้นตอนถัดไป
- บรรทัดที่ 16-37 กรณีค่าผลลัพธ์มี Error “address already in use” จะดำเนินการตรวจสอบพอร์ตที่ Service ทำงาน แล้วตรวจสอบ และหยุดการทำงานของ Processที่กำลังใช้งานพอร์ตนั้นอยู่ หลังจากนั้น Restart Service
- บรรทัดที่ 38-43 เป็นการตรวจสอบสถานะภาพของ Service ว่ากลับมาทำงานปกติหรือไม่
- บรรทัดที่ 44-55 เป็นการนำข้อมูลปัญหาที่รวบรวมไว้ และผลลัพธ์การดำเนินการส่งไปยังผู้ดูแลระบบทาง LINE
- บรรทัดที่ 56 เป็นการปิดการเชื่อมต่อ SSH กับเครื่องแม่ข่ายเสมือน



รูปที่ 3.38 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (check_firewall)

คำอธิบายรูปที่ 3.38

- เรียกใช้งานฟังก์ชัน check_firewall
- ตรวจสอบ Firewall Rule ของ Port 80 และ 443 โดยการใช้โปรโตคอล SSH
- แจ้งผลลัพธ์การตรวจสอบไปยังผู้ดูแลระบบ ผ่าน LINE Notify

```

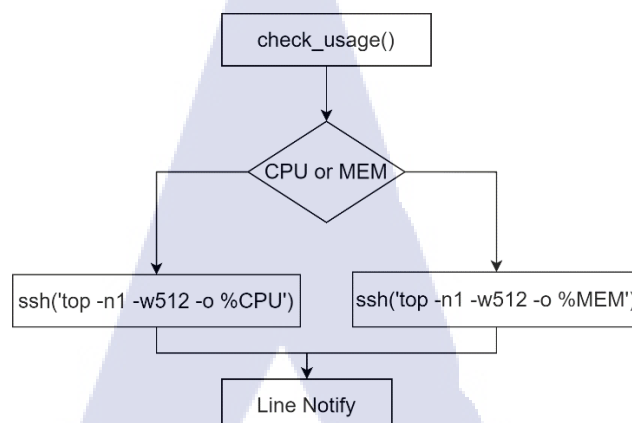
1 def check_firewall(incident):
2     ssh.connect(incident['ip'], port=22, username='username', password='password')
3     msg="IP= %s\n"%(192.168.xxx.xxx)+"---Firewall---\n"
4     check=[]
5     detail=[]
6     stdin, stdout, stderr = ssh.exec_command('sudo ufw status|grep '+incident['port'])
7     for line in iter(stdout.readline, ""):
8         check.append(ansi_escape.sub('', line))
9     if check==[]:
10        detail.append({'Port': incident['port'], 'Permit': 'Not Found', 'From': 'Not Found' })
11        msg=msg+"Port= %s\n"%(incident['port'])
12        msg=msg+" -Permit= %s\n"%(('Not Found'))
13        msg=msg+" -From= %s\n"%(('Not Found'))
14    else:
15        for line in check:
16            line=line.split()
17            if line[0]==incident['port'] or re.match(r"%sW[a-z]*" %(port), line[0]):
18                detail.append({'Port': line[0], 'Permit': line[1], 'From': line[2] })
19                msg=msg+"Port= %s\n"%(line[0])
20                msg=msg+" -Permit= %s\n"%(line[1])
21                msg=msg+" -From= %s\n"%(line[2])
22            else:
23                pass
24        if detail==[]:
25            detail.append({'Port': incident['port'], 'Permit': 'Not Found', 'From': 'Not Found' })
26            msg=msg+"Port= %s\n"%(incident['port'])
27            msg=msg+" -Permit= %s\n"%(('Not Found'))
28            msg=msg+" -From= %s\n"%(('Not Found'))
29        else:
30            pass
31    incident['detail']=detail
32    notify(msg)
33    ssh.close()

```

รูปที่ 3.39 ชุดคำสั่งฟังก์ชัน check_firewall

คำอธิบายรูปที่ 3.39

- บรรทัดที่ 2 เป็นการเชื่อมต่อเครื่องแม่ข่ายเสมือนที่พบปัญหาผ่านโปรโตคอล SSH
- บรรทัดที่ 6 เป็นการใช้คำสั่งตรวจสอบ Firewall Rule ที่เกี่ยวข้องกับพอร์ตที่ต้องการตรวจสอบ
- บรรทัดที่ 7-30 เป็นการตรวจสอบ ว่ามี Firewall Rule ที่เกี่ยวข้องหรือไม่ แล้วรวบรวมข้อมูลส่งไปยังผู้ดูแลระบบผ่านทาง LINE
- บรรทัดที่ 33 เป็นการปิดการเชื่อมต่อ SSH กับเครื่องแม่ข่ายเสมือน



รูปที่ 3.40 ขั้นตอนการทำงานของชุดคำสั่งฟังก์ชัน Incident Correction Procedure Control (check_usage)

คำอธิบายรูปที่ 3.40

- เรียกใช้งานฟังก์ชัน `check_usage`
- ตรวจสอบปริมาณการใช้งานซีพียู หรือหน่วยความจำหลัก โดยใช้โปรโตคอล SSH
- แจ้งผลลัพธ์การตรวจสอบไปยังผู้ดูแลระบบ ผ่าน LINE Notify

```

1 def check_usage(incident, component):
2     ssh.connect(incident['ip'], port=22, username='username', password='password')
3     msg="IP= %s\n"%(192.168.xxx.xxx)+"----%s Usage----\n"%(component)
4     check=[]
5     detail=[]
6     stdin, stdout, stderr = ssh.exec_command('top -n1 -w512 -o %'+component, get_pty=True)
7     for line in iter(stdout.readline, ""):
8         check.append(ansi_escape.sub('', line))
9     for line in check[7:12]:
10        line=line.split()
11        detail.append({'user': line[1], component: line[8], 'command': line[11] })
12        msg=msg+"Command: %s\n"%(line[11])
13        msg=msg+"  "+component+": "+line[8]+" %\n"
14        msg=msg+"  -User: %s\n"%(line[1])
15    incident['detail']=detail
16    notify(msg)
17    ssh.close()
  
```

รูปที่ 3.41 ชุดคำสั่งฟังก์ชัน `check_usage`

คำอธิบายรูปที่ 3.41

- บรรทัดที่ 2 เป็นการเชื่อมต่อเครื่องแม่ข่ายเสมือนที่พบปัญหาผ่านโปรโตคอล SSH

- บรรทัดที่ 6 เป็นการใช้คำสั่งตรวจสอบ Running Process ที่ใช้งานซีพียู หรือ หน่วยความจำหลักทั้งหมดในขณะนั้น
- บรรทัดที่ 7-16 เป็นการเลือกเฉพาะข้อมูลของ Running Process ที่ใช้ทรัพยากร สูงสุดอันดับที่ 1-5 ส่งไปยังผู้ดูแลระบบผ่านทาง LINE
- บรรทัดที่ 17 เป็นการปิดการเชื่อมต่อ SSH กับเครื่องแม่ข่ายเสมือน

3.4 การออกแบบการทดสอบ

เพื่อให้เห็นถึงประสิทธิภาพของระบบ ผู้ศึกษาได้เตรียมอุปกรณ์สำหรับการทดสอบดังนี้

1. เครื่องแม่ข่ายเสมือนที่ต้องการตรวจสอบสถานะภาพ (Monitoring Client) ที่อยู่ภายในระบบการประมวลผลแบบกลุ่มเมฆจำนวน 1 เครื่อง โดยมีคุณสมบัติดังนี้

- ซีพียู Intel(R) Xeon(R) CPU E7- 2860 @ 2.27GHz จำนวน 4 Core
- หน่วยความจำหลักขนาด 8 กิกะไบต์
- หน่วยความจำสำรองขนาด 41 กิกะไบต์
- ระบบปฏิบัติการ Ubuntu 16.04
- ติดตั้ง Application ได้แก่ SNMPD, Nginx, UFW และ Docker
- มี Docker Container ทำงานจำนวน 2 Container คือ Web Application และ Web Database
- ติดตั้ง Telegraf Agent และ Zabbix Agent

2. เครื่องแม่ข่ายเสมือนที่อยู่ภายในระบบการประมวลผลแบบกลุ่มเมฆ ที่ทำหน้าที่ตรวจสอบสถานะภาพ (Monitoring Server) จำนวน 1 เครื่อง โดยมีคุณสมบัติดังนี้

- ซีพียู Intel(R) Xeon(R) CPU E7- 2860 @ 2.27GHz จำนวน 16 Core
- หน่วยความจำหลักขนาด 16 กิกะไบต์
- หน่วยความจำสำรองจำนวน 50 กิกะไบต์
- ระบบปฏิบัติการ Ubuntu 18.04
- ติดตั้ง Application ได้แก่ SNMPD, Docker และ Python 3.6.9
- ติดตั้งชุดคำสั่งไพทอนส่วน Data Collection และ Diagnostic and Recovery ที่พัฒนาขึ้น
- ติดตั้งฐานข้อมูลอนุกรมแบบ container ได้แก่ Prometheus, InfluxDB และ Graphite

- ติดตั้งเครื่องมือ Monitoring แบบ container ได้แก่ Grafana, Chronograf และ Zabbix

ผู้ศึกษาได้ออกแบบการทดสอบระบบออกเป็น 2 ส่วน โดยมีรายละเอียดดังนี้

3.4.1 ส่วน Monitoring

ในการทดสอบประสิทธิภาพ ของส่วนติดตามสถานะภาพ ผู้ศึกษาแบ่งการทดลองออกเป็น 4 ด้านได้แก่

1. ด้านความถูกต้องแม่นยำของค่าข้อมูลสถานะภาพ เพื่อตรวจสอบค่าสถานะภาพซีพียู , หน่วยความจำหลัก, หน่วยความจำสำรอง และปริมาณการรับ-ส่งข้อมูลจราจรเครือข่าย ของเครื่อง Monitoring Client ว่ามีความถูกต้องแม่นยำ โดยเปรียบเทียบกับเครื่องมือติดตามสถานะภาพอื่นๆ ได้แก่ Zabbix Agent และ Telegraf Agent

2. ด้านปริมาณการใช้ทรัพยากรเครื่องแม่ข่ายเสมือน Monitoring Server ของระบบที่ออกแบบ เพื่อวัดเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลัก โดยแบ่งเป็น 2 ส่วนคือ ส่วนฐานข้อมูลอนุกรมเวลา Prometheus และส่วนเครื่องมือติดตามสถานะภาพ Grafana

2.1 ส่วนฐานข้อมูลอนุกรมเวลา Prometheus มีวิธีทดสอบโดยการ ส่งค่าข้อมูลไปเก็บยังฐานข้อมูลจำนวน 1,000 Metric ในทุกๆ 10 วินาที แล้ววัดวัดเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลัก โดยเปรียบเทียบกับ ฐานข้อมูลอนุกรมเวลาอื่นๆได้แก่ InfluxDB และ Graphite

2.2 ส่วนเครื่องมือติดตามสถานะภาพ Grafana มีวิธีการทดสอบโดยการสร้างกราฟข้อมูลจำนวน 400 กราฟหลังจากนั้นวัดเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลัก โดยเปรียบเทียบกับ เครื่องมืออื่นๆ ได้แก่ Chronograf และ Zabbix

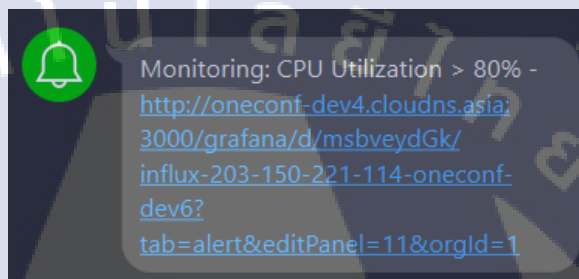
3. ด้านประสิทธิภาพการแจ้งเตือนความผิดปกติ เพื่อทดสอบความแม่นยำในการติดตามความผิดปกติของค่าสถานะภาพต่างๆ โดยการแจ้งเตือนทาง LINE ผู้ศึกษาจะทำการสร้างสถานการณ์จำลองให้สถานะภาพของเครื่องแม่ข่ายเสมือนมีความผิดปกติ ดังนี้

1. สถานการณ์ที่ปริมาณการใช้งานซีพียู มากกว่า 80 เปอร์เซ็นต์ โดยการใช้เครื่องมือ stress ด้วยวิธีให้ซีพียูกำหนดค่า สแควรูท (Square Root)
2. สถานการณ์ที่ปริมาณการใช้งานหน่วยความจำหลัก มากกว่า 80 เปอร์เซ็นต์ โดยการใช้เครื่องมือ stress ด้วยวิธีเพิ่มข้อมูลจำลองในหน่วยความจำหลัก
3. สถานการณ์ที่ปริมาณการใช้งานหน่วยความจำสำรอง มากกว่า 80 เปอร์เซ็นต์ โดยการเพิ่มไฟล์ข้อมูลขนาดใหญ่เข้าไปในหน่วยความจำสำรอง
4. สถานการณ์ที่ไม่มีการตอบสนองต่อการ Ping โดยการปิดเครื่องแม่ข่ายเสมือน

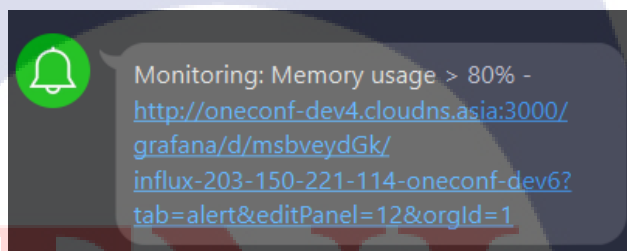
5. สถานการณ์ที่ TCP พอร์ต 80 หรือ 443 ไม่ตอบสนอง โดยการหยุดการทำงานแอปพลิเคชันที่ทำงานอยู่บนพอร์ต 80 หรือ 443

6. สถานการณ์ที่รหัสสถานภาพของ HTTP ไม่เท่ากับ 200 โดยการหยุดการทำงานของเว็บแอปพลิเคชันที่อยู่ในเครื่องแม่ข่ายเสมือน

