

Comparative Analysis of Pull and Push Monitoring Models for Information Services in CGNAT Environments

Chayaphon Suwanvorn^{1,*} and Chaiyaporn Khemapatapan²

^{1,2}Department of Computer Engineering, College of Engineering and Technology,
Dhurakij Pundit University, Bangkok 10210, Thailand; Email: ¹66130918@dpu.ac.th, ²chaiyaporn@dpu.ac.th

*Corresponding author: Chayaphon Suwanvorn (66130918@dpu.ac.th)

Received: 17 March 2026; Revised: 20 May 2026; Accepted: 23 May 2026; Published: 22 August 2026

Abstract

This research investigates the development of a monitoring system for information environments operating behind Carrier-Grade NAT (CGNAT), a significant constraint that prevents direct access from a central server to client nodes. The study conducts an in-depth comparative analysis of two architectures: (1) a Pull Model operating over an overlay network utilizing Tailscale technology, and (2) a traditional Push Model transmitting data via a Pushgateway. Experiments were performed on a live system deployed on Microsoft Azure, with network conditions simulated using pfSense. The results reveal significant trade-offs between the two approaches. The Pull Model via Tailscale demonstrates superior performance regarding alerting latency, real-time data continuity, and enhanced security, as it eliminates the need to expose public ports. Conversely, the Push Model is distinguished by lower system overhead on client nodes and greater configuration autonomy. These findings provide guidelines for architectural selection based on specific use cases. The Pull Model is recommended for scenarios prioritizing high security where specialized configuration expertise is limited; Tailscale's underlying Zero Trust Network Access (ZTNA) simplifies deployment through single-account authentication without complex setup. Alternatively, the Push Model is ideal for resource-constrained devices, such as IoT, provided that expert personnel are available to ensure secure configuration.

Keywords: Prometheus, CGNAT, Tailscale, Pull Model, Push Model



Copyright © 2026 Journal of Computer and Creative Technology. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC-BY-NC-ND).

Citation:

Suwanvorn, C., & Khemapatapan, C. (2026). Comparative Analysis of Pull and Push Monitoring Models for Information Services in CGNAT Environments. *Journal of Computer and Creative Technology*, 4(2), e3760.

<https://doi.org/10.65205/jcct.2026.e3760>

การพัฒนาระบบเฝ้าระวังประสิทธิภาพและความปลอดภัยพื้นฐานของระบบให้บริการสารสนเทศ

ชยพล สุวรรณาร^{1,*} และ ชัยพร เขมะภาคะพันธ์²

^{1,2}สาขาวิชาวิศวกรรมคอมพิวเตอร์ วิทยาลัยวิศวกรรมศาสตร์และเทคโนโลยี มหาวิทยาลัยธุรกิจบัณฑิต
จังหวัดกรุงเทพมหานคร 10210 ประเทศไทย; Email: ¹66130918@dpu.ac.th, ²chaiyaporn@dpu.ac.th

*ผู้ประพันธ์บทความ: ชยพล สุวรรณาร (66130918@dpu.ac.th)

รับบทความ: 17 มีนาคม 2569; แก้ไขบทความ: 20 พฤษภาคม 2569; ตอรับบทความ: 23 พฤษภาคม 2569;
เผยแพร่บทความ: 22 สิงหาคม 2569

บทคัดย่อ

งานวิจัยนี้ศึกษาวิธีการพัฒนาระบบเฝ้าระวังและติดตามประสิทธิภาพสำหรับระบบ สารสนเทศที่ทำงานอยู่ภายใต้สภาพแวดล้อมเครือข่ายแบบ Carrier-Grade NAT (CGNAT) ซึ่งเป็นข้อจำกัดสำคัญที่ทำให้ Server ส่วนกลางไม่สามารถเข้าถึงโหนดปลายทางได้โดยตรง งานวิจัยนี้จึงได้ทำการเปรียบเทียบเชิงลึกระหว่าง สถาปัตยกรรม 2 รูปแบบคือ 1. รูปแบบ Pull Model ที่ทำงานผ่านเครือข่ายส่วนตัวเสมือนด้วยเทคโนโลยีของ Tailscale และ 2. รูปแบบ Push Model แบบดั้งเดิมที่ส่งข้อมูลผ่าน Pushgateway โดยทำการทดสอบ บนระบบจริงที่ติดตั้งบน Azure และจำลองเครือข่ายด้วย pfSense ผลการศึกษาพบข้อแลกเปลี่ยนที่สำคัญ คือ สถาปัตยกรรมแบบ Pull Model ผ่าน Tailscale มีความโดดเด่นด้านความเร็วในการแจ้งเตือนความต่อเนื่องของข้อมูลหรือความ Real-Time และความปลอดภัยที่สูงกว่าเนื่องจากไม่ต้องเปิดพอร์ตสาธารณะ ในขณะที่สถาปัตยกรรมแบบ Push Model มีจุดเด่นด้านการใช้ทรัพยากรระบบบนโหนดปลายทางที่ต่ำกว่าและมีความเป็นอิสระในการตั้งค่าระบบมากกว่า ข้อค้นพบนี้สามารถใช้เป็นแนวทางในการ เลือกสถาปัตยกรรมให้เหมาะสมกับลักษณะงาน เช่น การใช้ Pull Model สำหรับระบบที่เน้นความมั่นคงความปลอดภัยสูง แต่ขาดผู้เชี่ยวชาญในการตั้งค่าความปลอดภัย การเลือกสถาปัตยกรรมแบบ Pull Model จึงเป็นทางเลือกที่ดีเพราะ Tailscale มีพื้นฐานความปลอดภัยแบบ Zero Trust Network Access (ZTNA) เป็นพื้นฐานอยู่แล้วเพียงแค่เข้าสู่ระบบ ด้วยบัญชีผู้ใช้เดียวกันก็สามารถใช้งานได้โดยไม่ต้องตั้งค่าซับซ้อน ส่วนสถาปัตยกรรมแบบ Push Model เหมาะสำหรับอุปกรณ์ที่มีทรัพยากรจำกัด เช่น IoT และจำเป็นต้องมีผู้เชี่ยวชาญในการตั้งค่าระบบเพื่อความปลอดภัย

คำสำคัญ: โพรมิธีอัส, ซีจีแนต, เทลสเกล, พูลโมเดล, พูชโมเดล



Copyright © 2026 Journal of Computer and Creative Technology. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC-BY-NC-ND).

การอ้างอิง:

ชยพล สุวรรณาร, และ ชัยพร เขมะภาคะพันธ์. (2569). การพัฒนาระบบเฝ้าระวังประสิทธิภาพและความปลอดภัยพื้นฐานของระบบให้บริการสารสนเทศ. วารสารคอมพิวเตอร์และเทคโนโลยีสร้างสรรค์, 4(2), e3760. <https://doi.org/10.65205/jcct.2026.e3760>

1. บทนำ (Introduction)

ในปัจจุบันระบบเทคโนโลยีสารสนเทศเป็นหัวใจสำคัญของการขับเคลื่อนธุรกิจ จากรายงานสถิติ ภัยคุกคามทางด้านไซเบอร์ (Cyber Attack Trends 2025) ประจำปี 2568 อ้างอิงจากข้อมูลของสำนักงานคณะกรรมการ การรักษาความมั่นคงปลอดภัยไซเบอร์แห่งชาติ (สกมช.) (National Cyber Security Agency, 2025) พบว่าการพยายามโจมตีระบบเครือข่ายมีอัตราเพิ่มสูงขึ้น ซึ่งหากระบบเฝ้าระวัง (Monitoring System) เกิดความล่าช้าหรือล้มเหลวเพียงไม่กี่นาทีอาจส่งผลกระทบต่อความเชื่อมั่นและสร้างความเสียหายเชิงธุรกิจอย่างมหาศาล ดังนั้นการเฝ้าระวังและติดตามสถานะของระบบจึงเป็นสิ่งที่ขาดไม่ได้เพื่อให้ผู้ดูแลระบบสามารถรับรู้ถึง ความผิดปกติและแก้ไขปัญหาได้ทันเวลาที่เครื่องมือ ที่ได้รับความนิยมอย่างแพร่หลายคือ Prometheus และ Grafana ซึ่งโดยปกติจะทำงานในรูปแบบ Pull Model (Saputra et al., 2024; Sirviö, 2021) หรือการดึงข้อมูล ตรงจากเครื่องปลายทาง แต่ว่าในยุคปัจจุบันที่ IPv4 ขาดแคลนผู้ให้บริการอินเทอร์เน็ต ส่วนใหญ่จึงหันมาใช้เทคโนโลยี Carrier- Grade NAT (CGNAT) (Hriţcan & Balan, 2024a; netElastic, n.d.) ทำให้โหนดปลายทางไม่มี IP ที่เป็น Public IP เป็น ของตัวเองส่งผลให้ Server ส่วนกลางไม่สามารถเข้าถึงเพื่อดึงข้อมูลได้ ปัญหานี้ส่งผลกระทบต่อโดยตรงต่อการ บริหารจัดการ ระบบไอทีสมัยใหม่ไม่ว่าจะเป็นการ Monitor เครื่องคอมพิวเตอร์ของพนักงานที่ทำงานในรูปแบบ Work From Anywhere (WHA) หรืออุปกรณ์ IoT (Hriţcan & Balan, 2024a; Saputra et al., 2024) ที่กระจายอยู่ ตามพื้นที่ต่าง ๆ จากการทบทวนวรรณกรรมที่ผ่านมาวิธีแก้ปัญหานี้มักมุ่งเน้นไปที่การใช้วิธีแบบดั้งเดิมคือใช้การ Push ข้อมูลผ่าน Pushgateway (Hellstern, 2022; Prometheus, n.d.a) ซึ่งแม้จะช่วยให้ส่งข้อมูลออกมาได้แต่ก็แลก มาด้วยความเสี่ยงด้านความปลอดภัยจากการเปิดพอร์ตสาธารณะ เช่น Port 9091 เพื่อรับข้อมูลจากภายนอก ซึ่งเป็นการเพิ่มพื้นที่การโจมตี (Attack Surface) ให้แก่ระบบ นอกจากนี้การใช้เทคโนโลยี Virtual Private Network (VPN) แบบดั้งเดิมเพื่อสร้างอุโมงค์ข้อมูล (Hriţcan & Balan, 2024a; Hriţcan et al., 2025) แม้จะช่วยแก้ปัญหาการเชื่อมต่อได้แต่ก็มักพบปัญหาเรื่อง ความซับซ้อนในการบริหารจัดการ Routing และมีภาระของระบบ (Overhead) สูง ซึ่งส่งผลกระทบต่อความต่อเนื่อง ของข้อมูลในระบบเฝ้าระวัง ด้วยเหตุนี้งานวิจัยนี้จึงมุ่งเน้นการเติมเต็มช่องว่างทางวิจัย (Research Gap) โดยการศึกษาและเปรียบเทียบประสิทธิภาพและความมั่นคงปลอดภัยของสถาปัตยกรรมแบบ Push Model แบบดั้งเดิมกับแนวทางการประยุกต์ใช้ Overlay Network บนพื้นฐานของแนวคิด Zero Trust Network Access (ZTNA) (Cloudflare, n.d.) โดยงานวิจัยนี้แนะนำการประยุกต์ใช้ Tailscale ร่วมกับ Microsoft Azure และ pfSense เพื่อออกแบบและพัฒนาสถาปัตยกรรมระบบเฝ้าระวังรูปแบบใหม่ที่สามารถทำงานได้อย่างมีประสิทธิภาพและเหมาะสมกับการใช้งานในรูปแบบต่าง ๆ ภายใต้ข้อจำกัดของเครือข่าย CGNAT ทั้งนี้ผลงานวิจัยมีส่วนสนับสนุนทางวิชาการและการประยุกต์ใช้งาน (Research Contributions) ดังต่อไปนี้

