

## การศึกษาและทดสอบการทำงานของ Container Network Interface (CNI)

Container Network Interface (CNI) study and testing

### รายชื่อผู้จัดทำ

นายกฤษณะ พุ่มพยอม เลขประจำตัว 633040145-9

นายตรีเพชร ตระจันทร์ เลขประจำตัว 633040156-4

### อาจารย์ที่ปรึกษา

รศ. ดร.ชัชชัย คุณบัว

### อาจารย์ที่ปรึกษาร่วม

ผศ. ดร.กรชวัล ชาಯดา

ผศ. ดร.จิระเดช พลสวัสดิ์

รายงานนี้เป็นรายงานงานโครงการของนักศึกษาชั้นปีที่ 4 ซึ่งเสนอเป็นส่วนหนึ่งใน

หลักสูตรวิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น พ.ศ. 2566

**Container Network Interface (CNI) study and testing**

**Project Member**

Mr. Krissana Pumpayom ID 633040145-9

Mr. Treepaech Treechan ID 633040156-4

**Project Advisor**

Assoc. Prof. Chatchai Khunboa

**Project Co-Advisor**

Asst. Prof. Kornchawal Chaipah

Asst. Prof. Jiradej Ponsawat

This is the report of fourth year project assignment submitted in partial fulfillment of the requirement for the Degree of Bachelor of Engineering.

Department of Computer Engineering

Faculty of Engineering, Khon Kaen University 2023

## ใบประเมินผลงานโครงการ

ชื่อเรื่องภาษาไทย

การศึกษาและทดสอบการทำงานของ Container Network Interface (CNI)

ชื่อเรื่องภาษาอังกฤษ

Container Network Interface (CNI) study and testing

ชื่อผู้ทำโครงการ

นายกฤษณะ พุ่มพยอม

เลขประจำตัว 633040145-9

นายตรีเพชร ตรีจันทร์

เลขประจำตัว 633040156-4

อาจารย์ที่ปรึกษา

ดร. ชัยชนะ คุณบัว

(รศ. ดร.ชัยชนะ คุณบัว)

อาจารย์ผู้ร่วมประเมินผล

ดร. วิวัฒน์

(ผศ. ดร.กรชวัล ชาญพา)

(ผศ. ดร.จิระเดช พลสวัสดิ์)

## บทคัดย่อ

ปัจจุบันเทคโนโลยีการจำลองเสมือนได้รับความนิยมอย่างมาก โดยเฉพาะการใช้ คอนเทนเนอร์ ชีงช่วยให้นักพัฒนาซอฟต์แวร์สามารถจัดการ และปรับใช้อปเพลิกेशันได้อย่างสะดวกและรวดเร็ว อย่างไรก็ตามคอนเทนเนอร์ แต่ละตัวจำเป็นต้องมีเครือข่ายเพื่อเชื่อมต่อกับ โลกภายนอก ซึ่งจุดนี้เองที่ Container Network Interface (CNI) เข้ามา มีบทบาทสำคัญ CNI ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อคอนเทนเนอร์ เข้ากับเครือข่ายจริง ช่วยให้นักพัฒนาสามารถจัดการเครือข่ายสำหรับคอนเทนเนอร์ได้อย่างมีประสิทธิภาพ ถึงแม้ CNI เป็นเทคโนโลยีที่มีประวัติยาวนาน แต่ผู้ใช้ควรเลือก CNI ที่เหมาะสมกับการใช้งานโดยพิจารณาจากปัจจัยต่างๆ เช่น ประเภทของภาระงาน ความซับซ้อนของการใช้งานและประสิทธิภาพที่ต้องการ

ในโครงการนี้นักศึกษาได้ให้ความสนใจในการทดสอบประสิทธิภาพของ CNI 3 ชนิด ได้แก่ Calico, Flannel และ Cilium เพื่อจะทำการทดสอบประสิทธิภาพ CNI กับอุปกรณ์ทางกายภาพที่จับต้องได้ คือ Raspberry Pi 4 และ ซอฟต์แวร์ที่จำลองการทำงานของคอมพิวเตอร์เสมือนจริงบนเครือข่ายเซิฟเวอร์ เราได้ทำการวัดผลประสิทธิภาพของ CNI โดยทำการปรับใช้คอนเทนเนอร์ Iperf3 ไปที่หนนดต่างๆภายในกลุ่มที่เราได้สร้างขึ้นไว้โดยที่มี CNI ชนิดต่างๆ ทำงานอยู่ จากนั้นสั่งให้ Iperf3 โคลอئนต์ส่งข้อมูลไปยัง Iperf3 เซิร์ฟเวอร์แล้วบันทึกข้อมูลที่ได้ลงในไฟล์ Text เพื่อที่จะศึกษาประสิทธิภาพของ CNI และนำไปใช้ประโยชน์ได้อย่างเหมาะสม

จากการศึกษาและทดสอบประสิทธิภาพของ CNI 3 ประเภท ได้แก่ Calico, Flannel และ Cilium พบว่า Calico จะมีประสิทธิภาพดีที่สุดในด้านเวลาแฝงโดยให้ค่าเวลาแฝง (Latency) ที่ต่ำและคงที่เหมาะสมกับการใช้งานทุกสภาพแวดล้อมที่มีความต้องการความเสถียร, Flannel จะมีประสิทธิภาพดีที่สุดในด้านปริมาณงาน (Throughput) โดยที่ให้ค่าเฉลี่ยสูงสุดและคงที่เหมาะสมกับการใช้งานทุกสภาพแวดล้อมที่มีความต้องการความเร็วสูง และ Cilium นั้นเหมาะสมกับการใช้งานที่มีขนาดแพ็คเก็ตขนาดเล็ก (ตั้งแต่ 100 ไบต์ ขึ้นไป) โดยที่จะให้ค่าปริมาณงานสูงสุดจนถึง 25,600 ไบต์ ในทุกสภาพแวดล้อมยกเว้นบนเครื่องจำลองเสมือน

## ABSTRACT

Current virtualization technology has gained great popularity. Specifically, using Containers, allows developers to manage However, each container needs a network to connect to the outside world. This is where the Container Network Interface (CNI) comes in. CNI acts as an intermediary. Connecting Containers to the Physical Network allows developers to efficiently manage container networks. Although CNI is a useful technology for managing networks for containers, users should choose the right CNI for their application based on factors such as the type of workload, the complexity of the application, and the desired performance.

In this project, students were interested in testing the performance of 3 types of CNI, namely Calico, Flannel, and Cilium. To test the performance of CNI with a tangible physical device, namely Raspberry Pi 4, and software that simulates the operation of a computer. Virtual reality on the server We measured CNI performance by deploying the iperf3 container to nodes within a cluster that we had created with different CNI types running, then instructing the Iperf3 Client to send the data to the Iperf3 Server and log it. The information is entered into a text file to study the efficiency of CNI and put it to appropriate use.

From studying and testing the performance of 3 types of CNI, namely Calico, Flannel, and Cilium, it was found that Calico has the best performance in terms of Latency, providing low and stable Latency values, suitable for use in all environments with If stability is required, Flannel performs best in terms of throughput, providing the highest average and constant performance in all high-speed environments, and Cilium is best suited for applications with small packet sizes. (100 bytes and up) with a maximum throughput of 25,600 bytes in all environments except virtual machines.