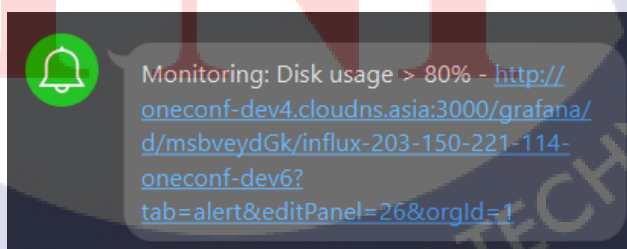
โดยในแต่ละสถานการณ์จะถูกจำลองขึ้นจำนวน 10 ครั้ง ซึ่งผลลัพธ์ที่คาดหวังคือระบบมีการแจ้งเตือนไปยังผู้ดูแลระบบผ่าน แอปพลิเคชัน LINE ทุกครั้งเมื่อสถานภาพของเครื่องแม่ข่ายเสมือนมีความผิดปกติ ดังรูปที่ 3.42 ถึงรูปที่ 3.47



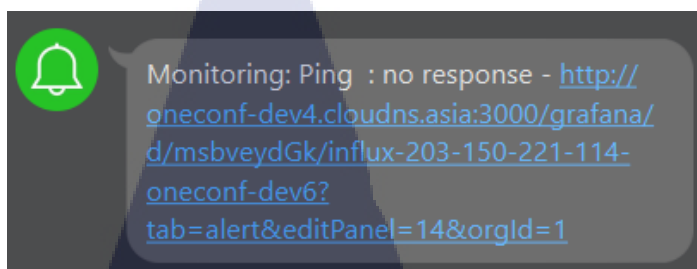
รูปที่ 3.42 การแจ้งเตือนปริมาณการใช้งานซีพียู มากกว่า 80 เปอร์เซ็นต์



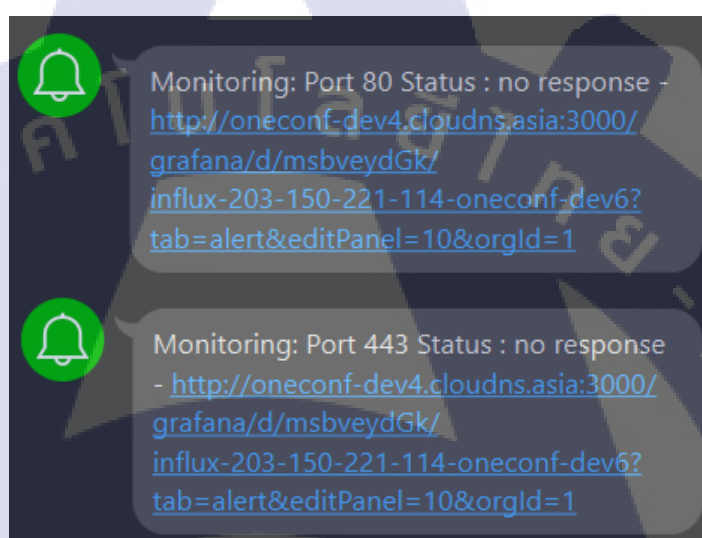
รูปที่ 3.43 การแจ้งเตือนปริมาณการใช้งานหน่วยความจำหลัก มากกว่า 80 เปอร์เซ็นต์



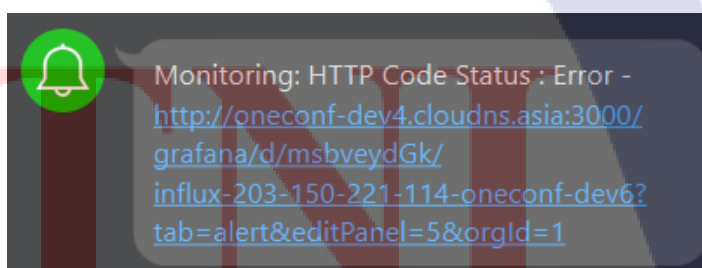
รูปที่ 3.44 การแจ้งเตือนปริมาณการใช้งานหน่วยความจำสำรอง มากกว่า 80 เปอร์เซ็นต์



รูปที่ 3.45 การแจ้งเตือนไม่มีการตอบสนองต่อการ Ping



รูปที่ 3.46 การแจ้งเตือน TCP พอร์ต 80 หรือ 443 ไม่ตอบสนอง

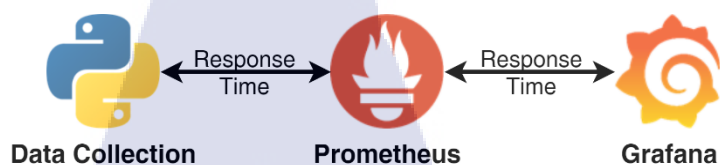


รูปที่ 3.47 การแจ้งเตือนรหัสสถานภาพของ HTTP ไม่เท่ากับ 200

4. ด้านเวลาในการตอบสนอง (Response Time)

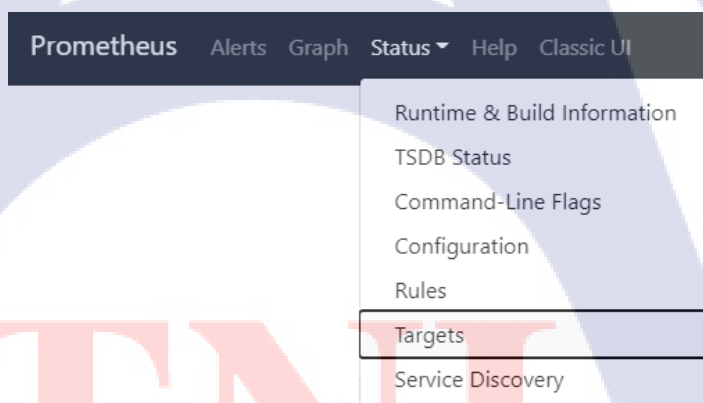
เนื่องจากระบบติดตามสถานภาพมีส่วนประกอบหลายส่วนทำงานร่วมกัน จึงต้องตรวจสอบ Response Time ที่ใช้ในการติดต่อสื่อสารซึ่งทำให้ทราบค่าเวลาของค่าสถานภาพที่ได้รับ

จากส่วน Data Collection จนไปถึงค่าที่แสดงในส่วน Monitoring ดังรูปที่ 3.48 ว่ามีเวลาคลาดเคลื่อนมาน้อยเพียงใด โดยแบ่งการตรวจสอบออกเป็น 2 ส่วนดังนี้



รูปที่ 3.48 การติดต่อสื่อสารของส่วนประกอบระบบติดตามสถานภาพ

1. เวลาในการตอบสนองของการสื่อสารข้อมูล ระหว่างส่วนฐานข้อมูลอนุกรมเวลา Prometheus กับ ส่วน Data Collection โดยสามารถตรวจสอบค่าเวลาที่ใช้ Prometheus ดึงข้อมูลจากส่วน Data Collection ได้จากหน้าเว็บ Prometheus ที่ได้ทำการติดตั้งไว้ โดยไปที่เมนู Status แล้วเลือก Target ดังรูปที่ 3.49 แล้วดูในส่วน Scrape Duration ดังรูปที่ 3.50 แล้วสังเกตค่าเป็นระยะเวลา 1 นาที



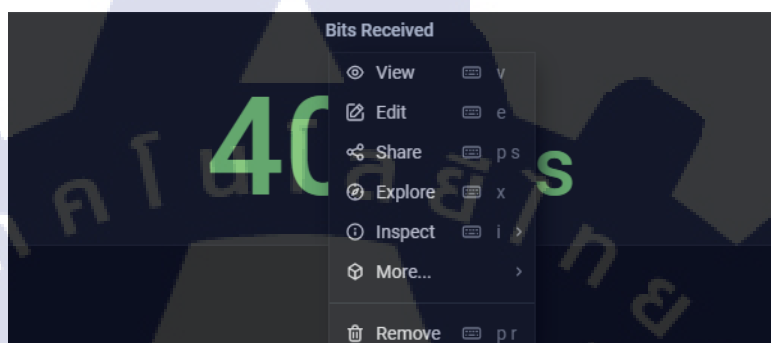
รูปที่ 3.49 Prometheus ส่วน Targets

prometheus (1/1 up) [show less](#)

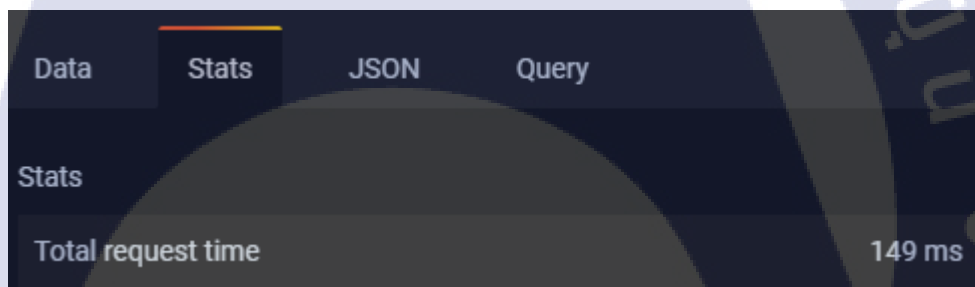
Endpoint	State	Labels	Last Scrape	Scrape Duration
http://metrics	:8001/ UP	job="prometheus"	919.000ms	206.620ms

รูปที่ 3.50 Prometheus ส่วน Scrape Duration

2. เวลาในการตอบสนองของการสื่อสารข้อมูล ระหว่างส่วนฐานข้อมูลอนุกรมเวลา Prometheus กับ ส่วนเครื่องมือติดตามสถานภาพ Grafana โดยสามารถตรวจสอบค่าเวลาที่ Grafana ดึงข้อมูลจากส่วน Prometheus ได้จากกราฟข้อมูล โดยคลิกที่กราฟ เลือก Inspect ดังรูปที่ 3.51 แล้วไปยังส่วน Stats แล้วดูในส่วน Total request time ดังรูปที่ 3.52 แล้วสังเกตค่าเป็นระยะเวลา 1 นาที



รูปที่ 3.51 Grafana ส่วน Inspect



รูปที่ 3.52 Prometheus ส่วน Total request time

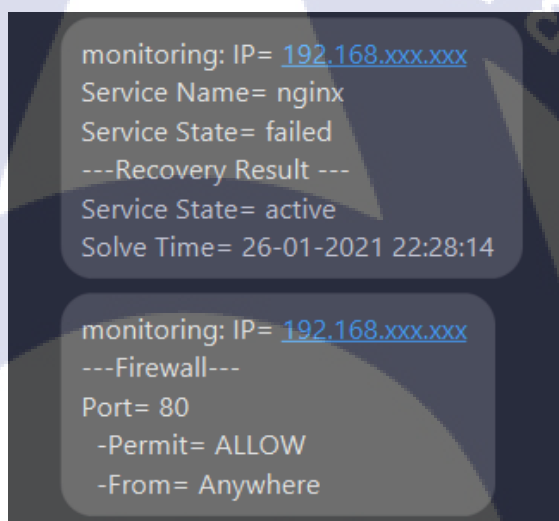
3.4.2 ส่วน Diagnostic and Recovery

ในการทดสอบประสิทธิภาพ ของส่วน Diagnostic and Recovery ผู้ศึกษาจะทำการทดสอบความสามารถในการตรวจสอบ และกู้คืนเว็บแอปพลิเคชันจากปัญหาในตารางที่ 3.5 โดยทำการสร้างสถานการณ์จำลองให้ส่วนประกอบของระบบเว็บแอปพลิเคชัน ที่อยู่ภายในเครื่อง Monitoring Client มีความผิดปกติ เพื่อทดสอบประสิทธิภาพการดำเนินการตามตารางที่ 3.6 ดังนี้

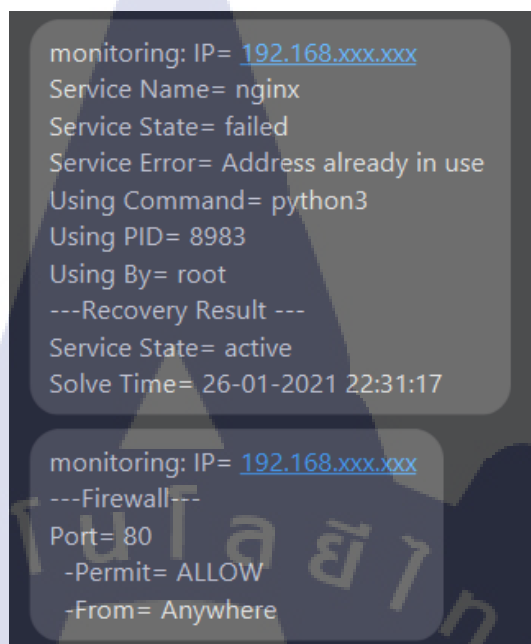
1. Nginx ไม่ทำงาน
2. Nginx ไม่ทำงานเนื่องจาก Port ถูกใช้งานอยู่

3. Container Web Application ไม่ทำงาน
4. Container Web Application ไม่ทำงานเนื่องจาก Port ถูกใช้งานอยู่
5. Container Web Database ไม่ทำงาน
6. Container Web Database ไม่ทำงานเนื่องจาก Port ถูกใช้งานอยู่
7. Docker ไม่ทำงาน
8. ปริมาณการใช้ซีพียูมากกว่า 80 เปอร์เซ็นต์
9. ปริมาณการใช้หน่วยความจำหลักมากกว่า 80 เปอร์เซ็นต์

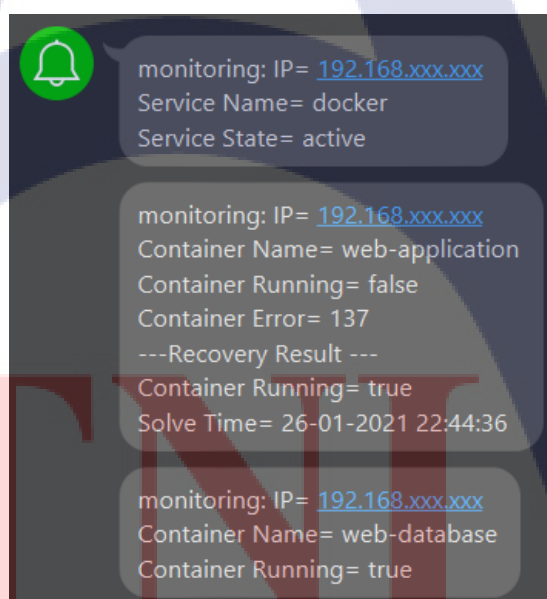
โดยในแต่ละสถานการณ์จะถูกจำลองขึ้นจำนวน 10 ครั้ง ซึ่งผลลัพธ์ที่คาดหวังคือสามารถกู้คืนส่วนประกอบของระบบ Web Application ได้ และส่งผลลัพธ์ระบบมีการแจ้งเตือนไปยังผู้ดูแลระบบผ่าน แอปพลิเคชัน LINE ดังรูปที่ 3.53 ถึงรูปที่ 3.61