1. ด้านวิศวกรรมประยุกต์ (Applied Engineering Contribution) แนะนำแนวทางการพัฒนาระบบ เฝ้าระวังแบบผสมผสาน (Hybrid Monitoring Architecture) ที่สามารถใช้กลไกแบบ Pull Model ของ Prometheus ผ่านเครือข่าย CGNAT ได้โดยไม่ต้องเปิดพอร์ตสาธารณะ

2. ด้านการวิเคราะห์เชิงเปรียบเทียบ (Comparative Contribution) จัดทำข้อมูลเชิงประจักษ์ (Empirical Evidence) เพื่อเปรียบเทียบตัวชี้วัด ได้แก่ ภาระของระบบ (System Overhead) ความหน่วงในการแจ้งเตือน (Latency) และความเสถียรของข้อมูล (Data Stability) ระหว่างสองสถาปัตยกรรมเพื่อสนับสนุนการ ตัดสินใจเชิงเทคโนโลยีในองค์กร

3. ด้านทฤษฎี (Theoretical Contribution) ขยายขอบเขตแนวคิด Zero Trust Network Access (ZTNA) ไปสู่มิติของ Observability โดยแสดงให้เห็นว่าการเชื่อมต่อแบบอิงตัวตนของโหนด (Identity-Based Connectivity) สามารถลดความขัดแย้งระหว่างการเข้าถึงข้อมูล (Reachability) และความมั่นคงปลอดภัย (Security) ในเครือข่ายที่ไม่น่าเชื่อถือ (Untrusted Networks) ได้อย่างมีประสิทธิภาพ

2. วัตถุประสงค์งานวิจัย (Research Objectives)

1. เพื่อพัฒนาและออกแบบระบบเฝ้าระวังประสิทธิภาพและความปลอดภัยให้สามารถทำงานผ่านเครือข่ายที่มีข้อจำกัด CGNAT ได้จริง

2. เพื่อศึกษาและประเมินประสิทธิภาพสถาปัตยกรรมแบบ Pull Model ที่ทำงานผ่าน Overlay Network (Tailscale)

3. เพื่อเปรียบเทียบความแตกต่างของสถาปัตยกรรม 2 รูปแบบ (Pull Model ผ่าน Tailscale กับ Push Model แบบดั้งเดิม) ใน 3 ด้านที่สำคัญ คือ ด้านประสิทธิภาพ ความหน่วงเวลาในการแจ้งเตือนและด้านความปลอดภัย

3. การทบทวนวรรณกรรมและทฤษฎีที่เกี่ยวข้อง (Literature Review)

3.1 แนวคิดและหลักการของระบบตรวจติดตาม (Monitoring Concepts)

การติดตามและเฝ้าระวัง (Monitoring) เป็นกระบวนการที่ครอบคลุมหลายมิติ ซึ่งเป็นปัจจัยหลักสำคัญของการเฝ้าติดตามระบบไม่ว่าจะเป็น การติดตามสถานะ CPU, RAM, Disk I/O เท่านั้นแต่ในบริบทของความมั่นคง ปลอดภัยไซเบอร์ สมัยใหม่ สกมช. (2025) ระบุว่าระบบเฝ้าระวังเป็นปราการด่านแรกในการระบุความผิดปกติ ก่อนที่ความเสียหายจะขยายตัวโดยเป้าหมายหลักจึงเป็นการเพิ่มความสามารถในการสังเกต (Observability) เพื่อลดระยะเวลาในการตรวจพบปัญหา (Mean Time to Detect- MTTD) และลดโอกาสเกิดการหยุดชะงักของระบบ (Downtime) ที่ส่งผลกระทบโดยตรงต่อความต่อเนื่องทางธุรกิจ (Business Continuity)

กระบวนการของระบบตรวจติดตามสามารถแบ่งออกเป็น 5 ขั้นตอน ได้แก่ 1. การรวบรวมข้อมูล (Data Collection) 2. การจัดเก็บข้อมูล (Data Storage) 3. การแสดงผล (Visualization) 4. การวิเคราะห์ (Analysis) และ 5. การแจ้งเตือนและการกู้คืน (Alerting and Recovering)

3.1.1 ประเภทและวิธีการรวบรวมข้อมูล (Data Collection Types and Methods)

การรวบรวมข้อมูลเป็นขั้นตอนแรกและสำคัญที่สุด โดยจำแนกวิธีการและประเภทของข้อมูลได้ดังนี้

1. White-Box Monitoring คือการติดตามระบบจากภายใน โดยอาศัยข้อมูลเชิงลึกจากตัวระบบเอง เช่น ค่า Metrics จากเซิร์ฟเวอร์ Log จากแอปพลิเคชัน หรือข้อมูลจาก Database วิธีนี้ช่วยให้เห็นภาพการทำงานของระบบภายใน และสามารถคาดการณ์ความผิดปกติของระบบได้ ซึ่ง Node Exporter ในงานวิจัยนี้เป็นเครื่องมือประเภท White-Box

2. Black-Box Monitoring คือการติดตามระบบจากภายนอก โดยไม่จำเป็นต้องรู้กระบวนการทำงานภายใน ส่วนใหญ่ใช้ตรวจสอบการเข้าถึงบริการยังสามารถเข้าถึงได้และตอบสนองได้เร็วเพียงพอ วิธีนี้ง่ายต่อการติดตั้ง และมีความปลอดภัยสูงแต่ให้ข้อมูลที่ค่อนข้างจำกัดในการแก้ปัญหาซอฟต์แวร์ที่ใช้ติดตามระบบจากภายนอก เช่น Blackbox Exporter

3. Pull-Based Model เป็นรูปแบบที่ระบบ Monitoring ร้องขอและเข้าไปดึงข้อมูล Metrics จากเครื่องเป้าหมาย เป็นระยะ ๆ โดย Prometheus ใช้หลักการนี้เป็นหลัก

4. Push-Based Model เป็นรูปแบบที่โหนดปลายทางเป็นฝ่ายส่งข้อมูลไปยังระบบ Monitoring กลาง ซึ่ง Prometheus รองรับการดำเนินงานในลักษณะนี้ผ่านส่วนประกอบที่เรียกว่า Pushgateway เหมาะสำหรับงานที่ทำงาน ในระยะสั้น ๆ (Short-Lived Jobs)

5. Metrics ข้อมูลเชิงปริมาณที่ใช้วัดประสิทธิภาพของระบบมี 2 ประเภทหลักคือ

1) Counter ค่าที่นับเพิ่มขึ้นเรื่อย ๆ เช่น จำนวนผู้เข้าชมเว็บไซต์

2) Gauge ค่า ณ จุดเวลาใดเวลาหนึ่ง เช่น อัตราการใช้งาน CPU

โดยข้อมูลประเภท Metrics ใช้พื้นที่จัดเก็บน้อยกว่าและง่ายต่อการนำมาคำนวณทางสถิติ

6. Logs ข้อมูลที่ระบุรายละเอียดเหตุการณ์ที่เกิดขึ้นในระบบ เป็นข้อมูลที่เจาะจงและแม่นยำ แต่ใช้ทรัพยากรในการจัดเก็บและประมวลผลสูงกว่า

3.1.2 การจัดเก็บและการแสดงผลข้อมูล (Data Storage and Visualization)

1. ข้อมูลที่รวบรวมมาจำเป็นต้องถูกจัดเก็บอย่างมีประสิทธิภาพเพื่อง่ายต่อการสืบค้นและวิเคราะห์ โดยเฉพาะข้อมูลแบบอนุกรมเวลา (Time Series Data) ซึ่งเป็นข้อมูลที่มีทั้งเวลาที่เก็บข้อมูลและค่าของข้อมูลนั้น ซึ่ง Prometheus ได้ออกแบบฐานข้อมูลนี้ไว้สำหรับการแสดงผลในระบบที่เรียกว่า Time Series Database (TSDB)

2. แดชบอร์ด (Dashboard) สื่อกลางหลักในการนำเสนอข้อมูล แดชบอร์ดที่ดีควรมีวัตถุประสงค์ที่ชัดเจน และแสดงข้อมูลที่จำเป็นทั้งหมดในหน้าเดียวเพื่อง่ายต่อการมอนิเตอร์ ซึ่งงานวิจัยนี้ได้เลือกใช้ Grafana เป็นเครื่องมือในการสร้างแดชบอร์ดแสดงผลข้อมูลจาก Prometheus เนื่องจากมีความยืดหยุ่นและสวยงามกว่า UI พื้นฐานของ Prometheus เอง

3.1.3 การวิเคราะห์เชิงความมั่นคงปลอดภัย (Security Analysis)

จากการรวบรวมข้อมูล จัดเก็บข้อมูล และการแสดงผลต่อมาก็ถึงการวิเคราะห์ข้อมูลที่ได้รับมาว่าเกิดข้อสงสัยหรือผิดปกติของระบบไหม ซึ่งตัวเลขที่ได้เก็บรวบรวมมา แสดงจะเป็นตัวบ่งบอกถึงปัญหาที่จะเกิดและสามารถคาดการณ์ได้

3.1.4 การแจ้งเตือนและการกู้คืน (Alerting and Recovery)

การแจ้งเตือนมีไว้สำหรับระบบไม่สามารถฟื้นตัว หรือจัดการปัญหาได้ด้วยตัวเอง และต้องได้รับการแก้ไขจากผู้ดูแล ซึ่งการแจ้งเตือนที่ควรแจ้งเตือนตามอาการที่ผู้ใช้รับรู้ได้ ไม่ใช่อิงตามค่าจาก Metrics เพียงอย่างเดียวทำให้ช่วยลดการแจ้งเตือนที่ไม่จำเป็นได้ Prometheus มี Alertmanager ที่ออกแบบมาเพื่อจัดการการแจ้งเตือนที่สามารถส่งต่อไปยังช่องทางอื่น ๆ ได้ง่าย ในงานวิจัยนี้ได้ให้ Alertmanager ส่งการแจ้งเตือนไปที่ Telegram

อย่างไรก็ตามในสภาพแวดล้อมที่มีข้อจำกัดการเข้าถึง (Restricted Networks) อย่าง CGNAT การใช้สถาปัตยกรรมแบบดั้งเดิมมักประสบปัญหาเรื่องความสามารถในการเข้าถึง (Reachability) ซึ่งนำไปสู่การเลือกใช้ Push-based Model ผ่าน Pushgateway แต่ผลงานวิจัยของ Hritcan & Balan (2024a) ชี้ให้เห็นว่าสถาปัตยกรรมแบบ Push อาจแลกมาด้วยความซับซ้อนในการจัดการความปลอดภัยและภาระงานของโหนดที่สูงขึ้น

3.2 สถาปัตยกรรมและองค์ประกอบของ Prometheus

Prometheus โอเพนซอร์สของการเฝ้าติดตามระบบที่ทรงพลังและเป็นที่ยอมรับเป็นอย่างมาก นอกจากการรวบรวมข้อมูลแบบเรียลไทม์ในแบบอนุกรมเวลา (Time-Series) เพื่อแสดงผลข้อมูลที่ได้มาแล้ว ยังสามารถกำหนดค่าให้แจ้งเตือนไปที่ช่องทางต่าง ๆ ได้อย่างง่ายดายและยังทำงานร่วมกับระบบอื่น ๆ ทั้งบน Cloud และ On-Premise ได้อย่างหลากหลายอีกด้วย ในหัวข้อนี้จะพูดถึงและทำความรู้จักกับ Prometheus ให้มากขึ้น เพื่อที่เข้าใจองค์ประกอบและการทำงานของ Prometheus

3.2.1 หลักการทำงานของ Prometheus

Prometheus ทางผู้พัฒนาได้พัฒนาให้ทำงานในด้าน Monitoring ได้อย่างมีประสิทธิภาพด้วยการออกแบบโปรแกรมให้มีการเก็บรวบรวมข้อมูลในรูปของ Metrics แบบ Time-Series Data เช่น การเก็บ Metrics Information โดยตัว Metric นั้นจะมีการระบุ Timestamp ของ Record ข้อมูลไว้ด้วย ซึ่งมี Option ในการระบุแบบ Key-Value Pairs โดยเรียกว่า Label ตัว Data ให้กับ Metric ของ Prometheus (Prometheus, n.d.)