## กิตติกรรมประกาศ

การศึกษาและทดสอบการทำงานของ Container Network Interface (CNI) สามารถสำเร็จลุล่วงได้ด้วยความกรุณาของ รศ. ดร.ชัชชัย คุณบัว ซึ่งเป็นอาจารย์ที่ปรึกษาโครงการ รวมทั้งอาจารย์ที่ปรึกษา ร่วมโครงการ ผศ. ดร.กรชวัล ชาญพา และ ผศ. ดร.จิระเดช พลสวัสดิ์ ที่ได้ให้คำปรึกษาทั้งทางด้านความรู้และเสนอแนะวิธีทางแก้ไขปัญหาที่พบเจอระหว่างการดำเนินงานตลอดจนอุปกรณ์ที่สำคัญต่อการดำเนินงานโครงการ จึงทำให้โครงการนี้สามารถดำเนินงานมาตลอดจนถึงปัจจุบัน คณะผู้จัดทำจึงขอขอบพระคุณอาจารย์ เป็นอย่างสูง ณ ที่นี่

นายกฤษณะ พุ่มพยอม

นายตรีเพชร ตรีจันทร์

## สารบัญ

หัวข้อ	หน้า
บทคัดย่อ .....	ก
ABSTRACT .....	ข
กิตติกรรมประกาศ .....	ค
สารบัญ .....	ง
สารบัญรูปภาพ .....	ช
สารบัญตาราง .....	ญ
บทที่ 1 .....	1
1.1 หลักการและเหตุผล .....	1
1.2 วัตถุประสงค์ .....	1
1.3 ขอบข่ายของโครงการ .....	2
1.4 ขั้นตอนและแผนการดำเนินโครงการ .....	2
1.5 งบประมาณที่ใช้ .....	2
1.6 ผลที่คาดว่าจะได้รับ .....	3
บทที่ 2 .....	4
2.1 แนวคิดและทฤษฎีที่เกี่ยวข้อง .....	4
2.1.1 Container Network Interface (CNI) .....	4
2.1.2 โครงสร้างเครือข่าย CNI .....	6
2.1.3 เครื่องจำลองเสมือน .....	8
2.1.4 เทคโนโลยีคอนเทนเนอร์ .....	9
2.1.5 คอนเทนเนอร์และเครื่องจำลองเสมือน .....	10
2.1.6 Docker .....	11
2.1.7 การปรับใช้ .....	12

2.1.8 Iperf3.....	17
2.1.9 โมเดลเครือข่าย CNI.....	18
2.1.10 CNI ที่ใช้ในโครงการ.....	19
2.2 งานวิจัยที่เกี่ยวข้อง.....	22
2.2.1 A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications [9] .....	22
2.2.2 Overview of Kubernetes Cni Plugins Performance [10].....	23
บพที่ 3 .....	25
3.1 เครื่องมือที่ใช้ในโครงการ .....	25
3.1.1 ฮาร์ดแวร์ที่ใช้ .....	25
3.1.2 ซอฟต์แวร์ที่ใช้ .....	25
3.2 ขั้นตอนการดำเนินงาน .....	26
3.2.1 ขั้นตอนการติดตั้งระบบปฏิบัติการบน Raspberry Pi 4 .....	26
3.2.2 ขั้นตอนการติดตั้งระบบปฏิบัติการบนเครื่องจำลองเสมือน .....	31
3.2.3 ขั้นตอนการติดตั้ง MicroK8s บนระบบปฏิบัติการ Linux .....	35
3.2.4 ขั้นตอนการเพิ่มโหนดเข้าไปในกลุ่มของ Kubernetes.....	37
3.2.5 ขั้นตอนการทดสอบด้วย Iperf3 .....	39
3.2.6 วิธีการเปลี่ยน CNI เป็น Calico .....	44
3.2.7 วิธีการเปลี่ยน CNI เป็น Cilium .....	45
3.2.8 วิธีการเปลี่ยน CNI เป็น Flannel.....	47
3.2.9 สภาพแวดล้อมโดยรวมทั้งหมดที่ใช้ในการทดสอบ .....	48
บพที่ 4 .....	50
4.1 ตารางผลการทดสอบ .....	50
4.1.1 Calico .....	50
4.1.2 Flannel .....	53

4.1.3 Cilium.....	55
4.2 กราฟแสดงประสิทธิภาพของ CNI ในแต่ละขนาดของแพ็กเก็ต .....	58
4.2.1 Calico .....	58
4.2.2 Flannel .....	60
4.2.3 Cilium.....	62
4.3 กราฟแสดงการเปรียบเทียบประสิทธิภาพของ CNI ทั้ง 3 ตัวในด้านต่างๆ ตามชนิดของโหนดที่จับคู่กัน .....	64
4.3.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R) .....	64
4.3.2 VM (S) กับ Raspberry Pi 4 (R) .....	66
4.3.3 Raspberry Pi 4 (S) กับ VM (R) .....	68
4.3.4 VM (S) กับ VM (R) .....	70
4.4 ผลของการจับคู่ฮาร์ดแวร์ในการทดสอบในแต่ละ Container Network Interface (CNI).....	72
4.4.1 Calico .....	72
4.4.2 Flannel .....	72
4.4.3 Cilium.....	72
4.5 ผลของการเปรียบเทียบแต่ละ Container Network Interface (CNI) ในแต่ละการจับคู่ฮาร์ดแวร์ ....	73
4.5.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R), VM (S) กับ Raspberry Pi 4 (R) และ Raspberry Pi 4 (S) กับ VM (R) .....	73
4.5.2 VM (S) กับ VM (R) .....	73
บทที่ 5 .....	74
5.1 สรุปผลการดำเนินงาน .....	74
5.2 ปัญหาและอุปสรรค .....	75
5.3 ข้อเสนอแนะ .....	76
บรรณานุกรม .....	77

## สารบัญรูปภาพ

รูปที่ 2.1 มุมมอง 100,000 ft บน CNI .....	5
รูปที่ 2.2 ข้อกำหนด CNI [2].....	6
รูปที่ 2.3 เครือข่าย VXLAN [3] .....	7
รูปที่ 2.4 เครือข่าย BGP [4] .....	8
รูปที่ 2.5 โครงสร้างของเครื่องจำลองเสมือน [5] .....	9
รูปที่ 2.6 โครงสร้างของคอนเทนเนอร์ [5].....	10
รูปที่ 2.7 การปรับใช้แบบดั้งเดิม [6].....	12
รูปที่ 2.8 การปรับใช้ด้วยเครื่องจำลองเสมือนจริง [6].....	14
รูปที่ 2.9 การปรับใช้ด้วยเทคโนโลยีคอนเทนเนอร์ [6].....	15
รูปที่ 2.10 การปรับใช้ด้วย Kubernetes [6] .....	17
รูปที่ 2.11 หลักการทำงานของ Iperf3 [7].....	17
รูปที่ 2.12 โหมด Overlay บน CNI [8] .....	18
รูปที่ 2.13 โหมด Underlay บน CNI [8].....	19
รูปที่ 3.1 ดาวน์โหลด Raspberry Pi Imager .....	26
รูปที่ 3.2 ติดตั้ง Raspberry Pi Imager .....	27
รูปที่ 3.3 หน้าโปรแกรม Raspberry Pi Imager .....	27
รูปที่ 3.4 อุปกรณ์ของ Raspberry Pi Imager .....	28
รูปที่ 3.5 ระบบปฏิบัติการของ Raspberry Pi Imager .....	29
รูปที่ 3.6 ระบบปฏิบัติการของ Other General-Purpose .....	29
รูปที่ 3.7 อุปกรณ์จัดเก็บข้อมูลของ Raspberry Pi Imager .....	30
รูปที่ 3.8 การตั้งค่าของโปรแกรม Raspberry Pi Imager .....	31
รูปที่ 3.9 การสร้างเครื่องจำลองเสมือนบน Proxmox .....	32
รูปที่ 3.10 รายละเอียดเครื่องจำลองเสมือน .....	32
รูปที่ 3.11 รายละเอียดระบบปฏิบัติการเครื่องจำลองเสมือน .....	32
รูปที่ 3.12 รายละเอียดการกำหนดระบบ .....	33
รูปที่ 3.13 รายละเอียดการกำหนดดิสก์ .....	33
รูปที่ 3.14 รายละเอียดการกำหนดหน่วยประมวลผลกลาง .....	34
รูปที่ 3.15 รายละเอียดการกำหนดหน่วยความจำ .....	34
รูปที่ 3.16 รายละเอียดการกำหนดเครือข่าย .....	34