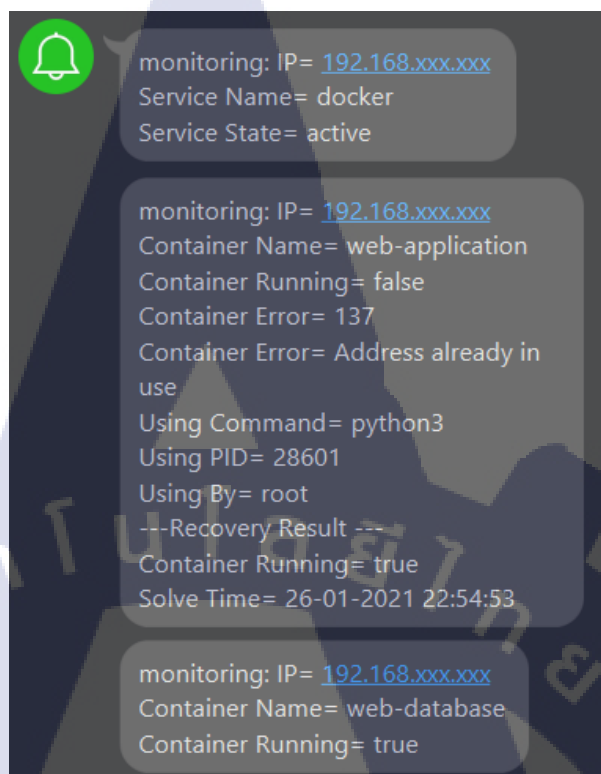
รูปที่ 3.53 การแจ้งเตือนเมื่อ Nginx ไม่ทำงาน



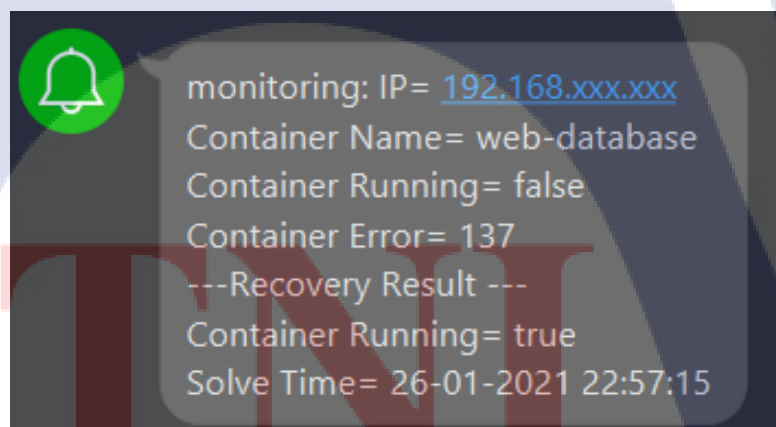
รูปที่ 3.54 การแจ้งเตือนเมื่อ Nginx ไม่ทำงานเนื่องจาก Port ถูกใช้งานอยู่



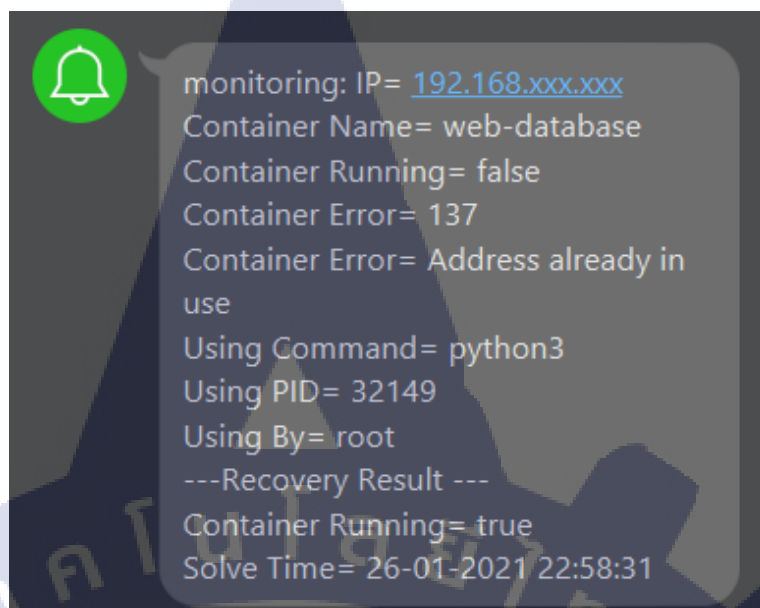
รูปที่ 3.55 การแจ้งเตือนเมื่อ Container Web Application ไม่ทำงาน



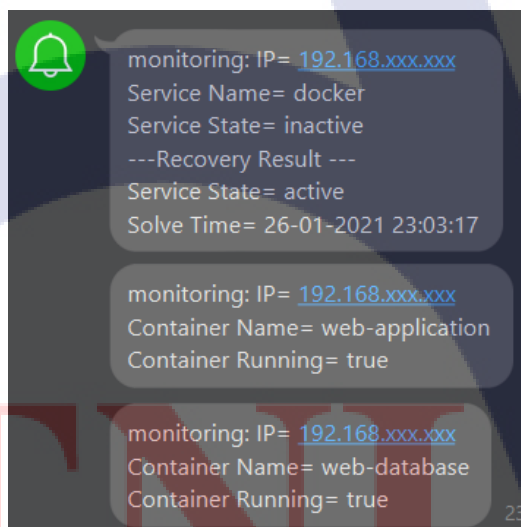
รูปที่ 3.56 การแจ้งเตือนเมื่อ Container Web Application ไม่ทำงาน เนื่องจาก Port ถูกใช้งานอยู่



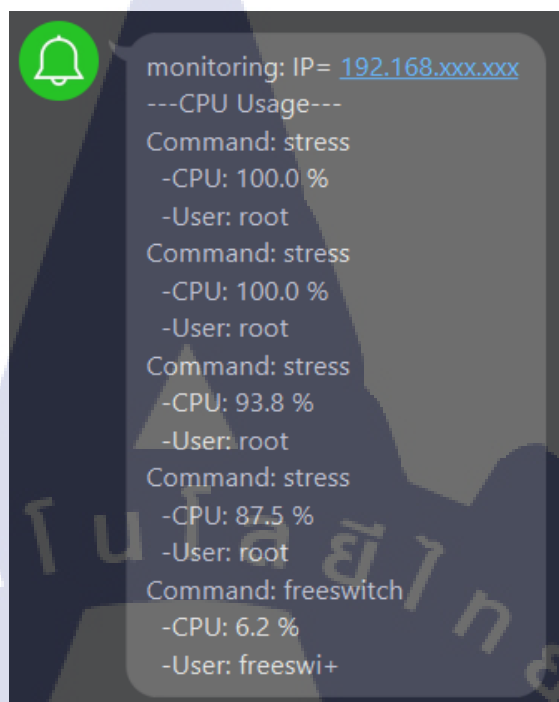
รูปที่ 3.57 การแจ้งเตือนเมื่อ Container Web Database ไม่ทำงาน



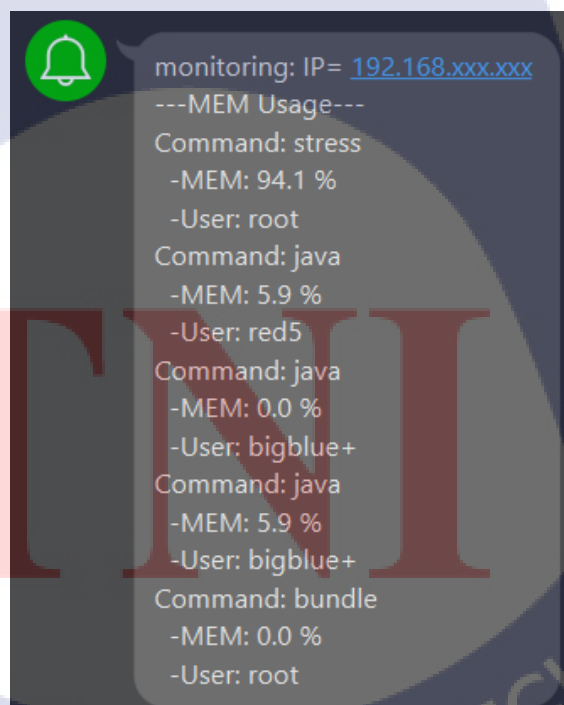
รูปที่ 3.58 การแจ้งเตือนเมื่อ Container Web Database ไม่ทำงาน เนื่องจาก Port ถูกใช้งานอยู่



รูปที่ 3.59 การแจ้งเตือนเมื่อ Docker ไม่ทำงาน



รูปที่ 3.60 การแจ้งเตือนเมื่อปริมาณการใช้ซีพียูมากกว่า 80 เปอร์เซ็นต์



รูปที่ 3.61 การแจ้งเตือนเมื่อปริมาณการใช้หน่วยความจำหลักมากกว่า 80 เปอร์เซ็นต์

บทที่ 4

ผลการทดลอง

จากการศึกษาทฤษฎี เทคโนโลยี และงานวิจัยที่เกี่ยวข้องเพื่อเป็นองค์ความรู้ในการศึกษาครั้งนี้ ผู้ศึกษาได้ทำการทดสอบเพื่อวัดประสิทธิภาพของระบบ โดยการนำผลลัพธ์ของการทดสอบมาทำการสรุปและวิเคราะห์เพิ่มเติม ดังนี้

- 4.1 ความถูกต้องแม่นยำของค่าข้อมูลสถานะภาพ
- 4.2 ปริมาณการใช้ทรัพยากร
- 4.3 การแจ้งเตือนความผิดปกติ
- 4.4 เวลาในการตอบสนองของส่วนติดตามสถานะภาพ
- 4.5 การตรวจสอบปัญหา และกู้คืนระบบ

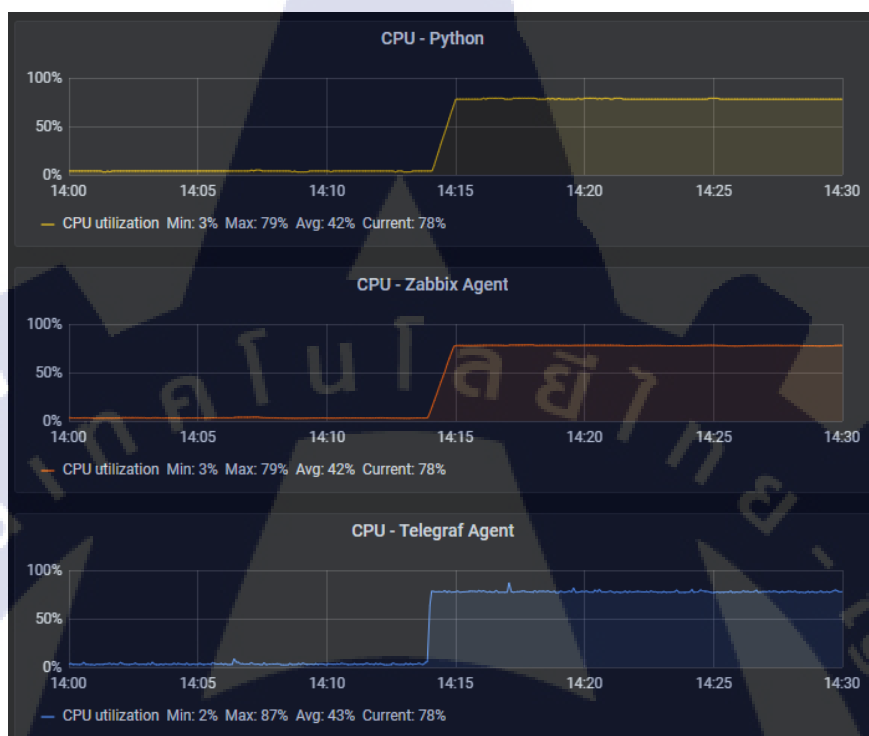
4.1 ความถูกต้องแม่นยำของค่าข้อมูลสถานะภาพ

จากการทดสอบด้านความถูกต้องแม่นยำของค่าข้อมูลสถานะภาพซีฟิยู, หน่วยความจำหลัก, หน่วยความจำสำรอง และปริมาณการรับ-ส่งข้อมูลจราจรเครือข่าย ของเครื่อง Monitoring Client โดยเปรียบเทียบระหว่างชุดคำสั่งไพทอนที่พัฒนาโดยใช้โปรโตคอล SNMP กับเครื่องมืออื่นๆที่เป็นแบบ Agent ติดตั้งในเครื่อง Monitoring Client ที่ให้ความแม่นยำสูงได้แก่ Zabbix Agent และ Telegraf Agent ซึ่งการทดสอบนำข้อมูลสถานะภาพของเครื่อง Monitoring Client จากเครื่องมือทั้ง 3 ตัวในช่วงเวลา 30 นาทีมาเปรียบเทียบกัน ดังรูปที่ 4.1 ถึง รูปที่ 4.5

4.1.1 ผลการทดสอบเปรียบเทียบค่าซีฟิยู

จากรูปที่ 4.1 ผลการทดสอบเปรียบเทียบค่าซีฟิยูของทั้ง 3 เครื่องมือพบว่า เนื่องจากแต่ละเครื่องมือมีเวลาที่ไปตรวจสอบค่าข้อมูลไม่พร้อมกันซึ่งจะทำให้ค่าข้อมูลล่าสุดไม่เท่ากัน ดังนั้นจึงต้องประเมินจากค่าเฉลี่ยของค่าข้อมูล ซึ่งค่าเฉลี่ยเปอร์เซ็นต์การใช้ซีฟิยู อยู่ระหว่าง 42-43 เปอร์เซ็นต์ ซึ่งจากรูปกราฟข้อมูลที่ได้จาก Telegraf Agent ในช่วงเวลา 14:14 เปอร์เซ็นต์การใช้งานซีฟิยูมีการเปลี่ยนแปลงเพิ่มขึ้นไปที่ 78 เปอร์เซ็นต์ สังเกตได้ว่าค่าเปอร์เซ็นต์ซีฟิยูที่ได้จากชุดคำสั่งไพทอนที่พัฒนาขึ้นจะมีการเปลี่ยนแปลงแบบไต่ระดับ ต่างกับค่าของ Telegraf Agent ที่ค่าข้อมูลจะเปลี่ยนแปลงทันที เนื่องจากค่าเปอร์เซ็นต์ซีฟิยูที่ได้จาก SNMP เป็นค่าเฉลี่ยเปอร์เซ็นต์ของซีฟิยูใน 1 นาที ซึ่งหมายความว่าหากเปอร์เซ็นต์ของซีฟิยู มีการเปลี่ยนแปลงจะต้องใช้เวลาประมาณ 1 นาทีเพื่อทราบค่าจริง ซึ่งไม่กระทบในการนำไปใช้งานจริง เนื่องจากในทางปฏิบัติการตีความหมายว่าซีฟิยูมีความ