หลักการทำงานที่สำคัญของ Prometheus คือการทำงานในรูปแบบ Pull-Based Model เป็นการที่ Prometheus Server เข้าไปดึงข้อมูล (Scraping) จากเครื่องเป้าหมายเป็นระยะ ๆ ตามกำหนดเวลาของไฟล์ Config ที่ได้ตั้งค่าเอาไว้ไฟล์ Config ที่ว่าเป็นไฟล์ YAML Format โดย Prometheus แบ่งหลักการทำงานหลัก ๆ ได้ดังนี้

1. การเก็บข้อมูล (Data Collection) หัวใจหลักของ Prometheus คือการ Scraping ข้อมูล Metrics จากเครื่องเป้าหมายที่กำหนดไว้โดยจะส่ง HTTP Request ไปยัง Endpoint ที่ชื่อว่า /metrics แต่ละ Target ตามเวลาที่ตั้งไว้

1.1 Targets คือ Service หรือ Application ที่เราต้องการจะ Monitor ซึ่งจะต้องมีส่วนที่เปิด Endpoint /metrics เพื่อให้ Prometheus เข้ามาดึงข้อมูลได้

1.2 Exporters กรณีที่บางระบบ เช่น Database, Hardware หรือ Message Queue) ไม่มี Endpoint /metrics มาให้เราสามารถใส่โปรแกรมตัวกลางที่เรียกว่า Exporter ได้ ซึ่ง Exporter จะทำหน้าที่ดึงข้อมูลจากระบบนั้น ๆ แล้วแปลงให้อยู่ในรูปแบบที่ Prometheus เข้าใจได้ก่อนจะเปิดเป็น /metrics Endpoint ให้ Prometheus เข้ามาดึงข้อมูลไปอีกที แม้สถาปัตยกรรมนี้จะช่วยให้เซิร์ฟเวอร์ควบคุมอัตราการดึงข้อมูลได้เอง (Load Control) แต่ในงานวิจัยที่ผ่านมาพบว่าโครงสร้างแบบนี้จะล้มเหลวทันทีเมื่อโหนดปลายทางอยู่หลังระบบ NAT หรือ Firewall ที่ไม่อนุญาตให้มีการเชื่อมต่อขาเข้า (Inbound Connection) ซึ่งเป็นประเด็นหลักที่งานวิจัยนี้มุ่งเน้นแก้ไข

สถาปัตยกรรมแบบ Pull-Based Model มีข้อดีคือ Prometheus จะเป็นศูนย์กลางที่รู้ว่าต้องเก็บข้อมูลจาก ที่ใดบ้างและสามารถตรวจสอบได้ทันทีหากเป้าหมายไหนมีปัญหาและไม่สามารถดึงข้อมูลได้

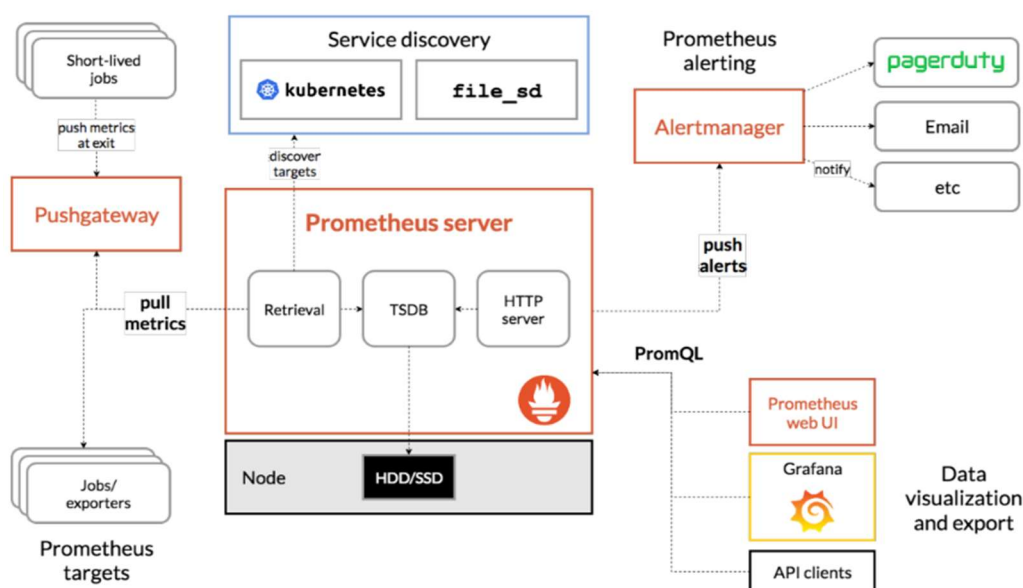


Figure 1. Prometheus Architecture

Source: Prometheus (n.d.b)

2. การค้นหาเป้าหมาย (Service Discovery) หากมีสภาพแวดล้อมที่ซับซ้อนอย่าง Microservices หรือ Kubernetes การกำหนด Targets แบบคงที่ (Static Configuration) อาจจะไม่สะดวกเพราะ Service ต่าง ๆ อาจจะมีการสร้าง ลบ หรือย้ายที่อยู่ตลอดเวลา Prometheus จึงมีความสามารถเรียกว่า Service Discovery เพื่อค้นหา Targets ที่จะทำการ Scraping ได้อย่างอัตโนมัติ เช่น

2.1 Kubernetes Service Discovery สามารถค้นหา Pods และ Service ทั้งหมดใน Cluster ของ Kubernetes ได้เอง

2.2 Consul Service Discovery ใช้กับระบบที่ลงทะเบียน Service ไว้กับ Consul

2.3 File-Based Service Discovery อ่านข้อมูล Targets จากไฟล์ JSON หรือ YAML ซึ่งสามารถอัปเดตได้ตลอดเวลา

3. การจัดเก็บข้อมูล (Storage) ข้อมูล Metrics ที่ถูกดึงมาจะถูกจัดเก็บลงในฐานข้อมูลที่ออกแบบมาเพื่อข้อมูลแบบ Time Series โดยเฉพาะเรียกว่า Time Series Database (TSDB)

3.1 Data Model ข้อมูลแต่ละชุดจะถูกระบุด้วย

1) Metric Name ชื่อของค่าชี้วัด เช่น http_requests_total (จำนวน Request ทั้งหมด)

2) Label Key-Value Pairs ที่ใช้ระบุคุณลักษณะของ Metric นั้น ๆ เช่น method="POST", endpoint="/api/users" ทำให้เราสามารถกรองและจัดกลุ่มข้อมูลได้อย่างยืดหยุ่นการเปลี่ยนแปลงค่าของ Label ใด ๆ จะถือว่าเป็น Time Series ใหม่ทันที

3.2 Sample ข้อมูลแต่ละจุดใน Time Series ประกอบด้วย

1) float64 value ค่าของ Metric

2) millisecond-precision timestamp เวลาที่เก็บข้อมูลนั้น TSDB ของ Prometheus ถูกปรับแต่งมาให้จัดเก็บและสืบค้นข้อมูลประเภทนี้ได้อย่างรวดเร็วและมีประสิทธิภาพสูง

4. การสืบค้นข้อมูล (Querying) ด้วย PromQL เมื่อเก็บข้อมูลได้แล้ว เราสามารถสืบค้นและนำข้อมูลมาวิเคราะห์ได้โดยใช้ภาษาที่มีชื่อว่า PromQL (Prometheus Query Language) ซึ่งเป็นภาษาที่ถูกออกแบบมาเพื่อทำงานกับ Time Series Data โดยเฉพาะเราสามารถใช PromQL ดังนี้

4.1 เลือก (Select) Time Series ที่ต้องการโดยอ้างอิงจาก Metric name และ Labels

4.2 กรอง (Filter) ข้อมูลตามเงื่อนไขต่าง ๆ

4.3 รวม (Aggregate) ข้อมูล เช่น หาผลรวม (sum), ค่าเฉลี่ย (avg), ค่าสูงสุด (max)

4.4 คำนวณ อัตราการเปลี่ยนแปลง (rate), การคาดการณ์ (predict) และฟังก์ชันอื่น ๆ

5. การแจ้งเตือน (Alerting) นอกจากนั้น Prometheus ยังสามารถกำหนดเงื่อนไขการแจ้งเตือน (Alerting Rules) ได้ด้วย โดยการเขียน Rules ด้วย PromQL

5.1 Prometheus Server จะทำการประมวลผล Rules ที่เรากำหนดไว้ในช่วงเวลาที่กำหนด

5.2 เมื่อเงื่อนไขตรงกับ Rules ที่กำหนด Prometheus จะส่งแจ้งเตือนไปยังอีกส่วนประกอบที่เรียกว่า Alertmanager ซึ่งทำหน้าที่รับ Alert จาก Prometheus Server แล้วจัดการสิ่งต่าง ๆ ดังต่อไปนี้

1) Deduplication รวมการแจ้งเตือนที่ซ้ำซ้อนกันเป็นอันเดียว

2) Grouping จัดกลุ่มการแจ้งเตือนที่เกี่ยวข้องกัน เช่น ปัญหาที่เกิดจาก Cluster เดียวกัน

3) Routing ส่งการแจ้งเตือนไปยังช่องทางที่ต้องการ เช่น E-Mail, Slack, Telegram

4) Silencing & Inhibition สามารถสั่งปิดการแจ้งเตือนชั่วคราว หรือจัดลำดับการแจ้งเตือนที่สำคัญให้แจ้งเตือนก่อนได้

6. การแสดงผล (Visualization) Prometheus มี Web UI สำหรับ Query และดูกราฟแบบง่าย ๆ แต่โดยทั่วไปแล้วนิยมใช้ร่วมกับ Dashboard ตัวอื่น ๆ โดยเฉพาะ Grafana ซึ่งสามารถเชื่อมต่อกับ Prometheus เป็น Data Source เพื่อสร้าง Dashboard ที่สวยงามและตอบโต้ได้