รูปที่ 3.17 รายละเอียดระบบปฏิบัติการบนเครื่องจำลองเสมือน .....	35
รูปที่ 3.18 วิธีติดตั้ง MicroK8s.....	35
รูปที่ 3.19 ผลลัพธ์เมื่อ MicroK8s ไม่ทำงาน .....	36
รูปที่ 3.20 สถานะ MicroK8s .....	36
รูปที่ 3.21 ตั้งค่านามแฝงและตรวจสอบสถานะของโหนด .....	37
รูปที่ 3.22 Token สำหรับโหนด Worker .....	37
รูปที่ 3.23 ผลลัพธ์เมื่อโหนด Worker เข้าร่วมกลุ่ม .....	38
รูปที่ 3.24 ตรวจสอบสถานะโหนดภายในกลุ่ม .....	38
รูปที่ 3.25 การปรับใช้ไฟล์ Yaml .....	40
รูปที่ 3.26 รายการ Pods ในกลุ่มของ Kubernetes.....	40
รูปที่ 3.27 กำหนดเซิร์ฟเวอร์ Iperf3 .....	42
รูปที่ 3.28 การใช้งาน Code Python.....	43
รูปที่ 3.29 วิธีการตรวจสอบการใช้งาน CNI Calico .....	44
รูปที่ 3.30 อินเตอร์เฟซของ CNI Calico.....	44
รูปที่ 3.31 วิธีการหยุดใช้งาน CNI Calico .....	44
รูปที่ 3.32 การติดตั้ง CNI Cilium.....	45
รูปที่ 3.33 วิธีการตรวจสอบการใช้งาน CNI Cilium.....	46
รูปที่ 3.34 อินเตอร์เฟซของ CNI Cilium .....	46
รูปที่ 3.35 วิธีการหยุดใช้งาน CNI Cilium.....	47
รูปที่ 3.36 วิธีการติดตั้ง CNI Flannel .....	47
รูปที่ 3.37 อินเตอร์เฟซของ CNI Flannel .....	48
รูปที่ 3.38 วิธีการหยุดใช้งาน CNI Flannel .....	48
รูปที่ 3.39 สภาพแวดล้อมโดยรวมทั้งหมดที่ใช้ในการทดสอบ .....	48
รูปที่ 4.1 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Calico ในแต่ละขนาดของแพ็กเก็ต .....	58
รูปที่ 4.2 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Calico ในแต่ละขนาดของแพ็กเก็ต .....	59
รูปที่ 4.3 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Flannel ในแต่ละขนาดของแพ็กเก็ต .....	60
รูปที่ 4.4 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Flannel ในแต่ละขนาดของแพ็กเก็ต .....	61
รูปที่ 4.5 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Cilium ในแต่ละขนาดของแพ็กเก็ต .....	62
รูปที่ 4.6 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Cilium ในแต่ละขนาดของแพ็กเก็ต .....	63

รูปที่ 4.7 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R).....	64
รูปที่ 4.8 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R) .....	65
รูปที่ 4.9 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ Raspberry Pi 4 (R) .....	66
รูปที่ 4.10 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ Raspberry Pi 4 (R) .....	67
รูปที่ 4.11 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ VM (R).....	68
รูปที่ 4.12 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ VM (R) .....	69
รูปที่ 4.13 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ VM (R).....	70
รูปที่ 4.14 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ VM (R).....	71

## สารบัญตาราง

ตารางที่ 1.1 ขั้นตอนการทำงานและแผนการดำเนินโครงการ .....	2
ตารางที่ 4.1 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ Raspberry(R) ของ CNI: Calico .....	50
ตารางที่ 4.2 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ VM(R) ของ CNI: Calico .....	51
ตารางที่ 4.3 ผลการทดสอบเฉลี่ยวของแต่ละขนาด packet ระหว่าง VM(S) กับ Raspberry(R) ของ CNI: Calico .....	51
ตารางที่ 4.4 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง VM(S) กับ VM(R) ของ CNI: Calico ....	52
ตารางที่ 4.5 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ Raspberry(R) ของ CNI: Flannel.....	53
ตารางที่ 4.6 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ VM(R) ของ CNI: Flannel.....	53
ตารางที่ 4.7 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง VM(S) กับ Raspberry(R) ของ CNI: Flannel.....	54
ตารางที่ 4.8 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง VM(S) กับ VM(R) ของ CNI: Flannel..	54
ตารางที่ 4.9 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ Raspberry(R) ของ CNI: Cilium.....	55
ตารางที่ 4.10 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง Raspberry(S) กับ VM(R) ของ CNI: Cilium.....	56
ตารางที่ 4.11 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง VM(S) กับ Raspberry(R) ของ CNI: Cilium.....	56
ตารางที่ 4.12 ผลการทดสอบเฉลี่ยวของแต่ละขนาด Packet ระหว่าง VM(S) กับ VM(R) ของ CNI: Cilium..	57

## บทที่ 1

### บทนำ

#### 1.1 หลักการและเหตุผล

ปัจจุบัน แอปพลิเคชัน และอุปกรณ์ Internet of Things หรือ IoT มีจำนวนมากขึ้นและมีแนวโน้มที่จะมากขึ้นเรื่อยๆ ในอนาคต โดยเป็นเครือข่ายของอุปกรณ์ที่เชื่อมต่อกันซึ่งสามารถรวมรวมและแลกเปลี่ยนข้อมูลได้ การพัฒนาแอปพลิเคชันขนาดใหญ่ที่มีความสำคัญต่อข้อมูลและเวลาแฝง แอปพลิเคชันเหล่านี้รวมรวม จัดเก็บ และวิเคราะห์ข้อมูลจำนวนมากจากข้อมูลที่ได้รับ ข้อมูลที่ถูกใช้อาจเป็นแบบเรียลไทม์ แม้จะมีข้อมูลจำนวนมากที่ต้องจัดเก็บและวิเคราะห์ ซึ่งอาจต้องใช้ฮาร์ดแวร์และซอฟต์แวร์ที่ซับซ้อน การพัฒนาและการปรับใช้ของแอปพลิเคชันเหล่านี้เป็นสิ่งที่สำคัญดังนั้น การปรับใช้ค่อนเทนเนอร์เป็นวิธีหนึ่งที่ตอบสนองการปรับใช้ของแอปพลิเคชันในปัจจุบันเนื่องจากช่วยเพิ่มประสิทธิภาพและความสามารถในการพัฒนาและการทำงานของแอปพลิเคชันด้วยคุณสมบัติ เช่น ความเป็นเสถียร ความเร็วในการเรียกใช้งาน ความปลอดภัย และการจัดการทรัพยากรอย่างมีประสิทธิภาพ

Container Networking Interface (CNI) คือ เฟรมเวิร์กที่ใช้ในการเชื่อมต่อและจัดการเครือข่ายของค่อนเทนเนอร์ในระบบคลาวด์ หรือ แพลตฟอร์มการทำงานแบบคลาวด์ โดย CNI จะดำเนินการในชั้นตอนที่ต่อเนื่องกับชั้นตอนการสร้างค่อนเทนเนอร์ ซึ่งทำให้สามารถกำหนดเครือข่ายสำหรับค่อนเทนเนอร์แต่ละตัวได้ตามที่กำหนด การเลือกใช้งาน CNI ที่ในปัจจุบันมีหลากหลายและมีโครงสร้างเครือข่ายที่ต่างกันหากเลือกใช้ CNI ที่เหมาะสมกับระบบนั้นจะช่วยให้ผู้ใช้งานสามารถกำหนดเครือข่ายให้กับค่อนเทนเนอร์ในลักษณะที่เหมาะสมกับการทำงาน และทำให้ค่อนเทนเนอร์สามารถเชื่อมต่อกับเครือข่ายภายนอกได้อย่างมีประสิทธิภาพ และปลอดภัย โดยไม่ต้องมีการแก้ไขแอปพลิเคชันภายในค่อนเทนเนอร์ใหม่แต่ครั้งที่มีการเปลี่ยนแปลงเครือข่ายหรือสภาพแวดล้อมการทำงาน นี่เป็นคุณสมบัติที่สำคัญในการให้บริการแอปพลิเคชันในระบบคลาวด์ ที่ต้องการความยืดหยุ่นและรวดเร็วในการปรับปรุงและการเปลี่ยนแปลงการทำงานใหม่

#### 1.2 วัตถุประสงค์

1. เพื่อวัดประสิทธิภาพของปลั๊กอิน CNI คือ Calico, Flannel และ Cilium
2. เพื่อหาร่วงปลั๊กอิน CNI ที่ได้ทำการเลือกมาศึกนั้นมีสิทธิภาพในด้าน ปริมาณงาน (Throughput) และ เวลาแฝง (Latency) เป็นอย่างไร
3. เพื่อหาร่วงปลั๊กอิน CNI ที่ได้ทำการเลือกมาศึกษาจะมีสิทธิภาพที่ดีที่สุดเมื่อนำไปใช้ในสภาพแวดล้อมแบบใด

### 1.3 ขอบข่ายของโครงการ

ทำการศึกษาการส่งข้อมูลของ CNI ที่ถูกพัฒนาและใช้งานในปัจจุบัน โดยเลือก CNI ดังนี้ Calico, Flannel และ Cilium พร้อมทำการติดตั้ง ทดสอบ และเปรียบเทียบประสิทธิภาพ ด้วยการส่งข้อมูลขนาดต่างๆ