ผิดปกติ นั้น ก็ต่อเมื่อค่าเปอร์เซ็นต์ของซีพียูสูงเกินกำหนดต่อเนื่องกันในระยะเวลาหนึ่ง เช่น ค่าเปอร์เซ็นต์ของซีพียูสูงเกินกำหนดต่อเนื่องกัน 10 นาที



รูปที่ 4.1 การเปรียบเทียบค่าเปอร์เซ็นต์ซีพียู

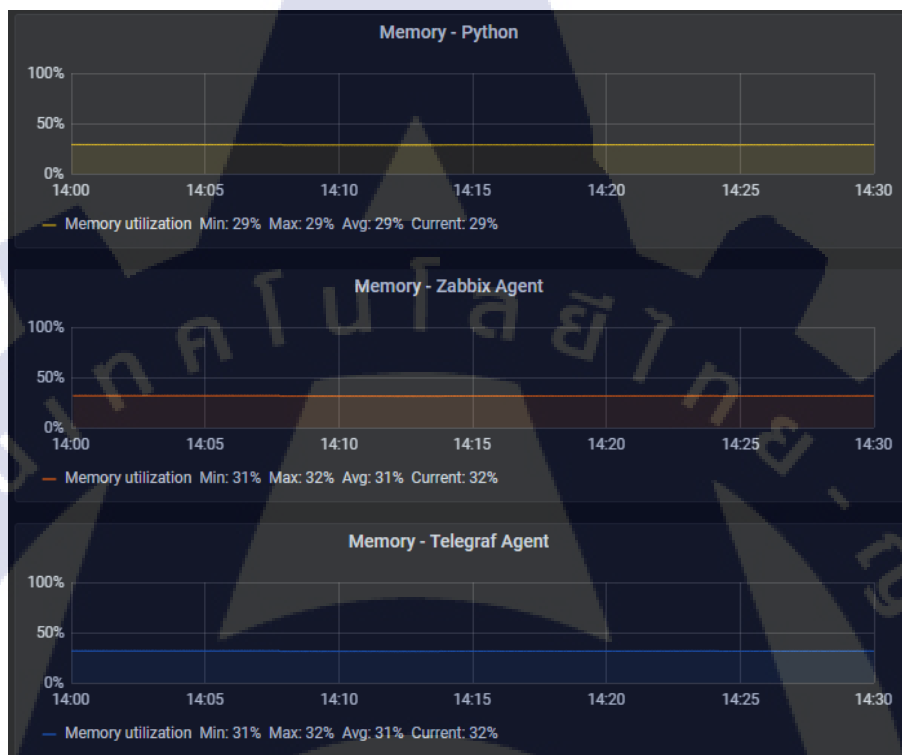
ตารางที่ 4.1 ตารางเปรียบเทียบค่าเปอร์เซ็นต์ซีพียู

เครื่องมือ	ค่าปริมาณการใช้งานซีพียู (%)			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Python	78	3	79	78
Zabbix Agent	78	3	79	78
Telegraf	78	2	87	78

4.1.2 ผลการทดสอบเปรียบเทียบค่าหน่วยความจำหลัก

จากรูปที่ 4.2 ผลการทดสอบเปรียบเทียบค่าหน่วยความจำหลักของทั้ง 3 เครื่องมือพบว่า มีค่าข้อมูลไม่ตรงกัน โดยค่าที่ได้จากชุดคำสั่งไพทอนจะมีค่าน้อยกว่าเครื่องมืออื่น เนื่องจากโปรโตคอล SNMP ไม่มีข้อมูล OID ที่เรียกว่า SReclaimable คือข้อมูลพื้นที่ในหน่วยความจำหลักที่

สามารถเรียกคืนได้ ซึ่งจะต้องถูกนำไปคำนวณรวมกับ mem_used ในตารางที่ 3.2 ต่างจากเครื่องมือที่เป็น Agent ติดตั้งบนเครื่อง Monitoring Client ที่สามารถไปดึงค่าข้อมูลจาก meminfo ใน Ubuntu มาคำนวณเปอร์เซ็นต์หน่วยความจำหลักได้อย่างครบถ้วน



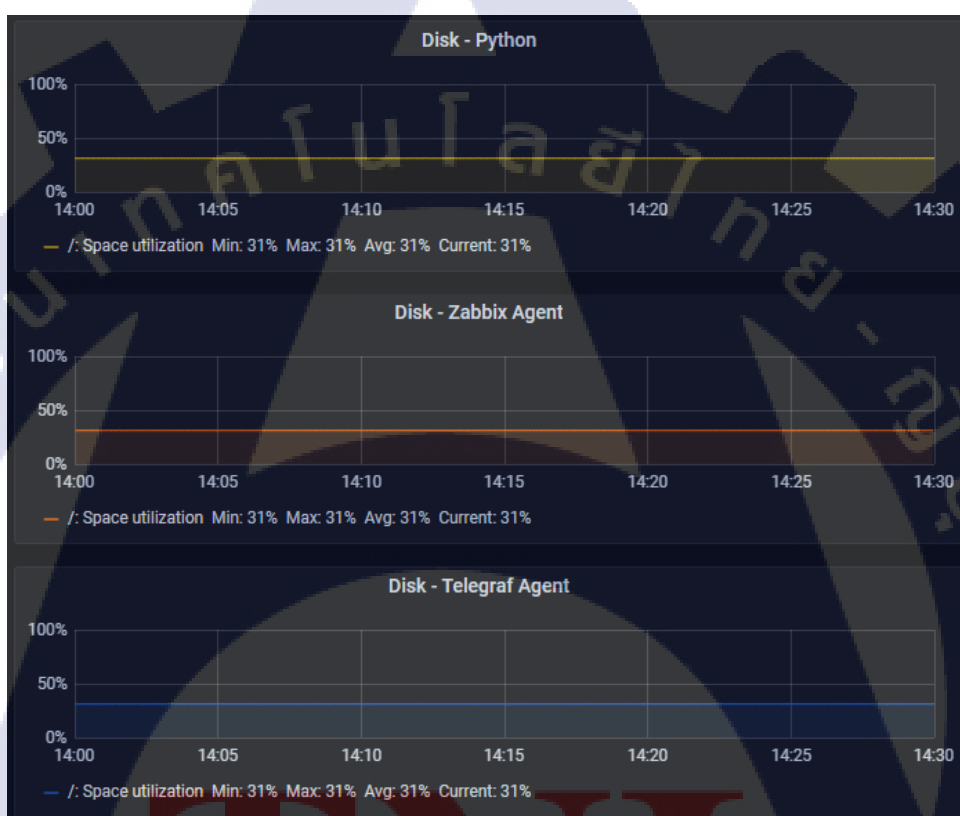
รูปที่ 4.2 การเปรียบเทียบค่าหน่วยความจำหลัก

ตารางที่ 4.2 ตารางเปรียบเทียบค่าหน่วยความจำหลัก

เครื่องมือ	ค่าปริมาณการใช้งานหน่วยความจำหลัก (%)			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Python	29	29	29	29
Zabbix Agent	32	31	32	31
Telegraf	32	31	32	31

4.1.3 ผลการทดสอบเปรียบเทียบค่าหน่วยความจำสำรอง

จากรูปที่ 4.3 ผลการทดสอบเปรียบเทียบค่าหน่วยความจำสำรองของทั้ง 3 เครื่องมือพบว่ามีค่าข้อมูลตรงกัน เนื่องในชุดคำสั่งไฟทอนดึงค่าปริมาณ Disk ที่ถูกใช้ และ ปริมาณ Disk ทั้งหมดมาคำนวณตามวิธีคำนวณในตารางที่ 3.2 จะได้ค่าเปอร์เซ็นต์ Disk ที่ถูกใช้ มีความถูกต้องแม่นยำกว่าการใช้ OID 1.3.6.1.4.1.2021.9.1.9 (diskPercent) ที่เป็น OID เปอร์เซ็น Disk ที่ถูกใช้ไปซึ่งมีค่าน้อยกว่าความเป็นจริง



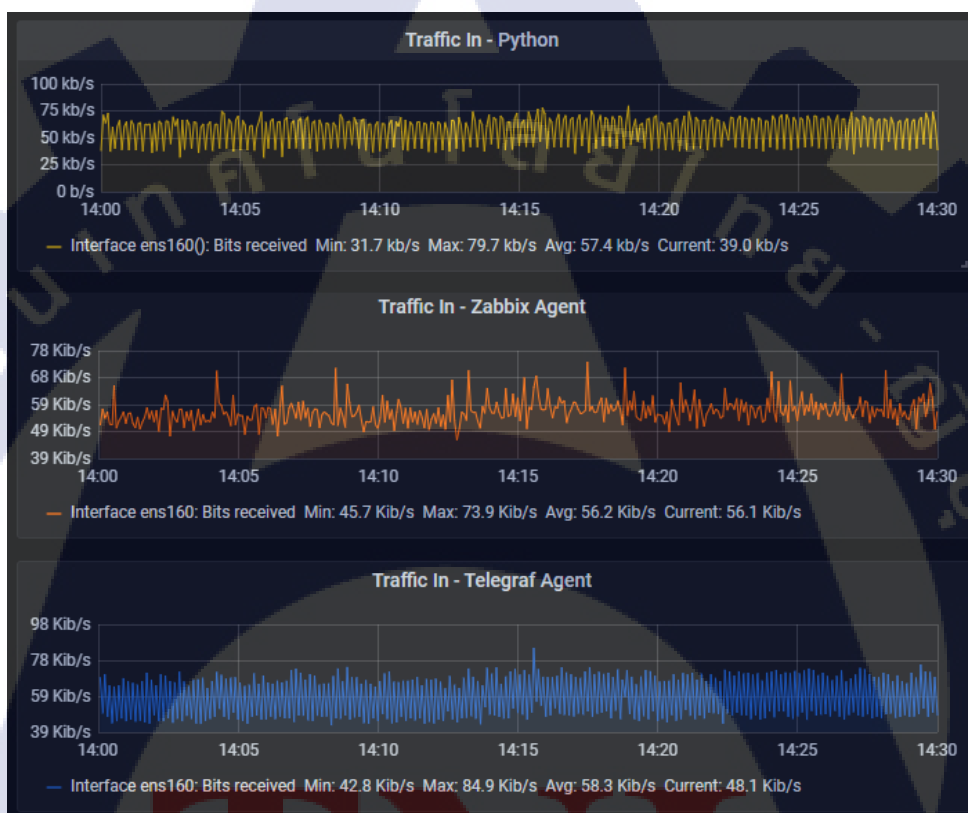
รูปที่ 4.3 การเปรียบเทียบค่าหน่วยความจำสำรอง

ตารางที่ 4.3 ตารางเปรียบเทียบค่าหน่วยความจำสำรอง

เครื่องมือ	ค่าปริมาณการใช้งานหน่วยความจำสำรอง (%)			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Python	31	31	31	31
Zabbix Agent	31	31	31	31
Telegraf	31	31	31	31

4.1.4 ผลการทดสอบเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่าย

จากรูปที่ 4.4 ผลการทดสอบเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่ายของทั้ง 3 เครื่องมือพบว่า เนื่องจากแต่ละเครื่องมือมีเวลาที่ใช้ตรวจสอบค่าข้อมูลไม่พร้อมกันซึ่งจะทำให้ค่าข้อมูลล่าสุดไม่เท่ากัน ดังนั้นจึงต้องประเมินจากค่าเฉลี่ยของค่าข้อมูลในช่วงระยะเวลาเดียวกัน ซึ่งค่าเฉลี่ยปริมาณการรับข้อมูลจราจรเครือข่ายอยู่ระหว่าง 56.2 ถึง 58.3 Kbps โดยค่าเฉลี่ยที่ได้จากชุดคำสั่งโฟตอนอยู่ที่ 57.4 Kbps



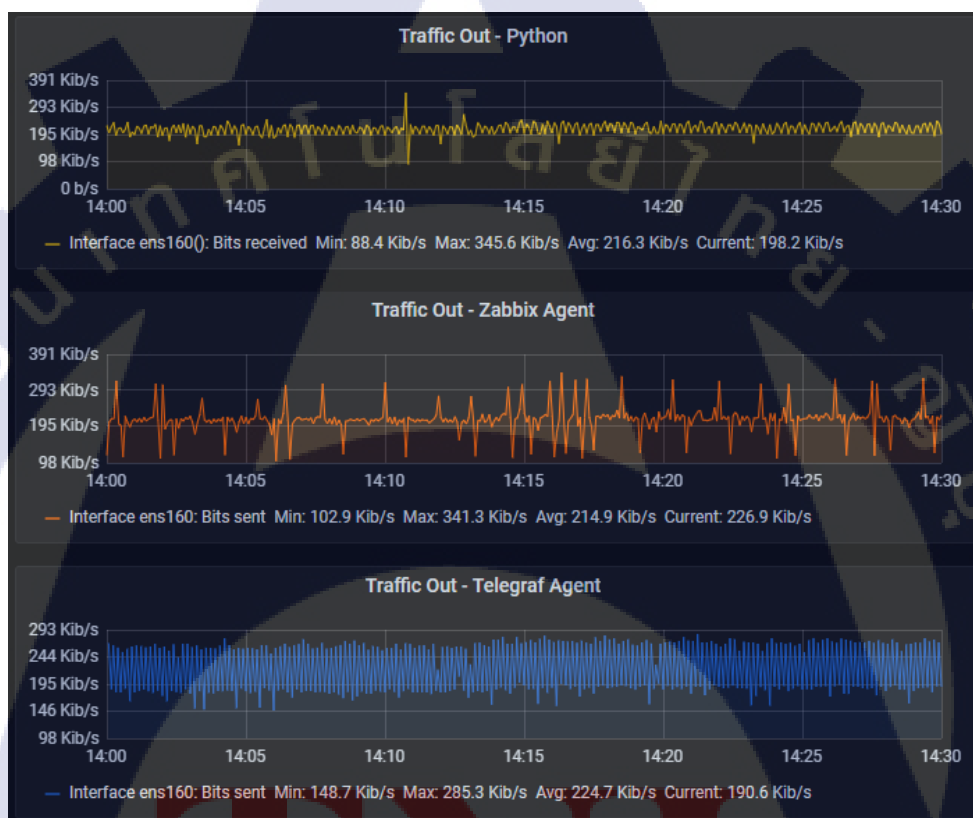
รูปที่ 4.4 การเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่าย