3.3 ข้อจำกัดของระบบเครือข่ายแบบ Carrier-Grade NAT (CGNAT)

NAT (Network Address Translation) คือเทคโนโลยีที่ใช้แปลง IP Address ของเครือข่ายภายใน (Private IP) ให้เป็น IP Address ของเครือข่ายภายนอก (Public IP) เพื่อให้คอมพิวเตอร์และอุปกรณ์ในเครือข่ายสามารถเชื่อมต่อกับอินเทอร์เน็ตภายนอกได้ด้วย Public IP เพียงไอพีเดียว

NAT มีหลายรูปแบบการทำงาน โดยหลัก ๆ จะแบ่งได้ดังนี้

1. Static NAT เป็นการจับคู่ Public IP กับ Private IP หนึ่งต่อหนึ่งอย่างถาวรมักใช้กับเซิร์ฟเวอร์ที่ต้องการให้คน ภายนอกสามารถเข้าถึงได้ตลอดเวลา เช่น Web Server หรือ Mail Server เป็นต้น

2. Dynamic NAT เป็นการจับคู่ Private IP กับ Public IP แบบ หนึ่งต่อหนึ่งเช่นกัน แต่ Public IP จะถูกดึงมาจาก Pool ที่เตรียมไว้เมื่อมีการใช้งานก็จะถูกแจกจ่ายชั่วคราว เมื่อเลิกใช้งานก็จะถูกคืนกลับไปยัง Pool เป็น Public IP แบบชั่วคราว

3. Port Address Translation (PAT) หรือ NAT Overload เป็นรูปแบบของ NAT ที่นิยมมากที่สุดในปัจจุบันทั้งในบ้านและองค์กรทั่วไปเป็นรูปแบบการแปลง Private IP หลาย ๆ IP ให้ไปใช้ Public IP เพียงไอพีเดียว โดยจะใช้หมายเลขพอร์ต (Port Number) ที่แตกต่างกันในการแยกแยะข้อมูลชุดไหนเป็นของคอมพิวเตอร์เครื่องใดทำให้ประหยัด Public IP ได้มากที่สุด

Carrier-Grade NAT (CGNAT) คือการทำ NAT ในระดับผู้ให้บริการอินเทอร์เน็ต (ISP) หรือที่เรียกว่า Large-Scale NAT (LSN) ด้วยเหตุผลหลักเพราะ Public IPv4 ของทั่วโลกไม่เพียงพอต่อจำนวนผู้ใช้ การทำงานของ Carrier-Grade NAT (CGNAT) เป็นการที่ผู้ให้บริการอินเทอร์เน็ต (ISP) นำหลักการ PAT มาประยุกต์ใช้ใน Scale ขนาดใหญ่โดยให้ลูกค้าหลาย ๆ บ้านหรือหลาย ๆ องค์กรมาแชร์ Public IP Address เดียวกันจาก Pool ของ ISP ที่มีเตรียมไว้ ผลที่เกิดขึ้นของการทำงานแบบ CGNAT นี้แทนที่ Router ที่บ้านหรือองค์กรจะได้รับ Public IP โดยตรงกลับได้เป็น Private IP ที่อยู่ในเครือข่ายของ ISP จากนั้น IP นี้จะถูกนำไปทำ NAT อีกชั้นที่ฝั่ง ISP เพื่อแปลงเป็น Public IP จริง ๆ ทำให้เกิด NAT ซ้อนกัน 2 ชั้น

ด้วยเหตุนี้จึงเกิดเป็นข้อจำกัดและแนวทางการวิจัยของงานวิจัยนี้ เพราะการทำงานที่เป็นมาตรฐานของ Prometheus คือ Pull-based Model ทำงานเชื่อมต่อไปหาเครื่องเป้าหมายไม่ได้โดยตรง ซึ่งงานวิจัยนี้ได้แก้ไขปัญหาดังกล่าวการใช้งานเครือข่ายเสมือนซ้อนทับด้วย Tailscale

3.4 ทางออกเชิงสถาปัตยกรรม Overlay Network และ Zero Trust

จากข้อจำกัดของงานวิจัยที่มีจึงทำให้งานวิจัยนี้หาวิธีและศึกษาจากงานวิจัยที่เกี่ยวกับเรื่องดังกล่าว เพื่อหาทางออกให้งานวิจัยด้วยการนำสถาปัตยกรรมแบบ Overlay Network เข้ามาใช้ และ Zero Trust เพื่อให้สถาปัตยกรรมที่งานวิจัยนี้ได้พัฒนาออกมามีประสิทธิภาพใช้งานได้และมีความปลอดภัย

1. เครือข่ายส่วนตัวแบบซ้อนทับ (Overlay Network) คือเครือข่ายเสมือน (Virtual Network) ที่ถูกสร้างขึ้นซ้อนทับเครือข่ายแบบกายภาพ (Physical Network) ที่เชื่อมต่ออยู่เดิม กล่าวคือในโลกของคอมพิวเตอร์ Underlay Network ก็คืออุปกรณ์เน็ตเวิร์คจริง ๆ เช่น สายเคเบิล สวิตช์ เราเตอร์ ที่เชื่อมต่อกัน ส่วน Overlay Network คือ เครือข่ายเชิงตรรกะ (Logical Network) ที่สร้างขึ้นโดยใช้ซอฟต์แวร์ เพื่อเชื่อมต่ออุปกรณ์ต่าง ๆ เข้าด้วยกันเป็น เครือข่ายเสมือน โดยไม่จำเป็นต้องสนใจว่าอุปกรณ์เหล่านั้นจะเชื่อมต่อกันอย่างไรในทางกายภาพ ขอแค่ให้สามารถส่งข้อมูล หากันผ่านเครือข่าย Underlay ได้ก็พอ

2. WireGuard คือซอฟต์แวร์ VPN (Virtual Private Network) แบบโอเพนซอร์ส ถูกออกแบบมาอย่างเรียบง่าย รวดเร็วและมีความปลอดภัยสูง ซึ่งได้รับการยอมรับอย่างกว้างขวางและถูกรวมเข้าเป็นส่วนหนึ่งของ Linux Kernel เป็นการการันตีถึงประสิทธิภาพและความน่าเชื่อถือ

หลักการการทำงานของ WireGuard มีแนวคิดที่ต่างจาก VPN แบบดั้งเดิมอย่าง OpenVPN หรือ IPsec อย่างสิ้นเชิง โดยเน้นความเรียบง่ายและใช้ เทคโนโลยีสมัยใหม่ หลักการทำงานสามารถแบ่งออกเป็นข้อ ๆ ได้ดังนี้

1) Public Key Cryptography เป็นหัวใจหลัก แทนที่จะใช้ระบบ Username/Password หรือ Certificate ที่ซับซ้อน WireGuard ใช้วิธีการคล้ายกับ SSH (Secure Shell) ในการยืนยันตัวตน

1.1) WireGuard จะไม่เรียกว่า Client หรือ Server แต่จะเรียกว่า Peer ซึ่งเท่าเทียมกัน

1.2) แต่ละ Peer จะสร้าง Key Pair ของตัวเอง ซึ่งประกอบด้วย Private Key ญุณแจส่วนตัวที่ต้องเก็บไว้บน อุปกรณ์ของตัวเองและ Public Key ญุณแจที่ใช้แลกเปลี่ยนหรือแจกจ่ายให้กับ Peer อื่น ๆ

การเชื่อมต่อจะเกิดขึ้นก็ต่อเมื่อ Peer สองฝั่งแลกเปลี่ยน Public Key กันและตั้งค่าในไฟล์ configuration ว่า Public Key นี้ได้รับอนุญาตให้เชื่อมต่อเข้ามาได้

2) สร้างอุโมงค์ (Tunnel) ที่เรียบง่ายเมื่อ Peer ทั้งสองฝั่งมี Public Key ของอีกฝ่ายแล้ว WireGuard จะสร้าง อินเทอร์เฟซเครือข่ายเสมือน (Virtual Network Interface) ขึ้นมาบนอุปกรณ์ของแต่ละฝั่งข้อมูลทั้งหมดที่จะส่งผ่าน VPN จะถูกส่งเข้ามาที่อินเทอร์เฟซเสมือนนี้ก่อน

3) การห่อหุ้มข้อมูลและเข้ารหัสข้อมูล (Encapsulation) เมื่อข้อมูล (IP Packet) วิ่งเข้ามาในอุโมงค์ WireGuard จะทำการกระบวนการต่อไปนี้

3.1) เข้ารหัส (Encrypt) ใช้การเข้ารหัสที่รวดเร็วและทันสมัย เช่น ChaCha20Poly1305 เพื่อเข้ารหัสข้อมูล ทั้งหมด

3.2) ห่อหุ้ม (Encapsulate) นำข้อมูลที่เข้ารหัสแล้วไปใส่ไว้ใน UDP Packet เหตุที่เลือกใช้ UDP แทน TCP เพราะ UDP มีความซับซ้อนน้อยกว่าและเร็วกว่ามาก เหมาะสำหรับการส่งข้อมูลแบบ Real-Time อย่าง VPN

4) ส่งข้อมูลผ่านอินเทอร์เนต UDP Packet ที่ห่อหุ้มข้อมูลที่เข้ารหัสไว้ จะถูกส่งผ่านเครือข่ายอินเทอร์เนตปกติไปยัง IP Address และ Port ของ Peer ปลายทางที่ระบุไว้ในค่าคอนฟิก

5) ถอดรหัสและส่งต่อไปยังปลายทาง เมื่อ Peer ปลายทางได้รับ UDP Packet แล้ว

5.1) ตรวจสอบและถอดรหัส (Decrypt) ใช้ Private Key ของตัวเองและ Public Key ของผู้ส่งเพื่อถอดรหัสข้อมูลที่อยู่ข้างใน

5.2) ถอดข้อมูลที่ถูกรหัส (Decapsulate) เพื่อนำ IP Packet เดิมที่อยู่ข้างในออกมา

5.3) ส่งต่อ IP Packet นั้นต่อไปยังปลายทางที่แท้จริงในเครือข่ายภายใน

กระบวนการที่กล่าวไปนี้เกิดขึ้นอย่างรวดเร็วและใช้ทรัพยากรเครื่องน้อยมาก ทำให้ WireGuard มีประสิทธิภาพสูงและเป็นที่ยอมรับกว่า VPN แบบดั้งเดิม

3. Tailscale คือบริการที่สร้างเครือข่ายส่วนตัวเสมือน (Virtual Private Network) หรือที่เรียกว่า Overlay Network ขึ้นมาได้อย่างง่ายดาย เพื่อเชื่อมต่ออุปกรณ์ทุกเครื่อง เช่น คอมพิวเตอร์ โทรศัพท์ เซิร์ฟเวอร์ เข้าด้วยกันอย่าง