### 1.4 ขั้นตอนและแผนการดำเนินโครงการ

**ตารางที่ 1.1 ขั้นตอนการทำงานและแผนการดำเนินโครงการ**

กิจกรรม	2566						2567	
	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.
ศึกษาค้นคว้าข้อมูลที่เกี่ยวข้อง								
ติดตั้งและทดสอบการใช้งานเบื้องต้น								
ออกแบบวิธีการทดลอง								
ทดสอบ CNI ทั้ง 3 ตัว และวิเคราะห์ผลการทดสอบ								
ปรับเปลี่ยนวิธีการทดสอบเพื่อให้เหมาะสม								
ทดสอบ CNI ทั้ง 3 ตัว และวิเคราะห์ผลการทดสอบอีกรอบ								
สรุปผลลัพธ์ที่เกิดขึ้น								

### 1.5 งบประมาณที่ใช้

โครงการนี้ใช้งบประมาณไปทั้งหมด 0 บาท

## 1.6 ผลที่คาดว่าจะได้รับ

1. เรียนรู้และทราบถึงประสิทธิภาพปลั๊กอิน CNI คือ Calico, Flannel, และ Cilium ในการจัดการและเชื่อมต่อเครือข่ายสำหรับคอนเทนเนอร์ได้อย่างชัดเจน
2. สามารถระบุว่าแต่ละปลั๊กอิน CNI มีความสามารถในด้านปริมาณงานและเวลาแฝงอย่างไร จากการเปรียบเทียบผลลัพธ์ของการทดสอบระหว่าง Calico, Flannel, และ Cilium จะช่วยให้เราสามารถเลือกใช้ปลั๊กอินที่เหมาะสมกับความต้องการของเราได้
3. เรียนรู้ CNI คือ Calico, Flannel, และ Cilium ที่นำมาศึกษาจะมีสิทธิภาพที่ดีที่สุดเมื่อนำไปใช้ในสภาพแวดล้อมแบบใดจะช่วยให้ทราบถึงความเหมาะสมของแต่ละเทคโนโลยีในสภาพแวดล้อมที่เราใช้อยู่

## บทที่ 2

### ทฤษฎีเกี่ยวข้อง

ในการดำเนินโครงการศึกษาและทดสอบการทำงานของ Container Network Interface (CNI) นั้นจำเป็นต้องทำการศึกษาสื่อสารสนเทศต่างๆ เพื่อทำความเข้าใจกับเนื้อหาที่เกี่ยวข้องของ CNI ทั้งเชิงพื้นฐาน และเฉพาะทาง เพื่อให้คณะผู้จัดทำสามารถที่จะดำเนินโครงการนี้สำเร็จลุล่วง ได้ดังนี้

#### 2.1 แนวคิดและทฤษฎีเกี่ยวข้อง

##### 2.1.1 Container Network Interface (CNI)

CNI เป็นมาตรฐานแบบเปิดที่กำหนดการทำงานระหว่างคอนเทนเนอร์รันไทม์กับปลั๊กอินเครือข่ายซึ่ยให้การกำหนดค่าเครือข่ายสำหรับ Pod ใน Kubernetes มีความยืดหยุ่นปลั๊กอิน CNI ทำงานที่ต่างๆ เช่น

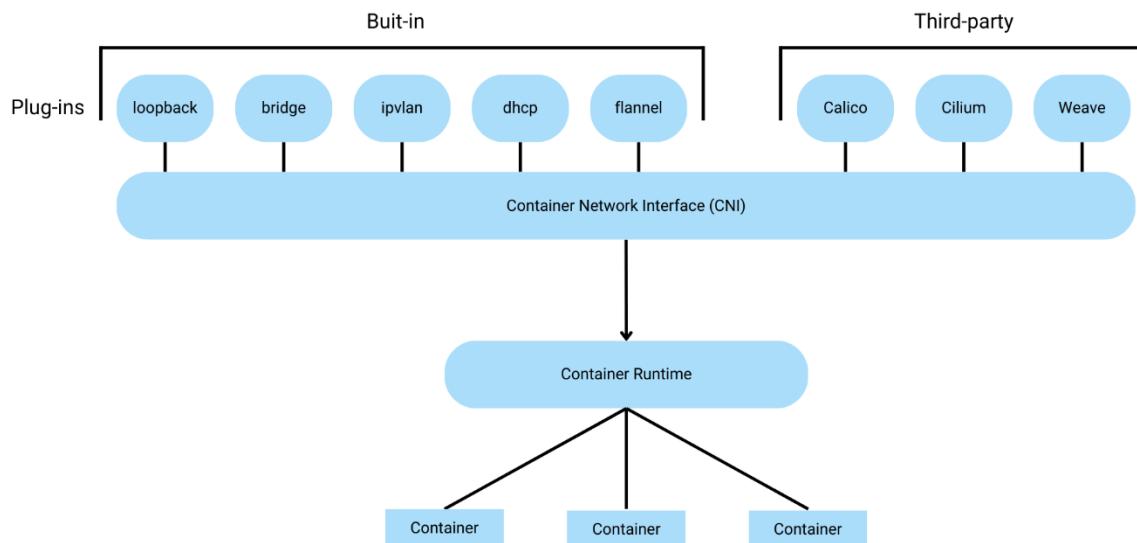
- กำหนดที่อยู่ IP
- สร้างอินเตอร์เฟสเครือข่าย
- ตั้งค่าเส้นทางเครือข่ายภายในคอนเทนเนอร์

โดยสรุป CNI ทำงานที่เป็นตัวกลางเชื่อมต่อระหว่างโปรแกรมรันคอนเทนเนอร์กับปลั๊กอินเครือข่าย ทำให้การจัดการเครือข่ายสำหรับคอนเทนเนอร์ใน Kubernetes และยืดหยุ่นมากขึ้น

CNI ทำงานบนสถาปัตยกรรมแบบปลั๊กอินที่เรียบง่าย เมื่อมีการสร้าง Pod ใน Kubernetes คอนเทนเนอร์รันไทม์ (เช่น Docker) จะเรียกปลั๊กอิน CNI เพื่อตั้งค่าสภาพแวดล้อมเครือข่าย ปลั๊กอินเหล่านี้ สามารถเขียนด้วยภาษาโปรแกรมต่างๆ และ สื่อสารกับคอนเทนเนอร์รันไทม์ผ่านทางอินพุตและเอาต์พุต มาตรฐาน พวกรเข้าใช้ประโยชน์จากสเต็กเครือข่าย Linux เพื่อกำหนดค่าเครือข่ายสำหรับคอนเทนเนอร์

วัตถุประสงค์หลักของ CNI คือการอนุญาตให้ใช้ปลั๊กอินเครือข่ายที่แตกต่างกันได้กับรันไทม์ ของคอนเทนเนอร์ นี้ช่วยให้ Kubernetes มีความยืดหยุ่นและทำงานร่วมกับแนวทางการเครือข่ายที่แตกต่าง กันได้ เช่น Calico, Flannel, และ Weave Net ปลั๊กอิน CNI รับผิดชอบในการกำหนดค่าอินเตอร์เฟสเครือข่าย ใน Pod เช่น กำหนดที่อยู่ IP, กำหนดค่าเส้นทาง, และการจัดการนโยบายความปลอดภัยของเครือข่าย

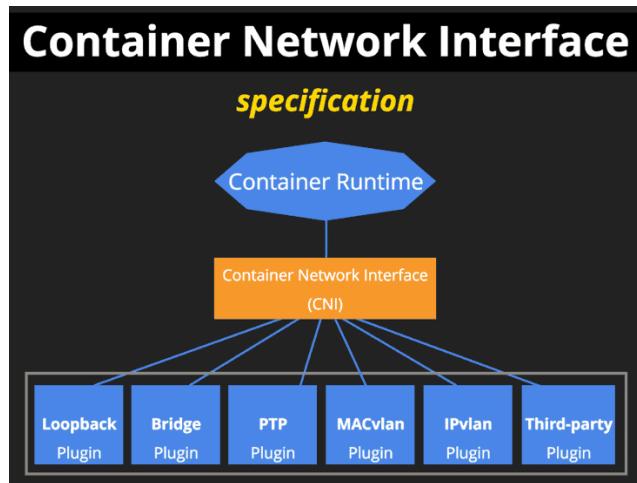
ปลั๊กอิน CNI มีหลากหลายชนิด ถูกออกแบบมาเพื่อตอบสนองความต้องการด้านเครือข่ายที่แตกต่างกันไป ตัวอย่างปลั๊กอินยอดนิยม เช่น Calico, Flannel, Weave และ Canal ปลั๊กอินเหล่านี้มีคุณสมบัติต่างๆ เช่น การแยกเครือข่าย นโยบายความปลอดภัย การกระจายโหลด และการเชื่อมต่อกับทรัพยากรเครือข่ายภายนอก