ตารางที่ 4.4 ตารางเปรียบเทียบค่าปริมาณการรับข้อมูลจราจรเครือข่าย

เครื่องมือ	ค่าปริมาณการรับข้อมูลจราจรเครือข่าย (Kib/s)			
	ค่าล่าสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Python	39	31.7	79.7	57.4
Zabbix Agent	56.1	45.7	73.9	56.2
Telegraf	48.1	42.8	84.9	58.3

4.1.5 ผลการทดสอบเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่าย

จากรูปที่ 4.5 ผลการทดสอบเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่ายของทั้ง 3 เครื่องมือพบว่า เนื่องจากแต่ละเครื่องมือมีเวลาที่ใช้ตรวจสอบค่าข้อมูลไม่พร้อมกันซึ่งจะทำให้ค่าข้อมูลล่าสุดไม่เท่ากัน ดังนั้นจึงต้องประเมินจากค่าเฉลี่ยของค่าข้อมูลในช่วงระยะเวลาเดียวกัน ซึ่งค่าเฉลี่ยปริมาณการส่งข้อมูลจราจรเครือข่ายอยู่ระหว่าง 214.9 ถึง 224.7 Kbps โดยค่าเฉลี่ยที่ได้จากชุดคำสั่งไพทอนอยู่ที่ 216.3 Kbps



รูปที่ 4.5 การเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่าย

ตารางที่ 4.5 ตารางเปรียบเทียบค่าปริมาณการส่งข้อมูลจราจรเครือข่าย

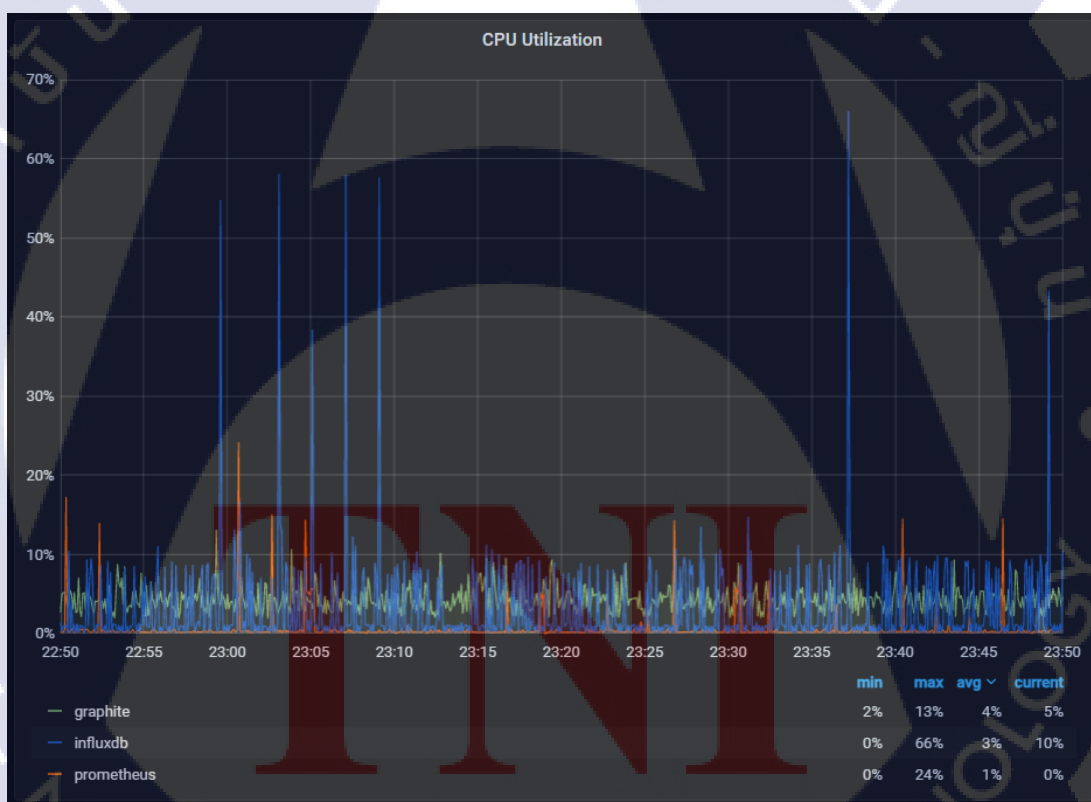
เครื่องมือ	ค่าปริมาณการส่งข้อมูลจราจรเครือข่าย (Kib/s)			
	ค่าล่าสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Python	198.2	88.4	345.6	216.3
Zabbix Agent	226.9	102.9	341.3	214.9
Telegraf	190.6	148.7	285.3	224.7

4.2 ปริมาณการใช้ทรัพยากร

จากการทดสอบด้านปริมาณการใช้ทรัพยากร ของระบบที่ออกแบบ เพื่อวัดเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลักของเครื่อง Monitoring Server จากฐานข้อมูลอนุกรมเวลา และเครื่องมือติดตามสถานะภาพในช่วงเวลา 60 นาทีมาเปรียบเทียบกัน ดังรูปที่ 4.6 ถึง รูปที่ 4.8

4.2.1 ผลการทดสอบเปรียบเทียบปริมาณการใช้ทรัพยากรของฐานข้อมูลอนุกรมเวลา

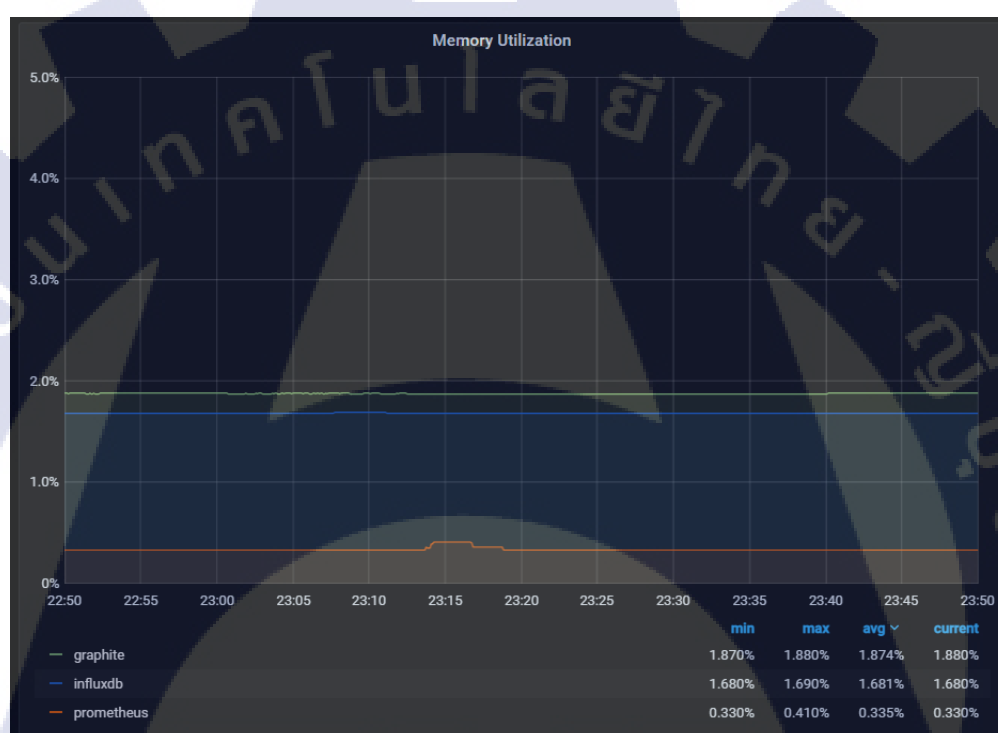
จากรูปที่ 4.6 และ 4.7 ผลการทดสอบวัดปริมาณ และเปรียบเทียบการใช้ทรัพยากรของฐานข้อมูลอนุกรมเวลาที่น่ามาใช้ในการออกแบบระบบ และฐานข้อมูลตัวอื่นๆ โดยการส่งค่าข้อมูลไปเก็บยังฐานข้อมูลจำนวน 1,000 Metric ในทุกๆ 10 วินาที เป็นเวลา 60 นาที แล้วเก็บค่าเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลัก พบว่า จากค่าเฉลี่ยการปริมาณการใช้ทรัพยากรของ Prometheus ซึ่งเป็นฐานข้อมูลที่ถูกนำมาใช้ออกแบบระบบ มีค่าเฉลี่ยเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลักน้อยที่สุด เมื่อเทียบกับ Graphite และ InfluxDB



รูปที่ 4.6 การเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของฐานข้อมูลอนุกรมเวลา

ตารางที่ 4.6 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของฐานข้อมูลอนุกรมเวลา

เครื่องมือ	ค่าเปอร์เซ็นต์การใช้ซีพียูของฐานข้อมูลอนุกรมเวลา			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Graphite	5	2	13	4
InfluxDB	10	0	66	3
Prometheus	0	0	24	1



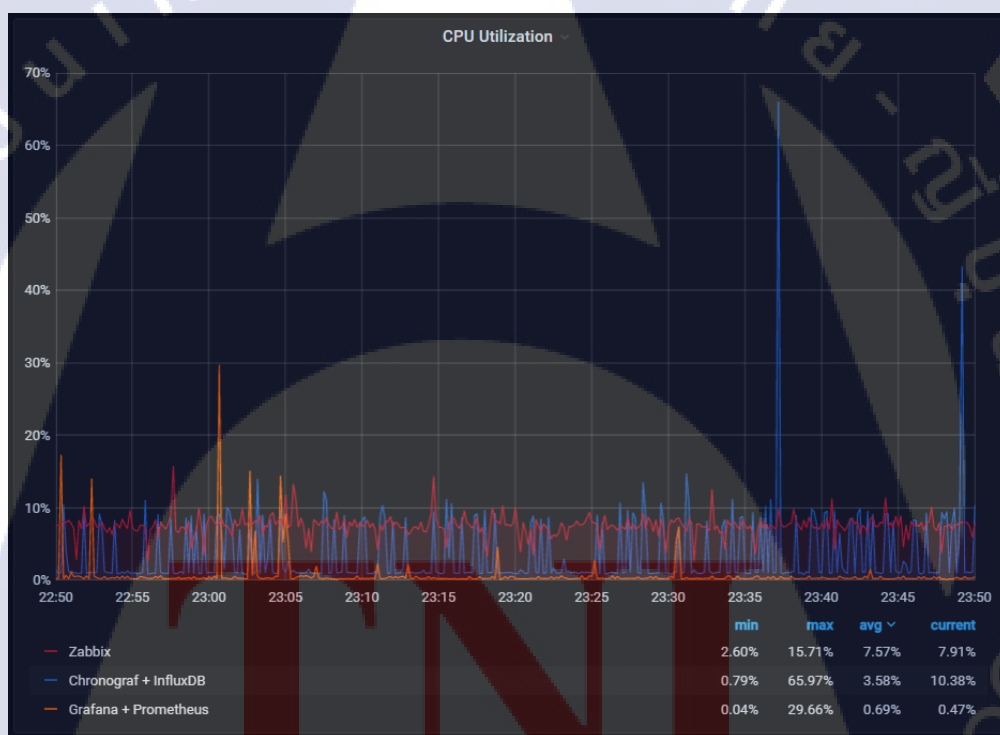
รูปที่ 4.7 การเปรียบเทียบค่าปริมาณการใช้หน่วยความจำหลักของฐานข้อมูลอนุกรมเวลา

ตารางที่ 4.7 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้หน่วยความจำสำรองของฐานข้อมูลอนุกรมเวลา

เครื่องมือ	ค่าเปอร์เซ็นต์การใช้หน่วยความจำสำรองของฐานข้อมูลอนุกรมเวลา			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Graphite	1.88	1.87	1.88	1.87
InfluxDB	1.68	1.68	1.69	1.68
Prometheus	0.33	0.33	0.41	0.33

4.2.2 ผลการทดสอบเปรียบเทียบปริมาณการใช้ทรัพยากรของเครื่องมือติดตามสถานะภาพ

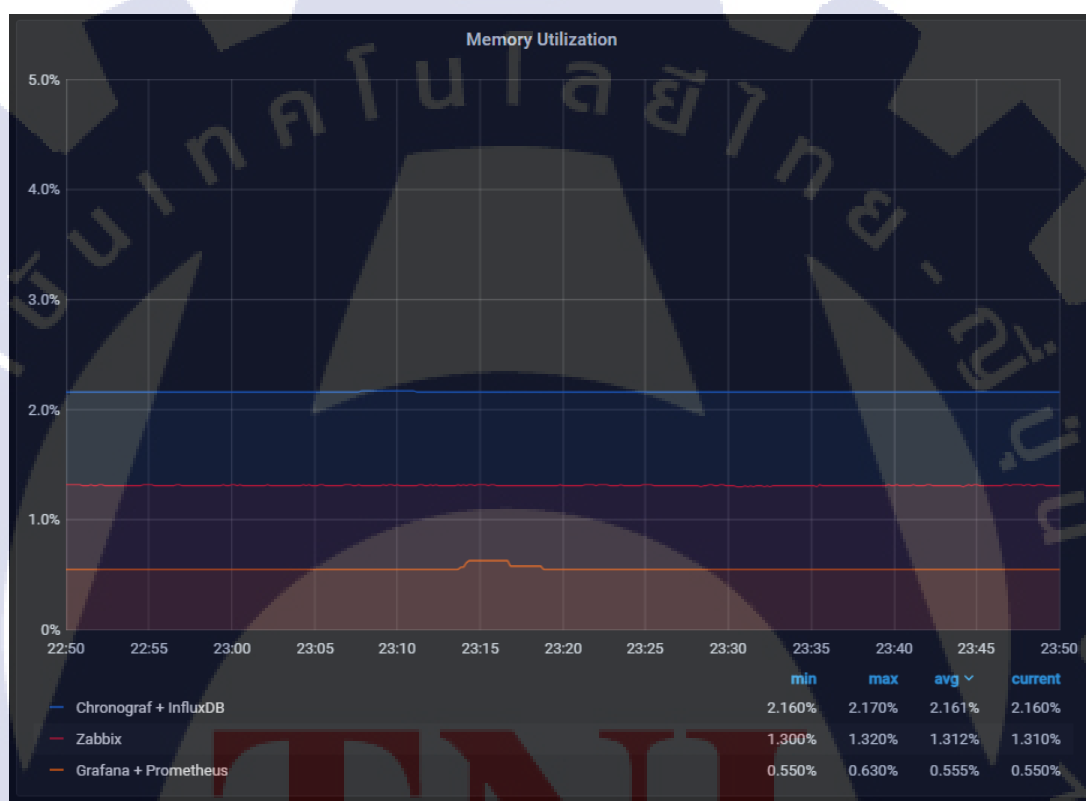
จากรูปที่ 4.8 และ 4.9 ผลการทดสอบวัดปริมาณ และเปรียบเทียบระหว่างเครื่องมือติดตามสถานะภาพ ที่นำมาใช้ในการออกแบบระบบ กับเครื่องมือติดตามสถานะภาพ ตัวอื่นๆ โดยการสร้างกราฟข้อมูลจำนวน 400 กราฟแล้วเก็บค่าเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลักในของแต่ละเครื่องมือเป็นระยะเวลา 60 นาที แต่เนื่องจาก Zabbix มีฐานข้อมูลรวมอยู่ด้วย เพื่อให้การทดสอบเครื่องมือมีความเท่าเทียมกันจึงต้องนำปริมาณการใช้งานของฐานข้อมูล Prometheus และ InfluxDB มาจับคู่คิดคำนวณรวมด้วยกันกับ Grafana และ Chronograf ตามลำดับ พบว่าจากค่าเฉลี่ยการปริมาณการใช้ทรัพยากรของ Grafana กับ Prometheus ซึ่งเป็นเครื่องมือที่ถูกนำมาใช้ออกแบบระบบ มีค่าเฉลี่ยเปอร์เซ็นต์การใช้ซีพียู และหน่วยความจำหลักน้อยที่สุด เมื่อเทียบกับ Zabbix และ Chronograf กับ InfluxDB