ปลอดภัย ไม่ว่าอุปกรณ์เหล่านี้จะอยู่ที่ไหนในโลกก็ตาม เปรียบเสมือนการสร้างอินเทอร์เน็ตส่วนตัวขึ้นมาให้ทุกเครื่องมองเห็นและสื่อสารกันได้โดยตรงเหมือนอยู่ในวง LAN เดียวกันโดยไม่ต้องตั้งค่า Firewall หรือ Router ที่ซับซ้อนเลย Hritcan & Balan (2024b) ได้นำ Tailscale มาใช้เพื่อเพิ่มความมั่นคงปลอดภัยในสภาพแวดล้อม IoT ได้สำเร็จ อย่างไรก็ตามการวิจัยดังกล่าวยังไม่ได้มุ่งเน้นไปที่การเปรียบเทียบประสิทธิภาพเชิงประจักษ์ (Empirical Comparison) ระหว่าง Pull Model ผ่าน ZTNA และ Push Model แบบดั้งเดิม โดยเฉพาะในมิติของความหน่วง (Latency) และภาระงานของระบบ (System Overhead) ภายใต้เครือข่าย CGNAT ที่มีการรับส่งข้อมูล Metrics อย่างต่อเนื่อง

4. กรอบแนวคิดงานวิจัย (Conceptual Framework)

4.1 การออกแบบสถาปัตยกรรมโดยรวม (Overall System Architecture)

ผู้วิจัยได้จำลองสถาปัตยกรรมออกเป็น 2 รูปแบบหลัก เพื่อทำการทดสอบเปรียบเทียบ ได้แก่

1. สถาปัตยกรรมแบบ Pull Model ผ่าน Tailscale ใช้ Prometheus Server บน Cloud (Azure) เชื่อมต่อกับเครื่อง Client ผ่านอุโมงค์ VPN[2, 3] ของ Tailscale (WireGuard Protocol) เพื่อดึงข้อมูลหรือ Scrape โดยตรงเหมือนอยู่ในวง LAN เดียวกัน

2. สถาปัตยกรรมแบบ Push Model เครื่อง Client จะรัน Script เพื่อส่งหรือ Push ข้อมูลออกไปยัง Pushgateway ที่ตั้งอยู่บน Cloud ผ่านอินเทอร์เน็ตสาธารณะซึ่งจำเป็นต้องมีการเปิด Inbound Port ที่ฝั่ง Cloud Server เพื่อรับข้อมูล

4.2 การเตรียมสภาพแวดล้อมการทดลอง

การทดสอบดำเนินการโดยใช้เครื่องมือและสภาพแวดล้อมดังนี้

1. Server Side ติดตั้งบน Microsoft Azure Virtual Machine รันระบบปฏิบัติการ Ubuntu Server 24.04 LTS ติดตั้ง Prometheus, Grafana, Alertmanager และ Pushgateway

2. Client Side จำลองโหนดปลายทางด้วย Virtual Machine ที่ใช้ vRouter (pfSense) เพื่อจำลองสภาพแวดล้อมแบบ CGNAT

3. Software อื่น ๆ ที่ใช้ในการทดลอง ใช้ Node Exporter ในการเก็บข้อมูล Metric และใช้ Tailscale ในการสร้าง Overlay Network

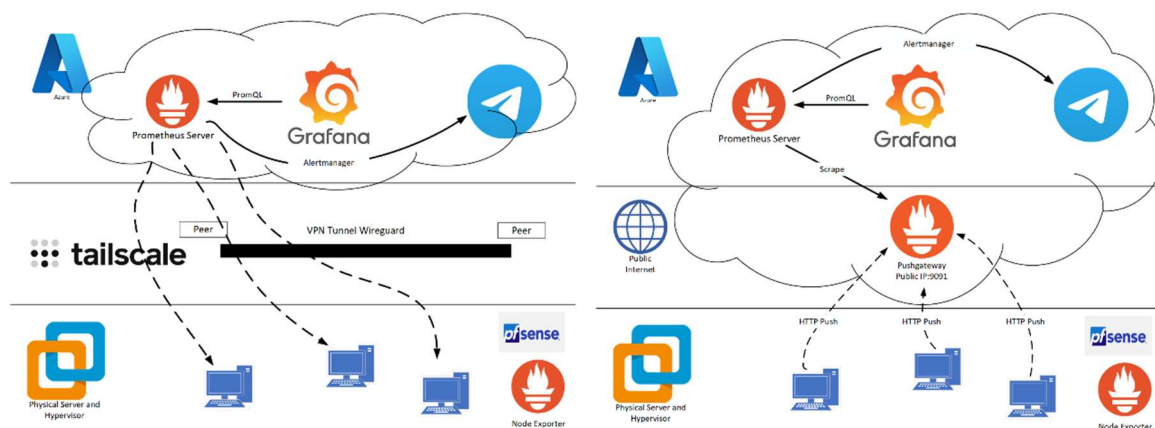


Figure 1. Architecture Comparison Pull Model via Tailscale vs. Push Model

ข้อกำหนดด้านฮาร์ดแวร์และโครงสร้างพื้นฐาน ดัง Table 1. และ ข้อกำหนดซอฟต์แวร์ ดัง Table 2.

Table 1. Hardware and Infrastructure Details

VM Name	IP	Tailscale IP	Description
Main-Server	192.168.1.140	100.87.179.19	Intel Xeon E5 2690, Ram128GB, Storage 1TB
pfSense	192.168.10.1	-	1vCPU, Ram4GB, Storage 50GB
Monitoring-Server Azure	20.6.xx.1xx	100.88.117.65	1vCPU, Ram1GB, Storage 30GB
Tailscale1	192.168.10.113	100.127.147.51	1vCPU, Ram2GB, Storage 20GB
Tailscale2	192.168.10.114	100.116.144.14	1vCPU, Ram2GB, Storage 20GB
Tailscale3	192.168.10.115	100.100.234.23	1vCPU, Ram2GB, Storage 20GB
Pushgateway1	192.168.10.110	-	1vCPU, Ram2GB, Storage 20GB
Pushgateway2	192.168.10.111	-	1vCPU, Ram2GB, Storage 20GB
Pushgateway3	192.168.10.112	-	1vCPU, Ram2GB, Storage 20GB

Table 2. Software Details

Software	Version	Description
Ubuntu Server	22.04 LTS	Primary Operating System (ระบบปฏิบัติการหลัก)
Prometheus	2.53.1	Metrics Collection and Storage System (ระบบรวบรวมและจัดเก็บเมตริก)
Grafana	12.0.2	Data Visualization System (ระบบแสดงผลข้อมูล)
Node Exporter	1.9.1	Metrics Collection Agent (Agent รวบรวมเมตริก)
Pushgateway	1.11.1	Gateway for Receiving Push-Based Data (Gateway สำหรับรับข้อมูลแบบ Push)
Tailscale	1.88.3	Overlay Network Creation (สร้างเครือข่าย Overlay Network)
pfSense	2.7.0	Virtual Router and Firewall (Router และ Firewall เสมือน)

5. วิธีดำเนินงานวิจัย (Research Methodology)

5.1 การเก็บรวบรวมและวิเคราะห์ข้อมูล

การวัดผลแบ่งออกเป็น 3 มิติของข้อมูล ได้แก่

1. ภาระของระบบ (System Overhead) วัดการใช้ทรัพยากรเครื่อง CPU, RAM, DISK I/O และ NETWORK I/O ของเครื่อง Client โดยใช้คำสั่ง SAR (System Activity Report) เพื่อบันทึกผล ในสถานะ Baseline ของเครื่อง เซิร์ฟเวอร์และโหนดปลายทางที่ทำงานในสถานะปกติหลังจากตั้งค่าและติดตั้ง Node Exporter เรียบร้อยและปิดการใช้งาน Tailscale และ Push Script ไว้เพื่อเปรียบเทียบกับ ขณะใช้งาน Pull และ Push Model โดยเก็บข้อมูลต่อเนื่องทั้งหมด 3 สถานะ สถานะละ 4 ชั่วโมง โดยเก็บทั้งเครื่องเซิร์ฟเวอร์และโหนดปลายทางทั้งหมดรวมเป็น 24 ชั่วโมง

2. ความหน่วงเวลาของการแจ้งเตือน (Alerting Latency) โดยการจับเวลาตั้งแต่ระบบเริ่มจำลองสถานการณ์ ผิด ปกติ (T1) จนถึงเวลาที่ Alertmanager ตรวจพบ (T2) และเวลาที่ Telegram ได้รับการแจ้งเตือน (T3) เพื่อหา Total Latency สำหรับคำสั่งที่ใช้จำลองสถานการณ์ในการทดสอบการแจ้งเตือนโดย ใช้คำสั่ง stress-ng เพื่อทดสอบด้าน CPU กับ RAM ส่วน Disk จะจำลองโดยการสร้างไฟล์ขนาดใหญ่ และ Network จะใช้คำสั่ง iperf3 ในการทดสอบ โดยการ

ทดสอบความหน่วงเวลาของการแจ้งเตือนนี้ไม่ได้ปรับค่าใด ๆ เพิ่มเติม (Default Configuration) ทำให้เวลาที่ Notification Latency (T3) ตรวจพบจะเป็นเวลาที่เข้าเงื่อนไขของ Alert ที่ 3 นาทีหรือ 186 วินาที เมื่อได้ผลลัพธ์ ทั้งหมดแล้วจะนำมาหาค่าเฉลี่ย (Average) และค่าความเบี่ยงเบนมาตรฐาน (Standard Deviation) การแจ้งเตือน จะจำลองสถานการณ์ทั้งหมด 4 ด้านคือ CPU, RAM, DISK และ NETWORK แบ่งออกเป็นจำลองฝั่ง Pull Model ด้านละ 10 ครั้งและ Push Model ด้านละ 10 ครั้งรวมเป็นทั้งหมด 80 ครั้งการแจ้งเตือน

3. ประเมินด้านความปลอดภัยทางไซเบอร์ วิเคราะห์ประเมินช่องโหว่ด้านความปลอดภัยและความน่าเชื่อถือของการส่งข้อมูลโดยด้านความปลอดภัยของระบบจะวิเคราะห์ไปถึงกระประยุคที่ใช้ Zero Trust Network Access (ZTNA) การยืนยันตัวตน (Authentication) การเข้ารหัสข้อมูล (Data Encryption) ความเสี่ยงจากการพึ่งพา(Third-Party Risk) และความเสี่ยงจากการลุกลามภายใน (Lateral Movement Risk) เป็นต้น โดยใช้เว็บไซต์ที่ให้บริการสแกนพอร์ต (whatismyip.com) และวิเคราะห์เชิงคุณภาพ Qualitative Analysis ตามทฤษฎี (ZTNA)