รูปที่ 2.1 มุมมอง 100,000 ft บน CNI

### 2.1.1.1 CNI อยู่ที่ไหน?

CNI เป็นเลเยอร์ของสิ่งที่เป็นนามธรรมระหว่างรันไทม์ของคอนเทนเนอร์ และโครงสร้างพื้นฐานของเครือข่าย สิ่งนี้ทำให้คอนเทนเนอร์รันไทม์เป็นอิสระจากโครงสร้างพื้นฐานของเครือข่าย ทำให้ปรับใช้แอปพลิเคชันคอนเทนเนอร์ในสภาพแวดล้อมที่หลากหลายได้ง่ายขึ้น CNI อยู่ในเครื่องไฮสต์เดียวกันกับรันไทม์ของคอนเทนเนอร์และโครงสร้างพื้นฐานของเครือข่าย เป็นกระบวนการซอฟต์แวร์ที่รับผิดชอบในการกำหนดค่าเครือข่ายสำหรับคอนเทนเนอร์ CNI สื่อสารกับโครงสร้างพื้นฐานของเครือข่ายเพื่อสร้างอินเทอร์เฟซเครือข่าย กำหนดที่อยู่ IP และตั้งค่าการกำหนดเส้นทาง



รูปที่ 2.2 ข้อกำหนด CNI [2]

### 2.1.2 โครงสร้างเครือข่าย CNI

#### 2.1.2.1 VXLAN (Virtual eXtensible Local Area Network)

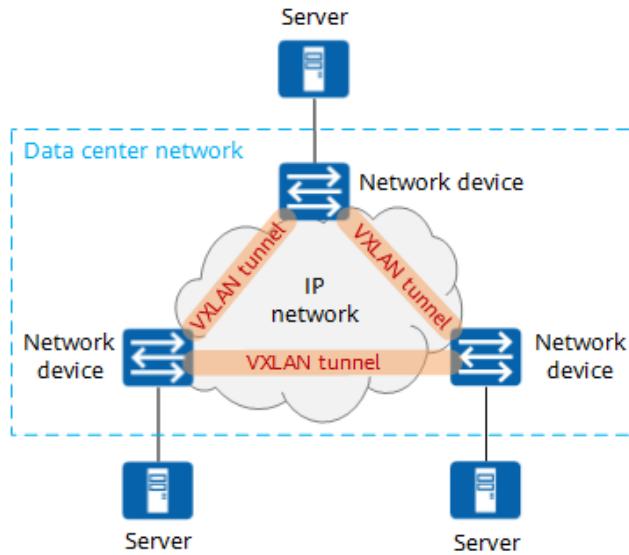
VXLAN (Virtual eXtensible Local Area Network) คือมาตรฐานเทคโนโลยีใหม่ของ Internet Engineering Task Force (IETF) ที่ช่วยให้เครือข่ายกายภาพเดียวสามารถแบ่งปันให้กับองค์กรหรือ "ผู้เช่า" หลายองค์กรได้ โดยที่ผู้เช่าแต่ละคนไม่สามารถมองเห็นทรัพยากรเครือข่ายของผู้อื่น

การทำงานของ VXLAN เปรียบได้กับ ห้องชุด ภายใน อาคารพักอาศัย ที่แต่ละห้องเป็นที่อยู่อาศัยส่วนตัวแยกจากกัน ภายในโครงสร้างเดียวกัน เช่นเดียวกับ VXLAN ที่เป็นเซกเมนต์เครือข่ายส่วนตัวแยกจากกัน ภายในเครือข่ายกายภาพที่ใช้ร่วมกัน

โดยทางเทคนิคแล้ว VXLAN สามารถแบ่งเครือข่ายทางกายภาพออกเป็นเครือข่ายใหม่ หรือ เครือข่ายลوجิคอลได้มากถึง 16 ล้านเครือข่ายทำงานโดยการห่อหุ้มเฟรม Ethernet เลเยอร์ 2 ภายในแพ็กเกจ UDP (User Datagram Protocol) เลเยอร์ 4 ร่วมกับ หัวข้อ (Header) ของ VXLAN

เมื่อใช้ร่วมกับ Ethernet Virtual Private Network (EVPN) ซึ่งทำหน้าที่ในการขนส่งทรัพยากร Ethernet ในเครือข่ายใหม่ใน WAN VXLAN จะช่วยให้เครือข่ายเลเยอร์ 2 ขยายไปยังเครือข่าย IP หรือ MPLS เลเยอร์ 3 ได้

ตามที่แสดงในภาพด้านล่าง จะเห็นว่า VXLAN เป็นเทคโนโลยีที่มีการส่งท่อเชื่อมต่อ (Tunneling) เข้ากันระหว่างอุปกรณ์เครือข่าย IP ระหว่างอุปกรณ์เครือข่ายต้นทางและปลายทาง เพื่อบรรจุแพคเกจทางด้านผู้ใช้และส่งต่อผ่านทางท่อ

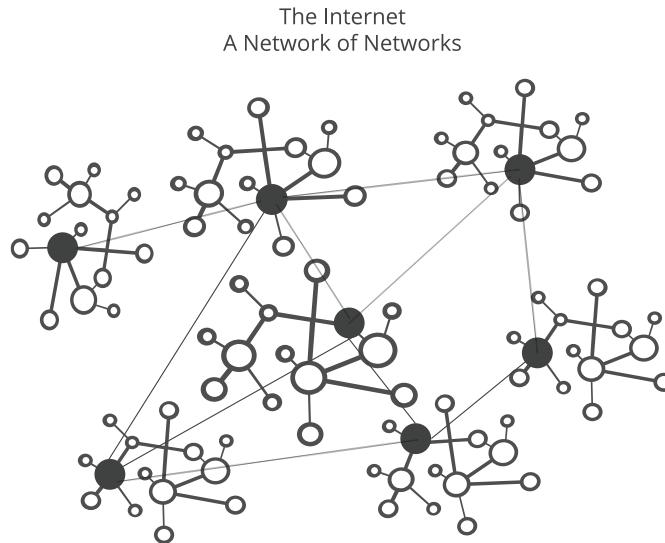


รูปที่ 2.3 เครือข่าย VXLAN [3]

#### 2.1.2.2 BGP ( Border Gateway Protocol )

Border Gateway Protocol (BGP) เปรียบเสมือนกับการบริการไปรษณีย์ของอินเทอร์เน็ต ในกรณีที่มีคนส่งจดหมายลงในกล่องไปรษณีย์ หน่วยงานไปรษณีย์จะประมวลผลจดหมายดังกล่าวและเลือกเส้นทางที่รวดเร็วและมีประสิทธิภาพในการส่งจดหมายไปยังผู้รับ ในทางเดียวกัน เมื่อมีคนส่งข้อมูลผ่านอินเทอร์เน็ต BGP จะรับผิดชอบในการพิจารณาเส้นทางที่มีอยู่ทั้งหมดที่ข้อมูลสามารถเดินทางได้ และเลือกเส้นทางที่ดีที่สุด ซึ่งมักหมายถึงการเดินทางผ่านระบบอโตโนมัสหลายๆ ระบบพร้อมกัน

BGP เป็นโปรโตคอลที่ทำให้อินเทอร์เน็ตสามารถทำงานได้โดยการเปิดการกำหนดเส้นทางข้อมูล ในกรณีที่มีผู้ใช้งานในสิงคโปร์หรือเว็บไซต์ที่เชื่อมต่อตันทางที่อยู่ในอาร์เจนตินา โปรโตคอล BGP คือที่มาของการเชื่อมต่อที่ทำให้การสื่อสาร เช่นนี้เกิดขึ้นอย่างรวดเร็วและมีประสิทธิภาพ

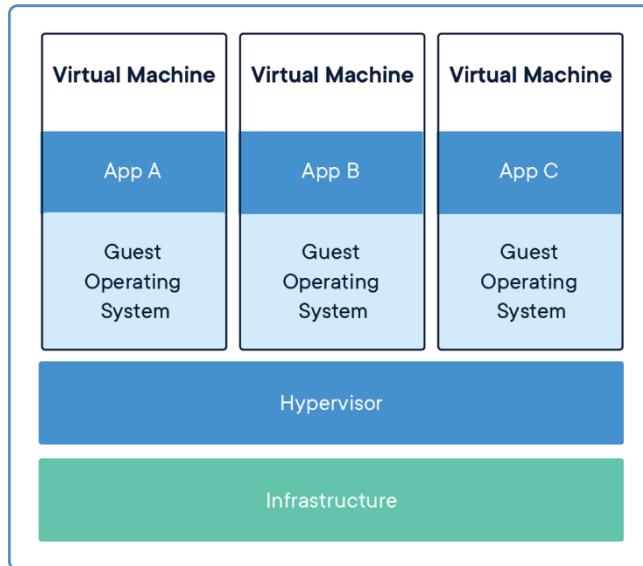


รูปที่ 2.4 เครือข่าย BGP [4]