รูปที่ 4.8 การเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของเครื่องมือติดตามสถานะภาพ

ตารางที่ 4.8 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้ซีพียูของเครื่องมือติดตามสถานภาพ

เครื่องมือ	ค่าเปอร์เซ็นต์การใช้ซีพียูของเครื่องมือติดตามสถานภาพ			
	ค่าล่าสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Chronograf กับ InfluxDB	10.38	0.79	65.97	3.58
Zabbix	7.91	2.6	15.71	7.57
Grafana กับ Prometheus	0.47	0.04	29.66	0.69



รูปที่ 4.9 การเปรียบเทียบค่าเปอร์เซ็นต์ใช้หน่วยความจำหลักของเครื่องมือติดตามสถานภาพ

ตารางที่ 4.9 ตารางเปรียบเทียบค่าเปอร์เซ็นต์การใช้หน่วยความจำสำรองของเครื่องมือติดตามสถานภาพ

เครื่องมือ	ค่าเปอร์เซ็นต์การใช้หน่วยความจำสำรองของเครื่องมือติดตามสถานภาพ			
	ค่าต่ำสุด	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย
Chronograf กับ InfluxDB	2.16	2.16	2.17	2.16
Zabbix	1.31	1.3	1.32	1.31
Grafana กับ Prometheus	0.55	0.55	0.63	0.55

4.3 การแจ้งเตือนความผิดปกติ

จากการทดสอบด้านประสิทธิภาพการแจ้งเตือนความผิดปกติ โดยการสร้างสถานการณ์จำลองให้สถานภาพของเครื่องแม่ข่ายเสมือนมีความผิดปกติ โดยจากการทดสอบประสิทธิภาพ โดยการสร้างสถานการณ์จำลองให้สถานภาพของเครื่องแม่ข่ายเสมือนมีความผิดปกติ สามารถสรุปได้ดังตารางที่ 4.10

ตารางที่ 4.10 ผลการทดสอบประสิทธิภาพการตรวจสอบสถานภาพ และการแจ้งเตือน

สถานการณ์ผิดปกติ	จำนวนครั้ง	จำนวนครั้งแจ้งเตือน	อัตราความสำเร็จ
1. ปริมาณการใช้งานซีพียู มากกว่า 80 เปอร์เซ็นต์	10	10	100 %
2. ปริมาณการใช้งานหน่วยความจำหลัก มากกว่า 80 เปอร์เซ็นต์	10	10	100 %
3. ปริมาณการใช้งานหน่วยความจำสำรอง มากกว่า 80 เปอร์เซ็นต์	10	10	100 %
4. ไม่มีการตอบสนองต่อการ Ping	10	10	100 %
5. TCP พอร์ต 80 หรือ 443 ไม่ตอบสนอง	10	10	100 %
6. รหัสสถานภาพของ HTTP ไม่เท่ากับ 200	10	10	100 %

จากตารางที่ 4.10 ผลการทดสอบการตรวจสอบสถานภาพ โดยการจำลองสถานการณ์ผิดปกติ พบว่าจากทุกสถานการณ์ผิดปกติ ระบบสามารถตรวจสอบสถานภาพที่ผิดปกติ และแจ้งเตือนไปยังผู้ดูแลระบบได้ทุกครั้ง โดยอัตราความสำเร็จอยู่ที่ 100 เปอร์เซ็นต์

4.4 เวลาในการตอบสนองของส่วนติดตามสถานภาพ

จากการทดสอบด้านเวลาในการตอบสนองของส่วนติดตามสถานภาพ โดยการตรวจสอบค่าจาก Prometheus และ Grafana เป็นระยะเวลา 1 นาที สามารถสรุปได้ดังตารางที่ 4.11

ตารางที่ 4.11 เวลาในการตอบสนองของส่วนติดตามสถานภาพ

เวลาในการตอบสนอง	เวลาดำสุด	เวลาสูงสุด	เวลาเฉลี่ย
1. Prometheus กับ Data collection	100.8 ms	436.1 ms	126.6 ms
2. Prometheus กับ Grafana	20 ms	157 ms	77.1 ms
รวม	120.8 ms	593.1 ms	203.7 ms

จากตารางที่ 4.11 ผลการทดสอบเวลาในการตอบสนองสื่อสารข้อมูล ตั้งแต่ส่วน Data collection ไปจนถึงการนำค่าข้อมูลที่ได้แสดงที่ Grafana โดยเวลาเฉลี่ยในการตอบสนองระหว่าง Prometheus กับ ส่วน Data collection ใช้เวลาเฉลี่ย 126.6 มิลลิวินาที และระหว่าง Prometheus กับ Grafana ใช้เวลาเฉลี่ย 77.1 มิลลิวินาที โดยเวลารวมของทั้ง 2 ส่วน เวลาดำสุดอยู่ที่ 120.8 มิลลิวินาที เวลาเฉลี่ยอยู่ที่ 203.7 มิลลิวินาที และเวลาสูงสุดอยู่ที่ 593.1 มิลลิวินาที

4.5 การตรวจสอบปัญหา และกู้คืนระบบ

จากการทดสอบด้านประสิทธิภาพการตรวจสอบปัญหา และกู้คืนระบบโดยการสร้างสถานการณ์จำลองให้ส่วนประกอบของระบบเว็บแอปพลิเคชัน ที่อยู่ภายในเครื่อง Monitoring Client มีความผิดปกติสามารถสรุปผลทดสอบได้ตามตารางที่ 4.12 และ 4.13

4.5.1 ผลการทดสอบอัตราความสำเร็จในการกู้คืนระบบ

จากตารางที่ 4.12 ผลการทดสอบอัตราความสำเร็จในการกู้คืนระบบ โดยการจำลองสถานการณ์ที่ส่วนประกอบของเว็บแอปพลิเคชัน มีความผิดปกติประกอบด้วย 7 สถานการณ์ โดยทดสอบสถานการณ์ละ 10 ครั้ง พบว่าชุดคำสั่งไพทอนส่วน Diagnostic and Recovery สามารถกู้คืนส่วนประกอบที่มีปัญหาได้ทุกครั้ง คิดเป็นอัตราความสำเร็จที่ 100% ในการกู้คืนระบบ

ตารางที่ 4.12 ผลการทดสอบอัตราความสำเร็จในการกู้คืนระบบ

ส่วนประกอบ	สถานการณ์ ผิดปกติ	จำนวนครั้ง	จำนวนครั้งที่กู้คืน สำเร็จ	อัตรา ความสำเร็จ
1. Nginx	หยุดทำงาน	10	10	100%
2. Nginx	Port ถูกใช้งาน อยู่	10	10	100%
3. Container Web Database	หยุดทำงาน	10	10	100%
4. Container Web Database	Port ถูกใช้งาน อยู่	10	10	100%
5. Container Web Application	หยุดทำงาน	10	10	100%
6. Container Web Application	Port ถูกใช้งาน อยู่	10	10	100%
7. Docker	หยุดทำงาน	10	10	100%

4.5.2 ผลการทดสอบเปรียบเทียบระยะเวลาในการกู้คืนระบบ

จากตารางที่ 4.13 ผลการทดสอบเปรียบเทียบระยะเวลาในการกู้คืนระบบ จากการจำลองสถานการณ์ในข้อที่ 4.4.1 พบว่าชุดคำสั่งไฟทอนส่วน Diagnostic and Recovery สามารถลดเวลาในการกู้คืนระบบลงได้ประมาณ 12 วินาที จนถึง 44.8 วินาที แตกต่างกันไปตามสถานการณ์ เมื่อเปรียบเทียบกับเวลากู้คืนระบบเฉลี่ย โดยผู้ดูแลระบบ

ตารางที่ 4.13 ผลการเปรียบเทียบเวลาในการกู้คืนระบบ

ส่วนประกอบ	สถานการณ์ ผิดปกติ	เวลากู้คืนระบบ เฉลี่ย โดยผู้ดูแล ระบบ (วินาที)	เวลากู้คืนระบบ เฉลี่ย โดยชุดคำสั่ง ไพทอน (วินาที)	เปรียบเทียบ (วินาที)
Nginx	หยุดทำงาน	39.2	15	-24.2
Nginx	Port ถูกใช้งาน อยู่	64.8	20	-44.8
Container Web Database	หยุดทำงาน	35	20	-15
Container Web Database	Port ถูกใช้งาน อยู่	52	40	-12
Container Web Application	หยุดทำงาน	67	45	-22
Container Web Application	Port ถูกใช้งาน อยู่	84	45	-39
Docker	หยุดทำงาน	79.8	46	-33.8

บทที่ 5

สรุป อภิปรายผล และข้อเสนอแนะ

จากการศึกษาเรื่อง การออกแบบระบบตรวจสอบและแสดงสถานภาพเครื่องแม่ข่ายเสมือนภายในระบบการประมวลผลแบบกลุ่มเมฆ ซึ่งการศึกษาในครั้งนี้มีวัตถุประสงค์ 3 ข้อ ดังนี้

1. ออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน
2. เพื่อให้มีระบบแจ้งเตือนผู้เกี่ยวข้องให้ทราบ เมื่อเครื่องแม่ข่ายเสมือนทำงานผิดปกติ
3. เพื่อให้มีระบบตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันภายในเครื่องแม่ข่ายเสมือน

สามารถสรุปผลโดยแบ่งออกตามหัวข้อดังต่อไปนี้

- 5.1 สรุปผล และอภิปรายผล
- 5.2 ข้อเสนอแนะ
- 5.3 ประโยชน์ที่ได้รับจากการทำสารนิพนธ์

5.1 สรุปผล และอภิปรายผล

จากการทดสอบการทำงานของระบบตรวจสอบและแสดงสถานภาพเครื่องแม่ข่ายเสมือนภายในระบบการประมวลผลแบบกลุ่มเมฆ สามารถตรวจสอบสถานภาพของเครื่องแม่ข่ายเสมือน แล้วแสดงผลในรูปแบบของกราฟข้อมูล โดยใช้เครื่องมือ Grafana ซึ่งสามารถแสดงกราฟข้อมูลได้หลากหลายรูปแบบ ดูข้อมูลย้อนหลังได้ และยังสามารถแจ้งเตือนไปยังผู้ดูแลระบบได้เมื่อพบความผิดปกติตามที่ได้ตั้งค่า Threshold นอกจากนี้ยังสามารถนำมาประยุกต์ใช้เพื่อตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันที่อยู่ในเครื่องแม่ข่ายเสมือน ที่ใช้ในการทดสอบได้โดยอัตโนมัติ ช่วยลดเวลาในการตรวจสอบปัญหา และแก้ไขปัญหาให้ผู้ดูแลระบบได้ โดยสามารถสรุปผลตามวัตถุประสงค์ของการศึกษาได้ดังนี้

วัตถุประสงค์ข้อที่ 1 ออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน จากการออกแบบระบบติดตามสถานภาพเครื่องแม่ข่ายเสมือน ระบบสามารถตรวจสอบและแสดงค่าสถานภาพเครื่องแม่ข่ายเสมือนได้เป็นที่น่าสนใจ แต่มีข้อสังเกตเรื่องค่าสถานภาพที่ได้จากการใช้โพรโทคอล SNMP บางค่าที่มีความคลาดเคลื่อน ซึ่งเป็นข้อจำกัดของโพรโทคอล ได้แก่ ค่าเปอร์เซ็นต์การใช้งานซีพียู และค่าเปอร์เซ็นต์การใช้งานหน่วยความจำหลัก แต่ไม่ส่งผลกระทบมากนักกับการใช้งานเนื่องจากมีความคลาดเคลื่อนเล็กน้อยเมื่อเทียบกับค่าสถานภาพที่ได้จากเครื่องมือ Agent

ในส่วน of เครื่องมือที่นำมาใช้ได้แก่ Grafana และ Prometheus มีอัตราการบริโภคทรัพยากรเครื่องแม่ข่ายเสมือนที่ต่ำที่สุด เมื่อเทียบกับเครื่องมืออื่นๆที่นำมาเปรียบเทียบ ซึ่งจะทำให้ระบบตรวจสอบ และแสดงสถานะภาพสามารถรองรับการตรวจสอบสถานะภาพเครื่องแม่ข่ายเสมือนได้จำนวนที่มากกว่าเครื่องมืออื่น ทำให้สามารถใช้ทรัพยากรของเครื่องแม่ข่ายเสมือนได้อย่างคุ้มค่ามากที่สุด และเวลาในการตอบสนอง ระหว่างส่วน Data Collection, Prometheus และ Grafana รวมแล้วใช้เวลาไม่เกิน 1 วินาทีตั้งแต่การติดตามสถานะภาพ จนแสดงถึงการแสดงผลสำหรับ 1 เครื่องแม่ข่ายเสมือน

วัตถุประสงค์ข้อที่ 2 เพื่อให้มีระบบแจ้งเตือนผู้เกี่ยวข้องให้ทราบ เมื่อเครื่องแม่ข่ายเสมือนทำงานผิดปกติ จากการทดสอบระบบสามารถแจ้งเตือนไปยังผู้เกี่ยวข้องได้ทุกครั้งเมื่อค่าสถานะภาพมีความผิดปกติ