5.2 Process กระบวนการทำงานของ 2 สถาปัตยกรรม

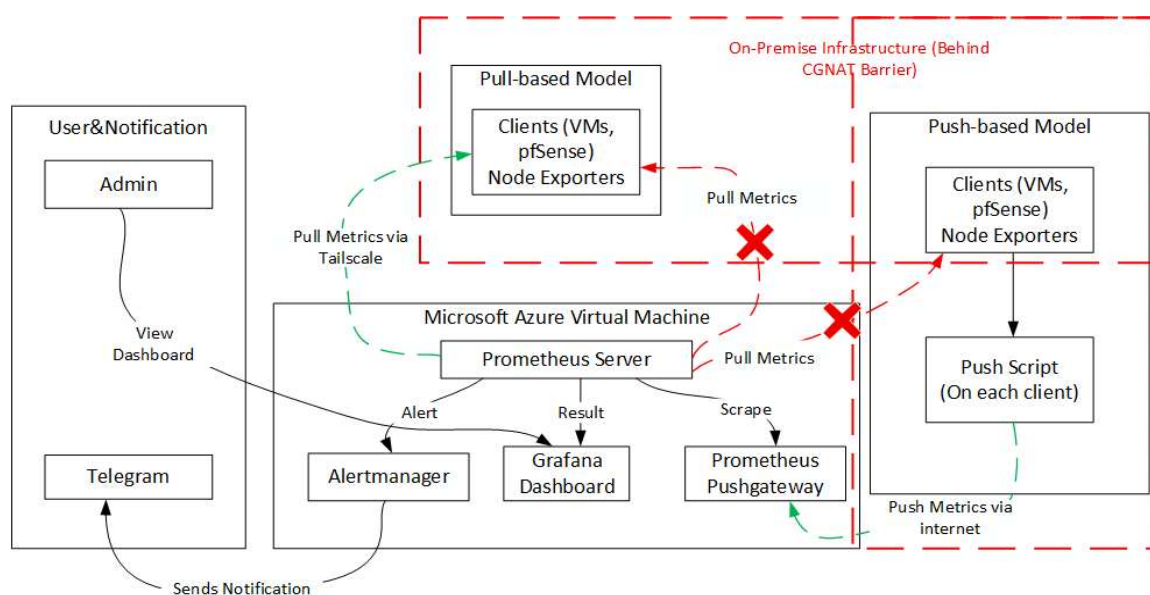


Figure 3. Data Flow Diagram

1. กระบวนการทำงานของสถาปัตยกรรมแบบดึงข้อมูล (Pull-based Model)

สถาปัตยกรรมนี้มีลักษณะการทำงานที่มี Prometheus Server เป็นศูนย์กลางคอยควบคุมและเริ่มการทำงานทั้งหมด ดังขั้นตอนต่อไปนี้

1.1 กระบวนการเริ่มต้น Prometheus Server ที่ทำงานอยู่บน Azure ที่ได้ทำการ Config เอาไว้ว่าเมื่อถึงเวลาที่กำหนดไว้ (Scrape Interval) Prometheus จะส่งคำขอข้อมูล (Pull Request) ไปยังเครื่องเป้าหมายที่ต้องการติดตาม

1.2 การเชื่อมต่อ คำขอดังกล่าวจะถูกส่งผ่านเครือข่ายส่วนตัวของ Tailscale ที่ทำหน้าที่เป็นอุโมงค์ ในการสื่อสารที่ปลอดภัยและเข้ารหัสข้อมูล (Encrypted Tunnel) ทำให้สามารถเชื่อมต่อกับเครื่องเป้าหมายได้โดยตรง แม้ว่าเครื่องนั้นจะทำงานอยู่หลัง pfSense หรือข้อจำกัดของ CGNAT ก็ตาม

1.3 การตอบสนอง เมื่อ Node Exporter บนเครื่องเป้าหมายได้รับคำขอ จะทำการรวบรวมข้อมูลเมตริกสถานะล่าสุดของเครื่อง เช่น CPU Ram แล้วส่งผลกลับ (Result) ไปยัง Prometheus Server ผ่านอุโมงค์ของ Tailscale

1.4 การประมวลผลและการแสดงผล เมื่อ Prometheus Server ได้รับข้อมูลเมตริกแล้ว จะทำการจัดเก็บลงฐานข้อมูลอนุกรมเวลา (Time-Series Database) และ Grafana Dashboard จะดึงข้อมูล (PromQL) จาก Prometheus Server นำไปแสดงผลเป็นกราฟและแผนภูมิที่ Admin สามารถเข้ามาตรวจสอบได้

1.5 การแจ้งเตือน หากข้อมูลเมตริกที่ได้รับมีค่าผิดปกติหรือไม่เป็นไปตามกฎที่ตั้งไว้ Alertmanager จะทำหน้าที่ส่งการแจ้งเตือน (Send Notification) ไปยังช่องทางที่กำหนดไว้ เช่น Telegram เพื่อให้ Admin รับทราบถึงปัญหา

2. กระบวนการทำงานของสถาปัตยกรรมแบบส่งข้อมูล (Push-Based Model)

สถาปัตยกรรมนี้มีลักษณะการทำงานที่เริ่มจากเครื่องเป้าหมาย โดยแต่ละเครื่องเป้าหมายมีหน้าที่ในการส่งข้อมูล ของตนเองไปที่ตัวกลางที่กำหนด Pushgateway ดั้งชั้นตอนต่อไป

2.1 กระบวนการเริ่มต้น เครื่องเป้าหมายแต่ละเครื่อง เมื่อถึงรอบเวลากำหนดของ Push Script ที่สร้างไว้จะทำงานอัตโนมัติ

2.2 การรวบรวมและส่งข้อมูล Push Script จะทำการดึงข้อมูลเมตริกล่าสุดจาก Node Exporter ที่ทำงานอยู่ บนเครื่องของตนเอง จากนั้นจะทำการ Push เมตริก ข้อมูลนั้นผ่านเครือข่ายอินเทอร์เน็ต ไปยัง Prometheus Pushgateway

2.3 ตัวกลางการรับข้อมูล Prometheus Pushgateway ซึ่งทำงานอยู่บน Azure จะทำหน้าที่เป็นตัวกลางในการรับข้อมูลเมตริกล่าสุดที่เครื่องเป้าหมายแต่ละเครื่องส่งมา

2.4 การรวบรวมข้อมูล Prometheus Server จะไม่เชื่อมต่อกับเครื่องเป้าหมายโดยตรง แต่จะทำการดึงข้อมูล (Scrape) จาก Pushgateway ซึ่งอยู่ในเครือข่ายเดียวกัน เพื่อรวบรวมข้อมูลเมตริกทั้งหมดมาไว้ที่ตัวกลาง

2.5 การประมวลผล แสดงผล และการแจ้งเตือนเมื่อ Prometheus Server ได้รับข้อมูลจาก Pushgateway กระบวนการที่เหลือก็จะเหมือนกับสถาปัตยกรรมแบบ Pull-Based Model ทุกประการ คือ ข้อมูลจะถูกจัดเก็บใน (TSDB) และนำไปแสดงผลบน Grafana Dashboard หากเกิดปัญหา Alertmanager ก็จะส่งแจ้งเตือนไปยัง Telegram

6. ผลการวิจัย (Results)

6.1 ผลการพัฒนาและออกแบบระบบเฝ้าระวังประสิทธิภาพและความปลอดภัยให้สามารถทำงานผ่านเครือข่ายที่มีข้อจำกัด CGNAT ได้จริง

จากการตั้งค่าระบบพบว่า Prometheus Server สามารถรวบรวมข้อมูลเมตริกจากเครื่องเป้าหมายทั้งสองรูปแบบได้สำเร็จและสามารถนำข้อมูลดังกล่าวมาแสดงผลบน Grafana Dashboard ได้โดยรายละเอียดขั้นพื้นฐานของแต่ละ เครื่องเป้าหมาย Tailscale ที่ได้มอนิเตอร์ไว้ โดยสามารถเลือกดูรายละเอียดลึก ๆ ของแต่ละเครื่องได้ด้วยการเลือกชื่อเครื่องหรือ Instance เป้าหมาย

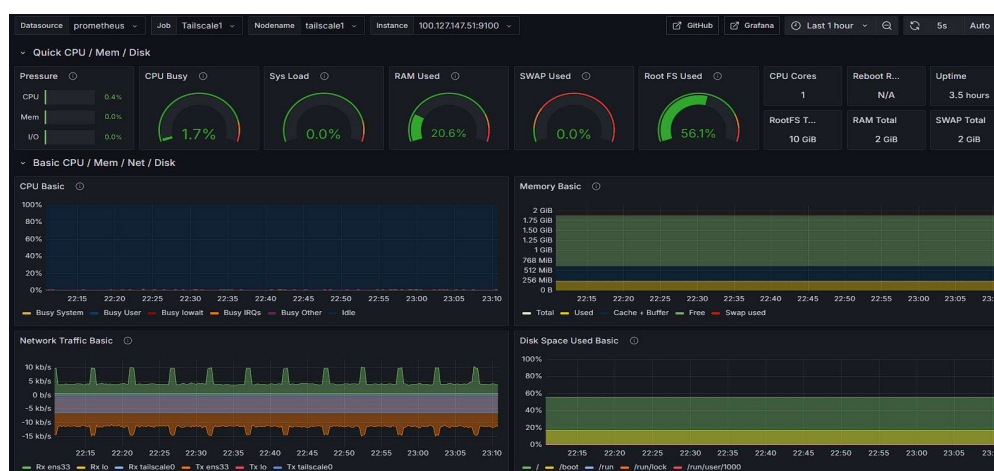


Figure 4. CPU, RAM, Memory, Network Traffic, and Disk Space Details for Pushgateway Target

6.2 ผลการศึกษาและประเมินประสิทธิภาพสถาปัตยกรรมแบบ Pull Model ที่ทำงานผ่าน Overlay Network (Tailscale)

ผลการวิเคราะห์ภาระของระบบ (System Overhead) จากการเก็บข้อมูลด้วยเครื่องมือ SAR พบว่า สถาปัตยกรรม ทั้ง 2 แบบสร้างภาระให้กับเครื่องแตกต่างกันอย่างชัดเจน โดย Push Model ใช้ทรัพยากรน้อยกว่ามาก เนื่องจากการ รัน Script ส่งข้อมูลเป็นรอบ ๆ ไม่มี Background Service ที่ทำงานตลอดเวลาเหมือน Tailscale ของฝั่ง Pull Model ผลเปรียบเทียบทรัพยากรตาม Table 3.

Table 3. Comparison of Client Net Overhead relative to Baseline

ทรัพยากร(Metric)	Pull Model (Overhead)	Push Model (Overhead)
CPU	2.31%	0.48%
RAM	9,348 KB	5,031 KB
NET (Transmit)	36.54 KB/s	2.64 KB/s
DISK (Read)	-17.96 KB/s	45.58 KB/s
DISK (Write)	-113.70 KB/s	61.11 KB/s

จากผลการทดลองเห็นว่า Push Model มีภาระงานที่เพิ่มขึ้นจาก Baseline น้อยกว่า Pull Model ทั้งด้าน CPU, RAM และ Network (Transmit) ส่วนสำหรับ DISK เนื่องจากการใช้คำสั่ง sar -A -o baseline.dat 1 14400 เครื่องเลย บันทึกราคาลง Disk ทั้งหมดทุกวินาที จึงทำให้ค่า Disk มีความผันผวนสูงอีกข้อสังเกต Network (Transmit) Pull Model มี ภาระด้านนี้สูงกว่า Push Model อย่างมีนัยสำคัญ

Table 4. Comparison of Server Net Overhead

Resources (Metric)	Pull Model (Overhead)	Push Model (Overhead)
CPU	3.11%	3.12%
RAM	436,230 KB	424,404 KB
NET (Transmit)	50.11 KB/s	67.17 KB/s
NET (Receive)	6.38 KB/s	28.89 KB/s

ข้อสังเกตภาระงาน CPU RAM บน Server ใกล้เคียงกันแต่พฤติกรรม Network ตรงข้ามกัน โดย ทาง Pull Model มีภาระของ Network ด้าน TX มากกว่าส่วน Push Model ภาระของ Network ด้าน RX มากกว่า

6.3 ผลการเปรียบเทียบความแตกต่างของสถาปัตยกรรม 2 รูปแบบ (Pull Model ผ่าน Tailscale กับ Push Model แบบดั้งเดิม) ใน 3 ด้านที่สำคัญ คือ ด้านประสิทธิภาพ ความหน่วงเวลาในการแจ้งเตือนและด้านความปลอดภัย

ผลการทดสอบความหน่วงเวลาของการแจ้งเตือน (Alerting Latency) ในด้านความเร็วในการตอบสนอง Pull Model ทำได้ดีกว่าอย่างเห็นได้ชัดในทุกการทดสอบ เช่น ในกรณีทดสอบ Disk Usage พบว่า Pull Model ใช้เวลาแจ้ง เตือน รวมเฉลี่ย 199.7 วินาที ขณะที่ Push Model ใช้เวลา 217.5 วินาที สาเหตุหลัก ๆ มาจาก Push Model ต้องส่ง ข้อมูลไปหา ตัวกลางก่อนคือ Pushgateway ก่อนทำให้เกิดความหน่วงเวลาสะสมและรอบการทำงานของ Script ผลการวิจัย ดัง Table 5.