### 2.1.3 เครื่องจำลองเสมือน

เครื่องจำลองเสมือน (Virtual Machine: VM) คือทรัพยากรคอมพิวเตอร์ที่เช่าอฟต์แวร์แทนคอมพิวเตอร์จริงในการรันโปรแกรมและใช้งานแอปพลิเคชัน เครื่องจำลองเสมือนแยก (Guest VM) หนึ่งเครื่องหรือมากกว่านั้นทำงานบนเครื่องจริง (Host Machine) แต่ละเครื่องเสมือนรันระบบปฏิบัติการของตัวเองและทำงานแยกจาก VM อื่น ๆ แม้ว่าทั้งหมดจะทำงานบนโฮสต์เดียวกัน ตัวอย่างเช่น เครื่องเสมือนที่มีระบบปฏิบัติการเป็น Mac สามารถทำงานบนคอมพิวเตอร์ที่มีระบบปฏิบัติการเป็น Windows ได้

เทคโนโลยีเครื่องจำลองเสมือนมี การใช้งานสำหรับหลาย กรณีทั้งในสภาพแวดล้อมภายในองค์กรและบนคลาวด์ ล่าสุด บริการคลาวด์สาธารณะกำลังใช้เครื่องเสมือนเพื่อจัดหาทรัพยากรแอปพลิเคชันเสมือนให้กับผู้ใช้หลายคนพร้อมกัน เพื่อการคำนวณที่มีประสิทธิภาพคุ้มค่าและยืดหยุ่นยิ่งขึ้น



รูปที่ 2.5 โครงสร้างของเครื่องจำลองเสมือน [5]

#### 2.1.4 เทคโนโลยีคอนเทนเนอร์

คอนเทนเนอร์ (Container) คือ หน่วยมาตรฐานของซอฟต์แวร์ที่บรรจุโค้ดและการพึ่งพา (Dependencies) ทั้งหมดไว้ เพื่อให้แอปพลิเคชันทำงานได้อย่างรวดเร็วและเข้าถึงได้บนสภาพแวดล้อมการคำนวณที่แตกต่างกัน

Docker Container Image เป็นแพ็คเกจซอฟต์แวร์ที่บางทีเป็นแบบสแตนด์อโลน และเรียกใช้งานได้ง่าย ประกอบด้วยทุกสิ่งที่จำเป็นสำหรับการรันแอปพลิเคชัน: โค้ด, รันไทม์, เครื่องมือระบบ, ไลบรารีระบบ และการตั้งค่า

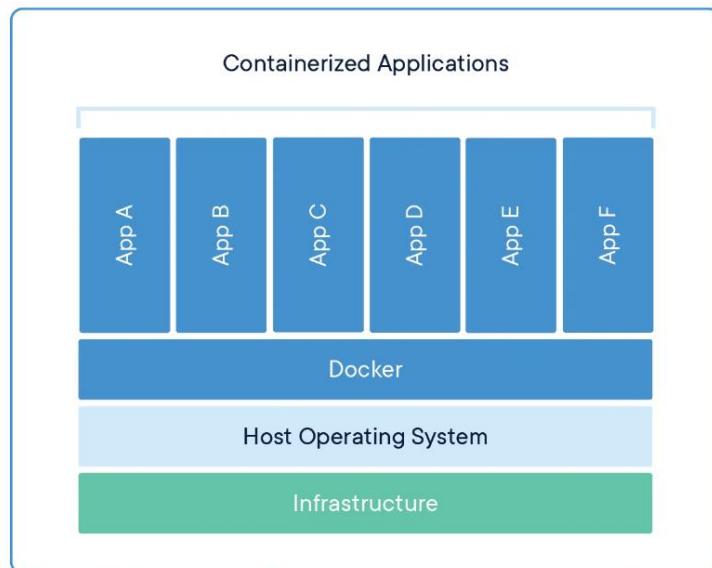
Container Image จะถูกจัดทำเป็นคอนเทนเนอร์ในขณะที่รันและในกรณีของ Docker Containers นั้น Image จะถูกจัดทำเป็นคอนเทนเนอร์เมื่อรันบน Docker Engine ซึ่งรองรับทั้งแอปพลิเคชันบน Linux และ Windows ซอฟต์แวร์ที่อยู่ในคอนเทนเนอร์จะทำงานเหมือนเดิมเสมอไม่ว่าจะเป็นโครงสร้างพื้นฐานแบบใด

คอนเทนเนอร์จะแยกซอฟต์แวร์ออกจากสภาพแวดล้อม และช่วยให้ทำงานได้อย่างสม่ำเสมอแม้จะมีความแตกต่าง เช่น ระหว่างการพัฒนาและการจัดแสดง

ข้อดีของการใช้คอนเทนเนอร์ Docker บน Docker Engine:

- มาตรฐาน: Docker สร้างมาตรฐานอุตสาหกรรมสำหรับคอนเทนเนอร์ ทำให้สามารถพกพาไปใช้ที่ได้

- น้ำหนักเบา: คอนเทนเนอร์ใช้เครื่องเรนระบบปฏิบัติการเดียวกันของเครื่อง ดังนั้นจึงไม่จำเป็นต้องมีระบบปฏิบัติการสำหรับแต่ละแอปพลิเคชัน ช่วยเพิ่มประสิทธิภาพของเซิร์ฟเวอร์ และลดต้นทุนเซิร์ฟเวอร์และใบอนุญาต
- ปลอดภัย: แอปพลิเคชันมีความปลอดภัยมากขึ้นในคอนเทนเนอร์ และ Docker มีความสามารถในการแยกส่วนเริ่มต้นที่เข็งแกร่งที่สุดในอุตสาหกรรม



รูปที่ 2.6 โครงสร้างของคอนเทนเนอร์ [5]

### 2.1.5 คอนเทนเนอร์และเครื่องจำลองเสมือน

คอนเทนเนอร์และเครื่องจำลองเสมือนแตกต่างกันในหลายๆ ด้าน คอนเทนเนอร์จำลองแกนของระบบปฏิบัติการ (Operating System หรือ OS) ของโฮสต์และใช้งานแอปพลิเคชันต่างๆ บนระบบปฏิบัติการเดียว สร้างสภาพแวดล้อมในการติดตั้งแอปพลิเคชันที่สม่ำเสมอในทุกขั้นตอนของวงจรชีวิตการพัฒนาซอฟต์แวร์ (Software Development Life Cycle หรือ SDLC) คอนเทนเนอร์เบาและเก็บเฉพาะส่วนประกอบที่จำเป็นในการเรียกใช้แอปพลิเคชันที่ถูกจัดเก็บในรูปแบบคอนเทนเนอร์ ส่งผลให้การใช้หน่วยความจำน้อยลงและสามารถย้ายที่ติดตั้งได้ในการจัดการกลุ่มนี้อหา มีการใช้เครื่องมือในการจัดการสอนซึ่งทำให้การติดตั้งคอนเทนเนอร์ เช่น การจัดการการเพิ่มขนาด การจัดการ การขยายและการเขื่อมต่อเครือข่าย เครื่องมือจัดการกลุ่มนี้นิยมประกอบด้วย Kubernetes, Apache Mesos, และ Docker Swarm

ในทางตรงกันข้าม เครื่องจำลองเสมือน (Virtual Machine หรือ VM) จำลองระบบคอมพิวเตอร์กลางทางกายภาพที่มีหน่วยประมวลผลหน่วยความจำ ติสก์ ฯลฯ VM ถือว่าเป็นคอมพิวเตอร์เสมือนและสามารถเรียกใช้ระบบปฏิบัติการหลายระบบในระบบคอมพิวเตอร์เดียวกัน และทำได้ด้วยการใช้

ตัวกำหนดความจุ (Hypervisor หรือ Virtual Machine Monitor) ที่เป็นตัวควบคุมเครื่องจำลองเสมือนในการทำงานโดยใช้ทรัพยากรของฮาร์ดแวร์ทางกายภาพในฐานะพื้นฐาน VM ซอฟต์แวร์ที่นิยมประกอบด้วย Azure VM, VMware Workstation Pro เป็นต้น VM ใช้หน่วยความจำมากกว่าและมีความเคลื่อนย้ายน้อยกว่าคอนเทนเนอร์ การเรียกใช้หลาย VM บนเซิร์ฟเวอร์กลางทางกายภาพเดียวกันอาจส่งผลให้เกิดความไม่เสถียรภาพในการทำงานบางครั้ง องค์กรใช้เครื่องมือตรวจสอบ VM เพื่อวิเคราะห์และรักษาประสิทธิภาพของเครื่องจำลองเสมือนในระบบ Hybrid, Cloud, และภายในองค์กร สภาพแวดล้อมองค์กรเหล่านี้ตรวจสอบสุขภาพของสิ่งแวดล้อมเสมือนจริงทั้งหมดและแก้ไขปัญหาที่เกี่ยวข้องกับประสิทธิภาพอย่างมีประสิทธิภาพ