วัตถุประสงค์ข้อที่ 3 เพื่อให้มีระบบตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันภายในเครื่องแม่ข่ายเสมือน จากการทดสอบระบบตรวจสอบ และกู้คืนระบบเว็บแอปพลิเคชันสามารถทำงานได้อย่างมีประสิทธิภาพโดยจากการจำลองสถานการณ์ความผิดปกติ สามารถกู้คืนส่วนประกอบของเว็บแอปพลิเคชันได้ทุกครั้ง และยังตรวจสอบระบบได้ในเบื้องต้น ช่วยลดเวลาในการกู้คืนระบบได้

5.2 ข้อเสนอแนะ

5.2.1 ในการใช้โพรโทคอล SNMP เพื่อตรวจสอบสถานะภาพในบางค่าข้อมูลเช่น ข้อมูลหน่วยความจำหลัก ยังมีข้อจำกัดบางประการ ที่ทำให้ความแม่นยำน้อยกว่าค่าข้อมูลที่ได้จากการติดตั้งเครื่องมือ Agent ซึ่งต้องหาวิธีการอื่นมาประยุกต์ปรับปรุงค่าข้อมูลให้แม่นยำมากยิ่งขึ้น

5.2.2 เนื่องจากระบบตรวจสอบสถานะภาพ มีการเก็บประวัติค่าข้อมูลสถานะภาพของเครื่องแม่ข่ายเสมือนไว้ จึงมีความเป็นไปได้ที่จะนำค่าข้อมูลในอดีตมาวิเคราะห์ความสัมพันธ์ เพื่อทำนายแนวโน้มของค่าสถานะภาพ ที่จะเป็นประโยชน์ต่อการเตรียมป้องกันปัญหาที่อาจจะเกิดขึ้น

5.3 ประโยชน์ที่ได้จากการทำสารนิพนธ์

5.3.1 ประโยชน์ที่ผู้ศึกษาได้รับจากการศึกษา

(1) ได้เรียนรู้วิธีการในการศึกษาและค้นคว้าเอกสารและงานวิจัยต่างๆ ที่เกี่ยวข้องในการศึกษาค้นคว้าหาข้อมูลในเรื่องที่ทำการศึกษา

(2) ได้เรียนรู้แนวคิด และการออกแบบระบบตรวจสอบและแสดงสถานะภาพเครื่องแม่ข่ายเสมือน

(3) ได้เรียนรู้การแก้ไขปัญหาในเชิงรุก โดยการต่อยอดจากระบบติดตามสถานะภาพมาพัฒนาระบบตรวจสอบ และกู้คืนเว็บแอปพลิเคชันช่วยแบ่งเบาภาระในการทำงาน

5.3.2 ประโยชน์ที่ผู้อื่นได้รับจากการศึกษาครั้งนี้

- (1) ผู้ดูแลระบบเครื่องแม่ข่ายเสมือน สามารถทราบถึงความผิดปกติของระบบ และสาเหตุที่เกี่ยวข้องได้อย่างรวดเร็ว
- (2) ผู้ดูแลระบบเครื่องแม่ข่ายเสมือน มีเครื่องมือที่ช่วยตรวจสอบ และแก้ไขปัญหาอัตโนมัติได้ในเบื้องต้น



บรรณานุกรม

TNI

บรรณานุกรม

- [1] K. Costello and L. Goasduff, “Gartner Says Worldwide IaaS Public Cloud Services Market Grew 31.3% in 2018,” Gartner.com [Online]. Available: www.gartner.com/en/newsroom/press-releases/2019-07-29-gartner-says-worldwide-iaas-public-cloud-services-market-grew-31point3-percent-in-2018. [Accessed: January 18, 2020].
- [2] เกษม กมลชัยพิสิฐม, “ระบบวางแผนทรัพยากรองค์กรภายใต้การประมวลผลแบบกลุ่มเมฆ : โอกาสและความท้าทาย,” *วารสารการอาชีวและเทคนิคศึกษา*, ปีที่ 3, ฉบับที่ 6, หน้า 75-82, กรกฎาคม-ธันวาคม 2556.
- [3] A. Eizadirad, “IaaS, PaaS, SaaS (Explained and Compared),” LinkedIn.com, 2021. [Online]. Available: <https://www.linkedin.com/pulse/iaas-paas-saas-explained-compared-arsalan-eizadirad>. [Accessed: February 14, 2020].
- [4] อมร เจือดี, “SNMP คืออะไร,” สำนักวิทยบริการและเทคโนโลยีสารสนเทศ, [Online]. Available: <https://arit.kru.ac.th/myweb/?p=2025>. [Accessed: February 15, 2021].
- [5] M. Rouse, “Python,” whatis.techtarget.com [Online]. Available: whatis.techtarget.com/definition/Python. [Accessed: January 22, 2020].
- [6] Prometheus, “OVERVIEW,” prometheus.io [Online]. Available: prometheus.io/docs/introduction/overview/. [Accessed: February 15, 2020].
- [7] Grafana, “The Analytics Platform for All Your Metrics,” Grafana.com [Online]. Available: grafana.com/grafana/. [Accessed: January 29, 2020].
- [8] Grafana, “Get started or start exploring Grafana,” Grafana.com [Online]. Available: grafana.com/docs/ [Accessed: January 29, 2020].
- [9] K. Anshul, “Use of Open source technologies for enterprise server monitoring using SNMP,” *International Journal on Computer Science and Engineering*, vol. 02, no. 07, pp. 2246-2252, October 2010.
- [10] G. Nidhishree and S. Manimala, “Design and implementation of SNMP based network device monitoring system,” *International Journal of Trend in Research and Development*, vol. 4, no. 3, pp. 383-385, May-June 2017.

- [11] M. Brattstrom and P. Morreale, "Scalable agentless cloud network monitoring," *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, New York, NY, January 26-28 2017, pp. 171-176.
- [12] เชษฐ ศรีแย้ม, "การพัฒนาระบบติดตามสถานะเครือข่ายคอมพิวเตอร์ : กรณีศึกษาวิทยาลัยราชภัฏเพชรบุรี," วิทยานิพนธ์ ศศ.ม.(สนเทศศาสตร์), มหาวิทยาลัยศิลปากร, นครปฐม, 2557.
- [13] จักรินทร์ พิมพ์ดี, "ระบบตรวจสอบการทำงานของอุปกรณ์เครือข่ายและเครื่องแม่ข่ายกรณีศึกษา บริษัท แอมเซล จำกัด," สารนิพนธ์ วท.ม.(เทคโนโลยีสารสนเทศ), มหาวิทยาลัยเทคโนโลยีมหานคร, กรุงเทพฯ, 2560.



ภาคผนวก

TNI

ภาคผนวก ก.

ชุดคำสั่งตรวจสอบสถานภาพเครื่องแม่ข่ายเสมือน

TNI

ชุดคำสั่งตรวจสอบสถานภาพเครื่องแม่ข่ายเสมือน

Monitoring.py

```
from datetime import datetime
import time
from easysnmp import snmp_get, snmp_set, snmp_walk
import subprocess
import socket
from tcpping import tcpping
from flask import Flask, jsonify, request, Response, session, make_response
import requests
import threading
import urllib
from urllib.request import Request, urlopen
from urllib.error import URLError, HTTPError

cms='public' #Community String
host=[
    {'ip': '127.0.0.1', 'url': 'http://127.0.0.1', 'port': ['80','443']}

app = Flask(__name__) # Create my app

def cpu(ip, url):
    global metric
    start = time.time()
    try:
        cpu_core = snmp_walk('1.3.6.1.4.1.2021.11.67', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        cpu_core = int(cpu_core[0].value)
        cpu_percent = snmp_get('1.3.6.1.4.1.2021.11.11.0', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
```

```

cpu_percent = str(cpu_percent).split()[1]
cpu_percent = 100-int(cpu_percent[7:-1])
metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('cpu_percent', ip, url,
cpu_percent)

metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('cpu_core', ip, url, cpu_core)
print('CPU SNMP Success')
except:
    print('CPU SNMP Failed')
    pass

end = time.time()
print ("CPU Collection Time (ms): ",round((end-start)*1000,3))

def memory(ip, url):
    global metric
    start = time.time()
    try:
        mem_avai = snmp_get('.1.3.6.1.4.1.2021.4.6.0', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        mem_total = snmp_get('.1.3.6.1.4.1.2021.4.5.0', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        mem_buffer = snmp_get('.1.3.6.1.4.1.2021.4.14.0', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        mem_cache = snmp_get('.1.3.6.1.4.1.2021.4.15.0', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        mem_avai = str(mem_avai).split()[1]
        mem_avai = float(mem_avai[7:-1])
        mem_total = str(mem_total).split()[1]
        mem_total = float(mem_total[7:-1])
        mem_buffer = str(mem_buffer).split()[1]
        mem_buffer = float(mem_buffer[7:-1])

```

```

mem_cache = str(mem_cache).split()[1]
mem_cache = float(mem_cache[7:-1])
mem_used = mem_total-(mem_avai+mem_buffer+mem_cache)
mem_percent = (mem_used/mem_total)*100.0

metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('mem_percent', ip, url,
mem_percent)

metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('mem_total', ip, url,
mem_total)

metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('mem_used', ip, url,
mem_used)

print('Memory SNMP Success')
except:
    print('Memory SNMP Failed')
    pass

end = time.time()
print ("Memory Collection Time (ms): ",round((end-start)*1000,3))

def disk(ip, url):
    global metric
    start = time.time()
    try:
        disk_available = snmp_get('1.3.6.1.4.1.2021.9.1.7.1', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        disk_available = str(disk_available).split()[1]
        disk_available = float(disk_available[7:-1])
        disk_used = snmp_get('1.3.6.1.4.1.2021.9.1.8.1', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        disk_used = str(disk_used).split()[1]
        disk_used = float(disk_used[7:-1])

```



```

disk_percent = disk_used/(disk_used+disk_available)*100.0
disk_total = disk_used+disk_available

metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('disk_total', ip, url, disk_total)
metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('disk_percent', ip, url,
disk_percent)
metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('disk_available', ip, url,
disk_available)
metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('disk_used', ip, url, disk_used)
print('Disk SNMP Success')
except:
    print('Disk SNMP Failed')
    pass

end = time.time()
print ("Disk Collection Time (ms): ",round((end-start)*1000,3))

def traffic(ip, url):
    global metric
    start = time.time()

    try:
        ifHCInOctets = snmp_walk('1.3.6.1.2.1.31.1.1.1.6', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        ifHCOutOctets = snmp_walk('1.3.6.1.2.1.31.1.1.1.10', hostname=ip,
community=cms, version=2, timeout=1, retries=0)
        IfName = snmp_walk('1.3.6.1.2.1.31.1.1.1.1', hostname=ip, community=cms,
version=2, timeout=1, retries=0)
        for i in range(len(IfName)):
            metric += "%s{ip=\"%s\",url=\"%s\",interface=\"%s\"} %s \n" %
('bits_received', ip, url,str(IfName[i].value) ,str(int(ifHCInOctets[i].value)*8))

```

```

        metric += "%s{ip=\"%s\",url=\"%s\",interface=\"%s\"} %s \n" % ('bits_sent',
ip, url, str(IfName[i].value), str(int(ifHCOutOctets[i].value)*8))

    print('Traffic SNMP Success')

except:

    print('Traffic SNMP Failed')

    pass

end = time.time()
print ("Traffic Collection Time (ms): ",round((end-start)*1000,3))

def ping(ip, url):
    global metric
    global threads
    start = time.time()

    if subprocess.call(["ping", "-c", "1", "-W", "1", ip])==0:

        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('ping', ip, url, '1')

    try:

        t = threading.Thread(target=cpu, args=(ip, url))
        threads.append(t)
        t.start()

        t = threading.Thread(target=memory, args=(ip, url))
        threads.append(t)
        t.start()

        t = threading.Thread(target=disk, args=(ip, url))
        threads.append(t)
        t.start()

```

```

        t = threading.Thread(target=traffic, args=(ip, url))
        threads.append(t)
        t.start()

    except:
        pass

    print('Ping Success')

else:
    metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('ping', ip, url, '0')
    print('Ping Failed')
end = time.time()
print ("Ping Collection Time (ms): ",round((end-start)*1000,3))

def port(ip, url, port):
    global metric
    start = time.time()
    try:
        if tcpping(ip, port=port, timeout=1):
            metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url,
port, '1')
        else:
            metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url,
port, '0')
    except:
        metric += "%s{ip=\"%s\",url=\"%s\",port=\"%s\"} %s \n" % ('port', ip, url, port,
'0')

    end = time.time()

```

```

print ("Port Collection Time (ms): ",round((end-start)*1000,3))

def http(ip, url):
    global metric
    start = time.time()
    try:
        response = urllib.request.urlopen('http://'+url, timeout = 1)
    except HTTPError as e:
        print('Error code: ', e.code)
        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('http_code', ip, url, e.code)
    except URLError as e:
        print('Reason: ', e.reason)
        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('http_code', ip, url, '0')
    except socket.timeout:
        print('Reason: ', socket.timeout)
        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('http_code', ip, url, '0')
    except requests.exceptions.RequestException:
        print('Reason: ',requests.exceptions.RequestException)
        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('http_code', ip, url, '0')
        raise Exception('Failed to connect to %s' % url) from None
    else:
        print ('HTTP Code: ',response.code)
        metric += "%s{ip=\"%s\",url=\"%s\"} %s \n" % ('http_code', ip, url,
response.code)

    end = time.time()
    print ("HTTP Collection Time (ms): ",round((end-start)*1000,3))

@app.route("/metrics", methods=["GET"])
def main():
    global threads
    threads = []

```

```

start = time.time()
global metric
metric = ""

for i in host: # Thread Group 1 : Ping Status + SNMP
    t = threading.Thread(target=ping, args=(i['ip'], i['url']))
    threads.append(t)
    t.start()

for i in host: # Thread Group 2 : Port Status
    for j in range(len(i['port'])):
        t = threading.Thread(target=port, args=(i['ip'], i['url'], i['port'][j]))
        threads.append(t)
        t.start()

for i in host: # Thread Group 3 : HTTP Status
    t = threading.Thread(target=http, args=(i['ip'], i['url']))
    threads.append(t)
    t.start()

for t in threads:
    t.join()

end = time.time()
print ("-----Total Data Collection Time (ms): ",round((end-start)*1000,3))

return metric

if __name__ == '__main__':
    app.run(host="127.0.0.1",port=8001)

```

ภาคผนวก ข.