งานวิจัยนี้มีค่า Notification Latency (T3-T2) ที่คงที่เท่ากันจากการทดสอบรวมทั้งหมด 4 ด้านจำนวน 80 ครั้ง แยกทั้ง 2 สถาปัตยกรรมที่ 186 วินาทีหรือ 3 นาที 6 วินาทีในทุกการทดสอบ เนื่องมาจากค่าเริ่มต้น (Default Configuration) ของไฟล์ Prometheus ได้กำหนดไว้ที่ค่านี้

ผลการวิเคราะห์ด้านความปลอดภัยทางไซเบอร์ ดัง Table 6.

Table 5. Summary of Notification Latency Test Results

Architecture	Resource Type	Detection Latency Mean(s)	Notification Latency Mean(s)	Total Latency Mean(s)	Total Latency (StdDev)
Pull Model	CPU	459.5	186.0	645.5	28.27
	RAM	39.7	186.0	225.7	50.61
	DISK	13.7	186.0	199.7	4.72
	NETWORK	18.4	186.0	204.4	3.75
Push Model	CPU	483.1	186.0	669.1	29.28
	RAM	67.3	186.0	253.3	43.52
	DISK	31.5	186.0	217.5	5.44
	NETWORK	83.7	186.0	269.7	47.68

Table 6. Cybersecurity and Zero Trust Architecture Analysis

Evaluation Criteria	Pull Model	Push Model
1. Attack Surface (ช่องทางการโจมตี)	Low (Strength) No ports are exposed to the public.	High (Weakness) Port 9091/TCP must be opened to receive data.
2. Data Encryption (การเข้ารหัสข้อมูล)	Yes (Automatic) End-to-end encryption using WireGuard is enabled immediately upon installation.	Supported (Configuration Required) The default connection uses plain HTTP. TLS/SSL can be enabled by creating a certificate and manually configuring the system.
3. Authentication (การยืนยันตัวตน)	Yes (Enforced) Users must authenticate through an Identity Provider before accessing the network.	Supported (Configuration Required) By default, the system accepts data from all sources. Basic Authentication or a Reverse Proxy can be configured to enforce password-based access control.
4. Third-Party Risk (ความเสี่ยงจากการพึ่งพา)	High (Weakness) The system depends on Tailscale's Coordination Server. If it becomes unavailable, nodes cannot be paired.	Low (Strength) The system is self-hosted and operates independently without relying on external servers.
5. Key&Cert Lifecycle (การบำรุงรักษา)	Platform-Defined Tailscale enforces a key-expiry policy for security. If the key is not renewed, the connection will be terminated immediately.	Admin-Defined Certificates have expiration dates, but administrators can renew them manually or configure automatic renewal.

Table 6. Cybersecurity and Zero Trust Architecture Analysis (Cont.)

Evaluation Criteria	Pull Model	Push Model
6. Lateral Movement Risk (ความเสี่ยงจากการ ลุกลามภายใน)	Medium (Weakness) If a client is compromised, it may be used as a pathway to access the server, especially if Tailscale ACLs are not properly configured.	Low (Strength) Communication is one-way. The client only sends data through the designated channel and cannot access the server through other channels.
7. Network Access (ZTNA) (ความสอดคล้องกับ Zero Trust)	Compliant Tailscale's network connections follow the ZTNA principle of "Never Trust, Always Verify."	Supported (Configuration Required) The default architecture is perimeter-based, but it can be upgraded to ZTNA by using mTLS (Mutual TLS) or an Authentication Proxy to verify every request. However, this approach increases management complexity.

จากการวิเคราะห์ผลการทดสอบเบื้องต้นนี้จะเห็นได้ว่า Pull Model ผ่าน Tailscale มีความเสถียรของข้อมูลที่น่าเชื่อถือกว่า และมีความปลอดภัยทางไซเบอร์มากกว่าในกรณีเริ่มต้นทั้ง 2 สถาปัตยกรรมเนื่องจากค่าเริ่มต้นของ Pull Model ไม่จำเป็นต้องเปิดพอร์ตใด ๆ ผู้สาธารณะ ส่วน Push Model จำเป็นต้องเปิดพอร์ต 9091/tcp ที่ Server เพื่อรับข้อมูล จากเครื่อง Client เป้าหมาย

7. สรุปผลการวิจัย (Conclusion)

จากการศึกษาออกแบบพัฒนาและเปรียบเทียบสถาปัตยกรรมระบบเผื่อระวังประสิทธิภาพและความปลอดภัยพื้นฐานของระบบสารสนเทศจำนวน 2 สถาปัตยกรรมด้วย Software Prometheus ได้แก่ สถาปัตยกรรมแบบดึงข้อมูล ที่ประยุกต์ใช้เครือข่ายตัวแบบซ้อนทับด้วย Tailscale และสถาปัตยกรรมแบบส่งข้อมูลด้วย Pushgateway เพื่อหาแนวทางที่เหมาะสมที่สุดสำหรับการทำงานภายใต้ Carrier-Grade NAT (CGNAT) โดยวัดผลจากภาระของระบบความหน่วงเวลา การแจ้งเตือนและด้านความปลอดภัยไซเบอร์ ดังสรุปผลการทดสอบดังนี้

1. ด้านภาระงานของระบบ (System Overhead) Push Model มีภาระงานบนเครื่อง Client ที่ต่ำกว่า Pull Model ในทุกสถานการณ์ทดสอบเนื่องจากการส่งข้อมูลอย่างเรียบง่ายโดยไม่ต้องรัน Service ใด ๆ ไว้ตลอดเวลา
2. ด้านความหน่วงเวลาในการแจ้งเตือน Pull Model มีความเร็วในการตอบสนองที่เหนือกว่า Push Model ในทุกสถานการณ์การทดสอบ ซึ่งชี้ให้เห็นถึงข้อแลกเปลี่ยนระหว่างการประหยัดทรัพยากรของ Push Model กับความเร็วในการตอบสนองการแจ้งเตือนของ Pull Model
3. ด้านความปลอดภัยทางไซเบอร์จะเห็นได้ว่า Pull Model มีความปลอดภัยที่เป็นพื้นฐานของสถาปัตยกรรมกว่า Push Model เนื่องจากมีค่าพื้นฐานมีการจัดการที่ง่ายและมีความปลอดภัยที่เป็นค่าเริ่มต้นที่ดีกว่ามีการเข้ารหัสแบบ End-to-End ด้วย WireGuard ส่วนทาง Push Model มีค่าเริ่มต้นพื้นฐานที่เสี่ยงต่อการถูกโจมตีมากกว่าแต่มีข้อดีคือ การจัดการที่เป็นอิสระ

8. อภิปรายผลการวิจัย (Discussion)

จากการศึกษาเปรียบเทียบสถาปัตยกรรม Pull Model และ Push Model นี้ทำให้ผู้วิจัยค้นพบข้อแลกเปลี่ยนเชิงเทคนิค 3 มิติ ดังอภิปรายได้ตามนี้

8.1 มิติด้านภาระงานของระบบและทรัพยากร

ผลการทดสอบชี้ให้เห็นว่า Push Model มีประสิทธิภาพในการจัดการทรัพยากรที่เหนือกว่า Pull Model โดยมีอัตราการใช้ CPU และ Network Bandwidth ต่ำกว่าผลลัพธ์นี้เป็นผลสืบเนื่องมาจากกลไกการทำงานประมวลผลเป็นรอบ Batch Processing ของ Push Script ที่ทำงานตามรอบแล้วจบการทำงานไปต่างจาก Pull Model ที่จำเป็นต้องรัน Tailscale Daemon หรือถูกเข้ารหัสอยู่ตลอดเวลาเพื่อเชื่อมต่อโมดูลเครือข่ายและการเข้ารหัสข้อมูลเพื่อแลกกับการทะลุผ่านระบบเครือข่ายภายใต้ CGNAT สำหรับการปรับใช้งานในระดับระบบขนาดใหญ่ที่มีโหนดปลายทางจำนวนมาก หากจำเป็นต้องเลือกใช้งานรูปแบบ Pull Model ควรพิจารณาปรับรอบ การดึงข้อมูล (Scrape Interval) ให้เหมาะสมกับความสำคัญของ Metric หรือใช้ Prometheus Federation เพื่อกระจายภาระงานของ Server และอาจจะเปลี่ยนรูปแบบการเฝ้า ติดตามระบบแบบเรียลไทม์ไปเป็นรูปแบบ Hybrid Monitoring ในงานที่มีความยืดหยุ่นได้

8.2 มิติด้านความหน่วงเวลาและการตอบสนอง

ด้านความเร็วของการแจ้งเตือน Pull Model ทำได้เหนือกว่าโดยสามารถตอบสนองต่อเหตุการณ์ผิดปกติได้ใกล้เคียง กับคำว่า Real Time มากกว่าสาเหตุหลักเกิดจาก Prometheus Server สามารถเข้าถึง Client เป้าหมายได้โดยตรงผ่าน Tailscale ในขณะที่ Push Model มีความหน่วงเวลาสะสมจากการที่เครื่อง Client เป้าหมายจะต้องส่งหรือ Push ข้อมูล มาที่ Pushgateway ก่อน Prometheus Server จะเข้ามาดึงข้อมูลไปอีกครั้งตามรอบเวลาที่กำหนด ซึ่งหากเกิดปัญหา ทางเครือข่ายระหว่างส่งข้อมูลจะส่งผลให้ข้อมูลค้างไม่อัปเดตและเกิดความเข้าใจผิดในการเฝ้าระวังติดตาม (False Negative) ได้ง่ายกว่าทางออกของการที่มีข้อมูลค้างหรือความเข้าใจผิดในข้อมูลที่ Push Model มีจำเป็นต้องประยุกต์ ใช้การตรวจสอบความพร้อมข้อมูลหรือ Heartbeat ร่วมกับ Alertmanager ให้แจ้งเตือนทันทีหากค่าของ push_time_seconds หายไปเกินระยะเวลาที่กำหนดให้ส่งการแจ้งเตือนทันที