### 2.1.6 Docker

Docker เป็นแพลตฟอร์มโอเพนซอร์สที่ช่วยให้นักพัฒนาสร้าง ปรับใช้ รัน อัปเดต และจัดการ "คอนเทนเนอร์" ซึ่งเป็นคอมโพเนนต์มาตรฐานที่สามารถทำงานได้จริง ประกอบด้วยโค้ดต้นฉบับของแอปพลิเคชัน ไลบรารีระบบปฏิบัติการและการอ้างอิงที่จำเป็นเพื่อรันโคเด้นน์ในสภาพแวดล้อมใดก็ได้

คอนเทนเนอร์ ช่วยลดความซับซ้อนในการพัฒนาและการส่งมอบแอปพลิเคชันแบบกระจายพากมันได้รับความนิยมมากขึ้นเรื่อยๆ เนื่องจากองค์กรต่างๆ เปลี่ยนไปใช้การพัฒนาแบบคลาวด์เนทีฟและสภาพแวดล้อมมัลติคลาวด์แบบไฮบริด นักพัฒนาสามารถสร้างคอนเทนเนอร์ได้โดยไม่ต้องใช้ Docker โดยทำงานโดยตรงกับความสามารถที่สร้างไว้ใน Linux และระบบปฏิบัติการอื่นๆ แต่ Docker ทำให้การสร้างคอนเทนเนอร์รวดเร็ว ง่ายดาย และปลอดภัยยิ่งขึ้น ขณะที่เขียนนี้ Docker รายงานว่ามีนักพัฒนาที่ใช้แพลตฟอร์มนี้เกิน 13 ล้านคน

Docker ยังหมายถึง Docker, Inc. ซึ่งเป็นบริษัทที่จำหน่าย Docker เวอร์ชันสำหรับใช้งานเชิงพาณิชย์ และยังหมายถึงโปรเจกต์โอเพนซอร์สของ Docker ซึ่ง Docker, Inc. และองค์กรอื่นๆ บุคคลอื่นๆ อีกมากมายมีส่วนร่วมในการพัฒนา

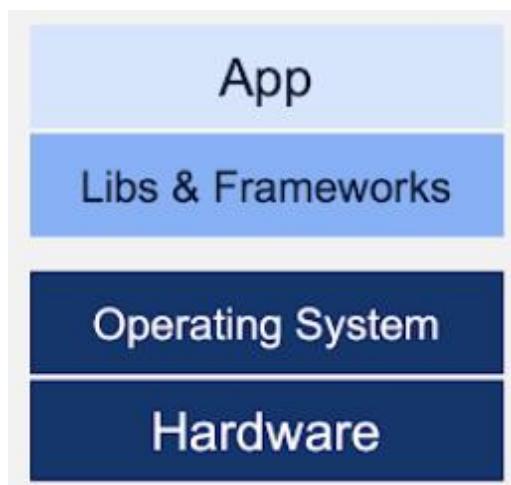
## 2.1.7 การปรับใช้

### 2.1.7.1 การปรับใช้แบบตั้งเดิม

การติดตั้งแบบการปรับใช้แบบตั้งเดิม หรือ Traditional Deployment เป็นวิธีการในการนำเสนอและปรับใช้ซอฟต์แวร์หรือแอปพลิเคชัน ที่พัฒนาขึ้นโดยที่ซอฟต์แวร์จะถูกติดตั้งบนเซิร์ฟเวอร์ทางกายภาพหรือคอมพิวเตอร์ วิธีการนี้มักจะเป็นที่นิยมในอดีต เนื่องจากมีความสะดวกและเชื่อถือได้ในการจัดการและควบคุมของซอฟต์แวร์เซิร์ฟเวอร์ที่ติดตั้งและคอมพิวเตอร์ที่ใช้งานจะอยู่ภายใต้การดูแลรักษาของทีมที่ได้รับมอบหมาย การอัปเกรด และ การบำรุงรักษาซอฟต์แวร์สามารถดำเนินการได้ภายในองค์กรโดยง่าย อย่างไรก็ตาม อาจมีความซับซ้อนในการจัดการของซอฟต์แวร์ที่ต้องการการปรับปรุงอย่างต่อเนื่องหรือการปรับขนาดเนื่องจากต้องมีการติดตั้งและการดูแลรักษาที่เป็นไปอย่างมีระเบียบและพิจารณาถึงความมั่นคง และความปลอดภัยในการเปิดเผยข้อมูลที่เกี่ยวข้อง

ข้อเสีย:

1. ไม่มีการแยกทรัพยากร
2. การใช้งานที่มากเกินไปโดยแอพเดียวอาจทำให้ทั้งระบบพังได้
3. ปัญหาการปรับขนาด
4. เวลาหยุดทำงานนาน ในตอนที่จะทำการปรับปรุงระบบ
5. มีราคาแพงในการบำรุงรักษาเซิร์ฟเวอร์ทางกายภาพ



รูปที่ 2.7 การปรับใช้แบบตั้งเดิม [6]

### 2.1.7.2 การปรับใช้ด้วยเครื่องจำลองเสมือนจริง

การติดตั้งแบบการปรับใช้ด้วยเครื่องจำลองเสมือนจริง หรือ Virtualized

Deployment เป็นวิธีการนำเสนอด้วยแอปพลิเคชันโดยใช้เทคโนโลยี การจำลองเสมือน เพื่อสร้างสภาพแวดล้อมที่ทำงานอยู่บน เซิร์ฟเวอร์ทางกายภาพหรือระบบคอมพิวเตอร์ที่มีความยืดหยุ่นและ ปรับเปลี่ยนได้ตามความต้องการ ใน การติดตั้งแบบนี้ โปรแกรมและแอปพลิเคชันจะถูกติดตั้งบน เซิร์ฟเวอร์ทาง กายภาพหรือคอมพิวเตอร์ที่มีระบบปฏิบัติการเสมือน (Virtualized Operating System) ทำให้สามารถรัน หลาย ๆ แอปพลิเคชันบนเครื่องเดียวได้โดย เป็นอิสระจากซอฟต์แวร์หรือระบบปฏิบัติการอื่น ๆ

การใช้งานแบบการปรับใช้ด้วยเครื่องจำลองเสมือนจริง มักเพิ่มความยืดหยุ่นในการ จัดการ และ การปรับปรุงซอฟต์แวร์สามารถเพิ่มหรือลดขนาดของระบบปฏิบัติการเสมือนจริงได้ตามความ ต้องการและสามารถทำการจัดการการทำสำเนาข้อมูลและการคืนค่าข้อมูล ได้อย่างมีประสิทธิภาพ เนื่องจาก สามารถใช้เทคโนโลยีการจำลองเสมือนเพื่อสร้างสภาพแวดล้อมทดสอบหรือที่เก็บข้อมูลแบบสำรองได้ในที่หนึ่ง

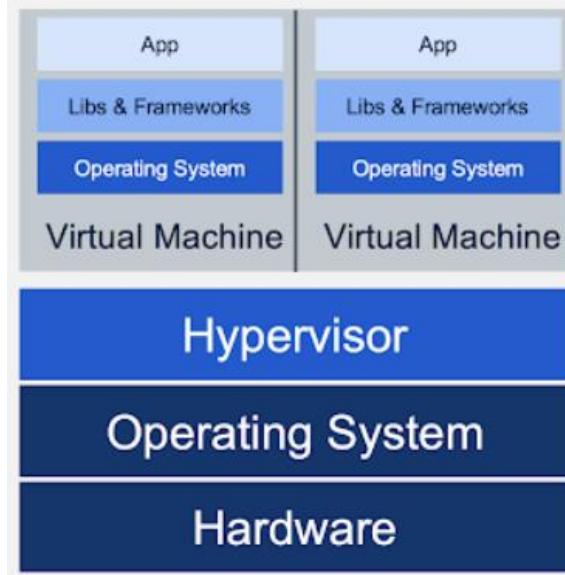
นอกจากนี้การใช้งานแบบการปรับใช้ด้วยเครื่องจำลองเสมือนจริงยังช่วยลดค่าใช้จ่าย ในด้านการดูแลรักษา และ อำนวยความสะดวกในการจัดการทรัพยากรโดยใช้การจำลองอุปกรณ์ハードแวร์ทาง กายภาพเป็นเครื่องมือสำหรับการดำเนินการ ทั้งนี้การใช้งานแบบการปรับใช้ด้วยเครื่องเสมือนจริงยังมี ความสำคัญในการเพิ่มความมั่นคง และ ความปลอดภัยในการใช้งานซอฟต์แวร์ โดยที่ผู้ใช้สามารถเข้าถึงและใช้ งานซอฟต์แวร์ได้ผ่านทางเครือข่ายที่มีการควบคุมและการรักษาข้อมูลที่เข้าถึงในที่ที่มั่นคงและปลอดภัย