ชุดคำสั่งตรวจสอบ และกู้คืนเว็บแอปพลิเคชัน

TNI

ชุดคำสั่งตรวจสอบ และกู้คืนเว็บแอปพลิเคชัน

diagnostic&recovery.py

```
from datetime import datetime
import time
import subprocess
from flask import Flask, jsonify, request, Response, session, make_response
import requests
import threading
import paramiko
import sys
import re
import json

url = 'https://notify-api.line.me/api/notify' #Line Notify
token = 'xxxxxxxxxxxxxxxxxxxxxxxx' #Token
headers = {'content-type':'application/x-www-form-urlencoded','Authorization':'Bearer '+token}

ssh = paramiko.SSHClient() # ssh utility
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ansi_escape = re.compile(r'\x0f\x1B(?:[@-Z\\-_]|\\[[0-?]*[ -/]*[@-~])') # ANSI
Character

app = Flask(__name__) # Create my app

def notify(msg):
    r = requests.post(url, headers=headers, data = {'message':str(msg)})
    print (r.text)
```



```

def fix_docker(incident, container):

    ssh.connect(incident['ip'], port=22, username='username', password='password')
    #User/Pass For access via SSH
    msg="IP= %s\n"%(192.168.xxx.xxx)+"Container Name= %s\n"%(container)
    print('Checking Container Status')
    stdin, stdout, stderr = ssh.exec_command("sudo docker container
inspect -f '{{.State.Running}}' "+container,get_pty=True) #Check docker status
    running =stdout.readlines()

    if "No such container" in str(running): #Check Error: No such container
        running=running[-1].splitlines()
        error_msg=running[0]
        print('Container not Found')
    else:
        running=running[0].splitlines()
        running=str(running[0])

    if running=='false': #Docker running=false then restart
        stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect -f
('{{.State.ExitCode}}' "+container,get_pty=True)
        exitcode = stdout.readlines()
        exitcode=exitcode[0].splitlines()
        exitcode=str(exitcode[0])
        msg=msg+"Container Running= %s\n"%(running)
        msg=msg+"Container Error= %s\n"%(exitcode)

        if exitcode=='137':
            print('Container ExitCode 137')

```

```

        stdin, stdout, stderr = ssh.exec_command("sudo sync; echo 2 >
/proc/sys/vm/drop_caches",get_pty=True) #Clear Memory Cache

        stdin, stdout, stderr = ssh.exec_command("sudo docker restart
"+container,get_pty=True)

        ('Container Restarting')

        errorcheck=stdout.readlines() #Error check
        if "Error" in str(errorcheck):
            if "address already in use" in str(errorcheck):
                print('Error : Address already in use')

                stdin, stdout, stderr = ssh.exec_command("sudo docker container
inspect -f '{{.HostConfig.PortBindings}}' "+container,get_pty=True)

                hostport = stdout.readlines() # check docker setting port
                hostport=hostport[0].splitlines()
                hostport=hostport[0].split(' ')
                hostport=hostport[1].split('{')
                hostport=hostport[0]

                stdin, stdout, stderr = ssh.exec_command("sudo lsof -i :"+hostport,
get_pty=True) #check PID

                address_check=stdout.readlines()
                address_check=address_check[1].split()

                msg=msg+"Container Error= %s\n"%(Address already in use')
                msg=msg+"Using Command= %s\n"%(address_check[0])
                msg=msg+"Using PID= %s\n"%(address_check[1])
                msg=msg+"Using By= %s\n"%(address_check[2])

                detail={}
                detail['reason']='Address already in use'
                detail['using_command']=address_check[0]
                detail['using_pid']=address_check[1]
                detail['using_by']=address_check[2]

```

```

        incident['detail']=detail

        print('Running Process Killing')

        stdin, stdout, stderr = ssh.exec_command("sudo kill -9
"+address_check[1], get_pty=True) #kill PID
        time.sleep(1)
        stdin, stdout, stderr = ssh.exec_command("sudo docker restart
"+container,get_pty=True)

        print('Container Restarting')

        errorcheck=stdout.readlines()
    else:
        pass
        errorcheck=errorcheck[-1].splitlines()
        error_msg=errorcheck[0]
    elif exitcode=='128':
        print('Container ExitCode 128')
        stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect
-f '{{.HostConfig.PortBindings}}' "+container,get_pty=True)
        hostport = stdout.readlines() # check docker setting port
        hostport=hostport[0].splitlines()
        hostport=hostport[0].split(' ')
        hostport=hostport[1].split('}')
        hostport=hostport[0]
        stdin, stdout, stderr = ssh.exec_command("sudo lsof -i :"+hostport,
get_pty=True) #check PID
        address_check=stdout.readlines()
        address_check=address_check[1].split()

    msg=msg+"Container Error= %s\n"%(Address already in use')
    msg=msg+"Using Command= %s\n"%(address_check[0])
    msg=msg+"Using PID= %s\n"%(address_check[1])
    msg=msg+"Using By= %s\n"%(address_check[2])

```

```

detail={}
detail['reason']='Address already in use'
detail['using_command']=address_check[0]
detail['using_pid']=address_check[1]
detail['using_by']=address_check[2]
incident['detail']=detail

print('Running Process Killing')

stdin, stdout, stderr = ssh.exec_command("sudo kill -9
"+address_check[1], get_pty=True) #kill PID
time.sleep(1)
stdin, stdout, stderr = ssh.exec_command("sudo docker restart
"+container,get_pty=True)

print('Container Restarting')
errorcheck=stdout.readlines() #Error check
if "Error" in str(errorcheck):
    errorcheck=errorcheck[-1].splitlines()
    error_msg=errorcheck[0]
else: # if exitcode=='0' or exitcode=='143':
    print('Container ExitCode 0 or 143')
    stdin, stdout, stderr = ssh.exec_command("sudo docker restart
"+container,get_pty=True)
    errorcheck=stdout.readlines()
    if "Error" in str(errorcheck):
        errorcheck=errorcheck[-1].splitlines()
        error_msg=errorcheck[0]

time.sleep(1) # delay for wait docker restarting
stdin, stdout, stderr = ssh.exec_command("sudo docker container inspect -f
'{{.State.Running}}' "+container,get_pty=True) #Check after restart docker
finalcheck = stdout.readlines()

```

```

finalcheck=finalcheck[0].splitlines()
finalcheck=str(finalcheck[0])
msg=msg+"---Recovery Result ---\n"

if finalcheck=='false':
    print('Recovery : Failed')
    msg=msg+"Container Running= %s\n"%(finalcheck)
    msg=msg+"Error Msg= %s\n"%(error_msg)
elif finalcheck=='true':
    print('Recovery : Success')
    msg=msg+"Container Running= %s\n"%(finalcheck)
    msg=msg+"Solve Time=
%s\n"%(datetime.fromtimestamp(time.time()).strftime('%d-%m-%Y %H:%M:%S'))
else:
    msg=msg+"Container Running= %s\n"%(finalcheck)
elif running=='true':
    msg=msg+"Container Running= %s\n"%(running)
else:
    msg=msg+"Error Msg= %s\n"%(error_msg)
print('Send Result to LINE')
notify(msg)
ssh.close()

def fix_service(incident, service):

    ssh.connect(incident['ip'], port=22, username='user01', password='Admin@inet!')
    check=[]
    msg="IP= %s\n"%('192.168.xxx.xxx')+"Service Name= %s\n"%(service)
    print('Checking Service Status')
    stdin, stdout, stderr = ssh.exec_command('sudo systemctl show '+service+' --
property=ActiveState',get_pty=True) #Check service state

```

```

running=stdout.readlines()
running=running[0].splitlines()
running=ansi_escape.sub("", running[1]).split('=')
running=running[1]

if running=='failed' or running=='inactive':
    print('Service Not Running')
    msg=msg+"Service State= %s\n"%(running)
    stdin, stdout, stderr = ssh.exec_command('sudo systemctl restart
'+service,get_pty=True)
    running=stdout.readlines()
    if running==[]:
        pass
    else:
        stdin, stdout, stderr = ssh.exec_command('sudo systemctl status
'+service)

        print('Restarting Service')
        for line in iter(stdout.readline, ""): #get status line
            check.append(line)

        if 'Address already in use' in str(check):
            print('Error : Address already in use')
            stdin, stdout, stderr = ssh.exec_command('sudo lsof -
i:'+incident['port']) #check service binded to port
            address_check=stdout.readlines()
            address_check=address_check[1].split()

            msg=msg+"Service Error= %s\n%"('Address already in use')
            msg=msg+"Using Command= %s\n"%(address_check[0])
            msg=msg+"Using PID= %s\n"%(address_check[1])
            msg=msg+"Using By= %s\n"%(address_check[2])

```

```

        detail={}
        detail['reason']='Address already in use'
        detail['using_command']=address_check[0]
        detail['using_pid']=address_check[1]
        detail['using_by']=address_check[2]
        incident['detail']=detail

        stdin, stdout, stderr = ssh.exec_command("sudo kill -9
"+address_check[1], get_pty=True) #kill PID
        stdin, stdout, stderr = ssh.exec_command('sudo systemctl restart
'+service,get_pty=True)
    else:
        pass

    time.sleep(1)
    stdin, stdout, stderr = ssh.exec_command('sudo systemctl show '+service+'-
-property=ActiveState',get_pty=True)
    finalcheck=stdout.readlines()
    finalcheck=finalcheck[0].splitlines()
    finalcheck=ansi_escape.sub("", finalcheck[1]).split('=')
    finalcheck=finalcheck[1]
    msg=msg+"---Recovery Result ---\n"

    if finalcheck=='failed' or finalcheck=='inactive':
        msg=msg+"Service State= %s\n"%(finalcheck)
        msg=msg+"Error Msg= %s\n"%(str(check))
        print('Recovery : Failed')
    elif finalcheck=='active':
        msg=msg+"Service State= %s\n"%(finalcheck)

```



```

        msg=msg+"Solve Time=
%s\n"%(datetime.fromtimestamp(time.time()).strftime('%d-%m-%Y %H:%M:%S'))
        rint('Recovery : Success')

    else:
        msg=msg+"Service State= %s\n"%(finalcheck)
    else:
        msg=msg+"Service State= %s\n"%(running)
        print('Send Result to LINE')
    notify(msg)
    ssh.close()

def check_firewall(incident):

    ssh.connect(incident['ip'], port=22, username='user01', password='Admin@inet!')

    msg="IP= %s\n"%('192.168.xxx.xxx')+ "---Firewall---\n"
    check=[]
    detail=[]

    stdin, stdout, stderr = ssh.exec_command('sudo ufw status|grep
'+incident['port']) #Filter port

    for line in iter(stdout.readline, ""):
        check.append(ansi_escape.sub("", line))

    if check==[]:
        detail.append({'Port': incident['port'], 'Permit': 'Not Found', 'From': 'Not Found'
        })
        msg=msg+"Port= %s\n"%(incident['port'])
        msg=msg+" -Permit= %s\n"%('Not Found')
        msg=msg+" -From= %s\n"%('Not Found')

```

```

        print('Rule Not Found!')

    else:

        print('Rule Found!')

        for line in check: #Firewall rule find port
            line=line.split()
            if line[0]==incident['port'] or re.match(r"%s\\W[a-z]*" %(port), line[0]):
                detail.append({'Port': line[0], 'Permit': line[1], 'From': line[2] })
                msg=msg+"Port= %s\\n"%(line[0])
                msg=msg+" -Permit= %s\\n"%(line[1])
                msg=msg+" -From= %s\\n"%(line[2])
            else:
                pass

        if detail==[]:
            detail.append({'Port': incident['port'], 'Permit': 'Not Found', 'From': 'Not
Found' })
            msg=msg+"Port= %s\\n"%(incident['port'])
            msg=msg+" -Permit= %s\\n%('Not Found')
            msg=msg+" -From= %s\\n%('Not Found')
        else:
            pass

        incident['detail']=detail
        print('Send Result to LINE')
        notify(msg)
        ssh.close()

def check_usage(incident, component):

    ssh.connect(incident['ip'], port=22, username='user01', password='Admin@inet!')
    msg="IP= %s\\n%('192.168.xxx.xxx')+ "---%s Usage---\\n"%(component)

```

```

check=[]
detail=[]

print('Top 1-5 Usage Running Process checking')
stdin, stdout, stderr = ssh.exec_command('top -n1 -w512 -o %'+component,
get_pty=True)

for line in iter(stdout.readline, ""):
    check.append(ansi_escape.sub("", line)) #remove ANSI + append

for line in check[7 :12]: #Top 1-5 Usage
    line=line.split()
    detail.append({'user': line[1], component: line[8], 'command': line[11] })
    msg=msg+"Command: %s\n"%(line[11])
    msg=msg+" -"+component+": "+line[8]+" %\n"
    msg=msg+" -User: %s\n"%(line[1])

incident['detail']=detail

    print('Send Result to LINE')

notify(msg)
ssh.close()

def classify(incident):

    print('Metric: ',incident['metric']);
    print('Value: ',incident['value']);
    print('IP: ',incident['ip']);

    if incident['metric']=='http_code':
        if incident['value']=='500':
            print('http_code = 404')
            fix_docker(incident, 'web-database')

```

```

elif incident['value']=='404':
    print('http_code = 404')
    fix_service(incident, 'docker')
    fix_docker(incident, 'web-application')
    fix_docker(incident, 'web-database')
else:
    pass
elif incident['metric']=='port':
    if incident['port']=='80' or incident['port']=='443':
        print('port = 0')
        fix_service(incident, 'nginx')
        check_firewall(incident)
    elif incident['metric']=='cpu_percent':
        print('cpu_percent > 80%')
        check_usage(incident, 'CPU')
    elif incident['metric']=='mem_percent':
        print('mem_percent > 80%')
        check_usage(incident, 'MEM')
    else:
        pass

@app.route('/shs', methods=['POST'])
def respond():
    incident={}

    if request.json['evalMatches'][0]['tags']['__name__']=='port':
        incident={
            "metric": request.json['evalMatches'][0]['tags']['__name__'],
            "port": request.json['evalMatches'][0]['tags']['port'],
            "value": str(request.json['evalMatches'][0]['value']),
            "ip": request.json['evalMatches'][0]['tags']['ip'],

```

```
"url": request.json['evalMatches'][0]['tags']['url'],
"timestamp": datetime.fromtimestamp(time.time()).strftime('%Y-%m-
%d %H:%M:%S')
}

print('Receive JSON Data')

else:
    incident={
        "metric": request.json['evalMatches'][0]['tags']['__name__'],
        "value": str(request.json['evalMatches'][0]['value']),
        "ip": request.json['evalMatches'][0]['tags']['ip'],
        "url": request.json['evalMatches'][0]['tags']['url'],
        "timestamp": datetime.fromtimestamp(time.time()).strftime('%Y-%m-
%d %H:%M:%S')
    }

    print('Receive JSON Data')

try:
    return Response(status=200)
finally:
    classify(incident)

if __name__ == '__main__':
    app.run(host="127.0.0.1",port=8082)
```