Figure 5. CPU Utilization of Pushgateway Target After Closing Port 9091 on Azure

8.3 มิติด้านความมั่นคงปลอดภัยไซเบอร์

จากการเปรียบเทียบคุณสมบัติด้านความปลอดภัยทั้ง 2 สถาปัตยกรรมตามเกณฑ์ประเมินทั้ง 7 ด้านสามารถอภิปรายจุดแข็งจุดอ่อนแต่ละรูปแบบดังนี้

1. สถาปัตยกรรม Pull Model มีการออกแบบความปลอดภัยพื้นฐานเป็นค่าเริ่มต้น ซึ่งช่วยลดภาระของผู้ดูแลระบบ แต่มีความเสี่ยงในการพึ่งพาภายนอก โดยจุดแข็งของสถาปัตยกรรมนี้คือการทำงานแบบ Stealth Mode ไม่เปิดพอร์ตใด ๆ สู่สาธารณะทำให้ระบบลดช่องทางการโจมตี (Attack Surface) และ Brute force จากภายนอกได้ Pull Model มีความสอดคล้องกับปรัชญา Zero Trust Network Access (ZTNA) โดย Never Trust, Always Verify อย่างสมบูรณ์โดยมีการยืนยัน ตัวตน ผ่าน Identity Provider และเข้ารหัสแบบ End-to-End ด้วย WireGuard Protocol โดยอัตโนมัติ ซึ่งช่วยลดความผิดพลาดของผู้ดูแล หรือ Human Error ข้อควรระวังเนื่องจากระบบต้องพึ่งพา Coordination

Server ของ Tailscale ในการจับคู่ Node หาก Server มีปัญหาอาจส่งผลกระทบต่อการทำงานเชื่อมต่อได้และด้วยลักษณะเครือข่ายแบบ Mesh Network นี้หากเครื่อง Client ถูกโจมตีหรือถูกรบกวนจากผู้โจมตีอาจใช้เป็นฐานในการสแกนหาช่องโหว่หรือโจมตีไปยัง Server ได้ หาก ผู้ดูแลระบบไม่กำหนดนโยบาย Access Control List (ACL) ให้รัดกุม

2. สถาปัตยกรรม Push Model มีการออกแบบความปลอดภัยพื้นฐานที่ต่ำในค่าเริ่มต้นของระบบ ซึ่งเป็นเพราะ Push Model มีความเป็นอิสระที่มากกว่าในการควบคุมและความเป็นส่วนตัวของข้อมูลแต่ต้องอาศัยความเชี่ยวชาญในการติดตั้งระบบป้องกันเพิ่มเติมโดยจุดแข็งที่สำคัญคือการสื่อสารแบบทางเดียวที่ Client มีหน้าที่แค่เพียงส่งข้อมูลออกไปในช่องทางที่กำหนดเท่านั้น ไม่สามารถเข้าถึง Server หรือเครือข่ายภายในได้ จึงมีความเสี่ยงต่ำที่ผู้โจมตี จะใช้ Client ที่ถูกควบคุมเพื่อโจมตีกลับมายัง Server Push Model เป็นระบบแบบอิสระ (Self-hosted) ที่ทำงานได้ด้วยตัวเอง ไม่ต้องพึ่งพาเซิร์ฟเวอร์อื่นภายนอกทำให้เหมาะกับองค์กรที่มีผู้ดูแลที่เชี่ยวชาญหรือไม่ต้องการจัดการความเสี่ยงต่อ การพึ่งพภายนอก (Third-Party Risk) ข้อควรระวังค่าเริ่มต้นของ Push Model จำเป็นต้องเปิดพอร์ต 9091 เพื่อรอรับข้อมูลจากเครื่อง Client แบบ Plain HTTP และไม่มีการยืนยันตัวตนผู้ดูแลจำเป็นต้องพัฒนาระบบความปลอดภัย เช่น พิจารณาสร้าง Reverse Proxy, ติดตั้ง TLS Certificate และ กำหนด Authentication เพื่อยกระดับ ความปลอดภัยจาก Perimeter-Based ขึ้นให้สอดคล้อง กับมาตรฐาน ZTNA

9. ข้อเสนอแนะงานวิจัย (Recommendation)

1. ควรมีการทำการศึกษาด้านความสามารถในการขยายระบบ (Scalability Study) โดยเพิ่มจำนวนโหนดปลายทางขึ้นเป็น 500 – 1,000 โหนดเพื่อประเมินประสิทธิภาพและจุดคอขวด (Bottleneck) ที่อาจเกิดขึ้นกับ Prometheus Server เครือข่าย Tailscale Coordination Server และการจัดการ Mesh Network ขนาดใหญ่
2. ควรมีการทำการวิจัยเชิงลึกด้านภาระของระบบ (Performance Overhead Analysis) เพื่อวัดผลกระทบ ของการใช้ Tailscale ต่อการใช้งาน CPU, RAM, และ Network อย่างละเอียดและเปรียบเทียบกับเทคโนโลยี Overlay Network อื่น ๆ เช่น ZeroTier หรือ Netmaker
3. ควรมีการทดลองภายใต้สภาพเครือข่ายที่ไม่สมบูรณ์ (Unreliable Network Conditions) โดยใช้เครื่องมือจำลองเครือข่ายเพื่อสร้างสถานะที่มีความหน่วงสูงและ Packet Loss เพื่อประเมินว่าสถาปัตยกรรมใดมีความทนทาน และฟื้นตัวได้ดีกว่ากัน
4. ทดสอบ Push Model แบบ Remote-Write ทดสอบสถาปัตยกรรม Push อีกรูปแบบหนึ่ง (Agent ส่งข้อมูลตรงเข้า Prometheus) เพื่อเปรียบเทียบว่าสามารถลดความหน่วง 2 ต่อ ของ Pushgateway ได้หรือไม่

10. กิตติกรรมประกาศ (Acknowledgement)

ผู้วิจัยขอขอบพระคุณผู้อำนวยการและคณะผู้บริหารสถาบันสิ่งแวดล้อมไทย นำโดย ดร.วิจารย์ สิมาฉายา บอร์ดบริหารผู้อำนวยการฝ่าย รวมถึงหัวหน้าฝ่ายและแผนกต่าง ๆ ทุกท่านที่ได้มอบทุนการศึกษารวมถึงโอกาสดี ๆ และ การสนับสนุนในงานวิจัยนี้

11. การเปิดเผยการใช้ปัญญาประดิษฐ์เชิงสร้างสรรค์ (Declaration of Generative AI)

งานวิจัยนี้ใช้ปัญญาประดิษฐ์เชิงสร้างสรรค์โดยใช้เครื่องมือ Google Gemini ในการออกแบบการแสดงผลข้อมูล (Data Visualization) ใช้สนับสนุนในการเขียนคำสั่ง PromQL สำหรับ Grafana Dashboard เพื่อเปรียบเทียบข้อมูล ของการจำลองสถานการณ์ต่าง ๆ ในส่วนของวิธีดำเนินงานวิจัย (Research Methodology) และใช้เพื่อร่างโครงสร้างของงานวิจัยนี้ก่อนที่จะมาเรียบเรียงใหม่อีกครั้ง คำสั่งที่ใช้ (Prompt) ช่วยเขียน Code คำสั่ง Prometheus หรือ PromQL เพื่อตรวจสอบการใช้งานทรัพยากรของเครื่องทั้งหมด 4 ด้านดังนี้ CPU, RAM, DISK, Network และคำสั่ง ช่วยร่างโครงสร้างของงานวิจัยเกี่ยวกับการทำงานของระบบ Network ภายใต้ CGNAT โดยใช้เทคโนโลยี Tailscale, Grafana, Pushgateway แบ่งออกเป็น 2 รูปแบบคือ Pull กับ Push มาทั้ง 5 บท และหลังจากนั้นผู้วิจัยได้นำโครงสร้างมา เขียนในแต่ละบทใหม่อีกครั้ง อีกเครื่องมือที่ใช้คือ NotebookLM ใช้เครื่องมือนี้สนับสนุนการเขียนรูปแบบการอ้างอิง ให้ตรงตามมาตรฐานที่กำหนดโดยใช้ Prompt คำสั่ง จากรูปแบบการอ้างอิงในงานวิจัยนี้ช่วยทำให้อยู่ในรูปแบบ APA หลังจากนั้น

ผู้เขียนได้ตรวจทานและคัดกรองผลลัพธ์โดยละเอียดอีกครั้ง และยืนยันความรับผิดชอบต่อความถูกต้องของบทความนี้แต่เพียงผู้เดียว

12. การแสดงบทบาทผู้เขียน (CRediT Author Statement)

Chayaphon Suwanvorn: Conceptualization, Software, Formal analysis, Investigation, Resources, Data curation, Writing-original Draft, Visualization

Chaiyaporn Khemapatapan: Conceptualization, Writing-review&editing, Methodology, Supervision, Validation, Project administration

13. เอกสารอ้างอิง (References)

Cloudflare. (n.d.). *Zero Trust Security | What is a Zero Trust Network?*

<https://www.cloudflare.com/learning/security/glossary/what-is-zero-trust>

Hellstern, T. (2022). *CERNBox Monitoring with Pushgateways* [CERN Summer Student Program Report].

Hriţcan, D.-F., & Balan, D. (2024a, September 19-20). Exposing IoT Platforms Securely and Anonymously Behind CGNAT. *2024 23rd RoEduNet Conference: Networking in Education and Research*, 1–4.

Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/RoEduNet64292.2024.10722287>

Hriţcan, D.-F., & Balan, D. (2024b, May 23-25). Using Tailscale and PfSense for Security and Anonymity of IoT Environments. *2024 International Conference on Development and Application Systems*, 91–94.

Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/DAS61944.2024.10541192>

Hriţcan, D.-F., Graur, A., Bălan, D.-G., & Timofte, E.-M. (2025). A Performance Analysis of VPN Technologies Used in an IoT Environment. *Advances in Electrical and Computer Engineering*, 25(2), 81–88.

<https://doi.org/10.4316/AECE.2025.02009>

National Cyber Security Agency. (2025). *Monthly Threat Patterns* [Files]. NCSA Drive.

<https://drive.ncsa.or.th/s/bNymMs4BbNjE8x4?dir=/&editing=false&openfile=true> (In Thai)

netElastic. (n.d.). *Conserve IPv4 Addresses and Transition to IPv6*. <https://netelastic.com/cgnat-solutions>

Prometheus. (n.d.a). *Configuration*.

<https://prometheus.io/docs/prometheus/latest/configuration/configuration>

Prometheus. (n.d.b). *Prometheus Architecture* [Infographic].

<https://prometheus.io/docs/introduction/overview>

Saputra, M. Y. E., Noprianto, Arief, S. N., Wijayaningrum, V. N., & Syaifudin, Y. W. (2024, January 28-29).

Real-Time Server Monitoring and Notification System with Prometheus, Grafana, and Telegram

Integration. *2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems*, 1808–1813). Institute of Electrical and Electronics Engineers.

<https://doi.org/10.1109/ICETSI61505.2024.10459488>

Sirviö, J. (2021). *Monitoring of a Cloud-Based IT Infrastructure*. [Master's thesis, University of Turku].

UTUPub. <https://urn.fi/URN:NBN:fi-fe2021052030903>