ข้อดี:

1. ใช้ทรัพยากรได้ดีกว่าวิธีเดิม ๆ
2. แอปพลิเคชันถูกแยกออกจากกัน

ข้อเสีย:

1. OS images มีน้ำหนักมาก (GB) และมีกระบวนการเริ่มต้นที่ช้า
2. แอปพลิเคชันไม่สามารถพกพาได้
3. ไม่สามารถปรับขนาดได้
4. อาจมีราคาแพง



รูปที่ 2.8 การปรับใช้ด้วยเครื่องจำลองเสมือนจริง [6]

#### 2.1.7.3 การปรับใช้ด้วยเทคโนโลยีคอนเทนเนอร์

การติดตั้งแบบการปรับใช้ด้วยเทคโนโลยีคอนเทนเนอร์ หรือ Container Deployment เป็นวิธีการนำเสนอ และ ใช้งานซอฟต์แวร์หรือแอปพลิเคชันโดยใช้เทคโนโลยีคอนเทนเนอร์ เพื่อ สร้างสภาพแวดล้อมที่เรียกว่า "คอนเทนเนอร์" ซึ่งเป็นพื้นที่แยกจากแอปพลิเคชันอื่นๆ บนระบบปฏิบัติการที่ เป็นเซิร์ฟเวอร์ทางกายภาพ หรือ คอมพิวเตอร์ที่มีความสามารถในการรันคอนเทนเนอร์

การปรับใช้ด้วยเทคโนโลยีคอนเทนเนอร์ แอปพลิเคชัน และ บริการต่าง ๆ จะถูก จัดเก็บภายในคอนเทนเนอร์และแยกจากระบบปฏิบัติการ และ พื้นที่ทำงานของ莎าร์ดแวร์ทางกายภาพโดยส่วน ของแอปพลิเคชันจะถูกแพ็คเรียบร้อยในคอนเทนเนอร์พร้อมกับทุกอย่างที่ต้องการในการเรียกใช้งาน รวมถึง ไลบรารี และ สิ่งที่จำเป็นทั้งหมด การใช้งานแบบนี้ทำให้แต่ละแอปพลิเคชันสามารถทำงานได้แยกจากกันโดย ไม่มีผลกระทบกับอีกส่วนหนึ่งของระบบ และช่วยเพิ่มความยืดหยุ่นในการจัดการและการปรับปรุง

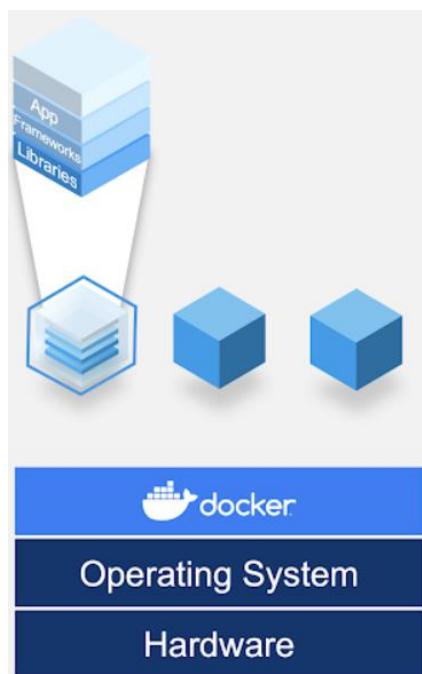
การใช้งานด้วยรูปแบบการปรับใช้ด้วย เทคโนโลยีคอนเทนเนอร์รวมถึงความสามารถ ในการสร้างและทำลายคอนเทนเนอร์ทำได้อย่างรวดเร็ว การโยกย้ายคอนเทนเนอร์ระหว่างระบบ และ ความสามารถในการทำสำรองข้อมูลและคืนค่าได้อย่างมีประสิทธิภาพ เรียกได้ว่า การปรับใช้คอนเทนเนอร์ เป็น วิธีการที่สามารถลดความซับซ้อนในการจัดการระบบและเพิ่มความเร็วและ ประสิทธิภาพในการใช้งานและ พัฒนาซอฟต์แวร์และแอปพลิเคชันในองค์กร

ข้อดี:

1. น้ำหนักเบา (Mbs) และ สามารถเริ่มต้นได้อย่างรวดเร็ว
2. คอนเทนเนอร์สามารถพกพาได้
3. ราคาไม่แพง
4. ปรับขนาดได้

ข้อเสีย:

1. แอปพลิเคชันไม่ได้แยกอย่างสมบูรณ์ ความปลอดภัยเป็นเรื่องที่น่ากังวล
2. การจัดการถือเป็นสิ่งสำคัญ



รูปที่ 2.9 การปรับใช้ด้วยเทคโนโลยีคอนเทนเนอร์ [6]

#### 2.1.7.4 การปรับใช้ด้วย Kubernetes

การปรับใช้ด้วย Kubernetes เป็นวิธีการในการนำเสนอและจัดการกลุ่มของคอนเทนเนอร์โดยใช้ระบบ Kubernetes ซึ่งเป็นเครื่องมือสำหรับจัดการและควบคุมคอนเทนเนอร์ระดับมาตรฐาน โดย Kubernetes ระบบโอเพนซอร์สที่ถูกพัฒนาโดย Google และได้รับการสนับสนุนจากชุมชนกว้างขวาง

การปรับใช้ด้วย Kubernetes คอนเทนเนอร์และแอปพลิเคชันจะถูกติดตั้งและเรียกใช้งานในรูปแบบของ "Pods" ซึ่งเป็นกลุ่มของคอนเทนเนอร์ที่รวมกันและทำงานร่วมกัน การจัดการ Pod และการปรับปรุงแอปพลิเคชันสามารถทำได้โดย Kubernetes โดยอัตโนมัติ ซึ่งช่วยลดความซับซ้อนในการดูแล

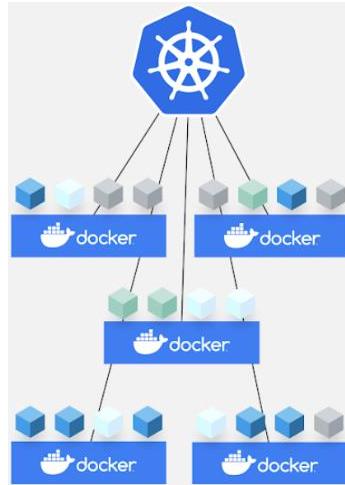
และควบคุมระบบ Kubernetes ยังมีความสามารถในการจัดการแบบอัตโนมัติ และ สร้างความสามารถในการขยายขนาด ที่มีประสิทธิภาพสำหรับแอปพลิเคชัน นอกจากนี้ยังมีการจัดการการเปลี่ยนแปลงและการปรับปรุงให้เกิดขึ้นโดยอัตโนมัติตามที่ต้องการ และยังมีความสามารถในการจัดการแบบเรียลไทม์ เพื่อให้แอปพลิเคชันทำงานได้อย่างต่อเนื่องและมีประสิทธิภาพในการใช้ทรัพยากร

#### ข้อดี:

1. ช่วยลดความเสี่ยงในการดำเนินงานการอัปเดตแอปพลิเคชัน ซึ่งช่วยให้มีการกำหนดเวอร์ชันและการกู้คืนเป็นเรื่องง่ายและปลอดภัย
2. ช่วยให้สามารถอัปเดตและจัดการกับแอปพลิเคชันได้อย่างมีประสิทธิภาพ โดยมีการทำงานอัตโนมัติเพื่อให้แน่ใจว่าแอปพลิเคชันยังคงทำงานแม้ว่าจะมีการเปลี่ยนแปลงเวอร์ชันของแอปพลิเคชันกำลังดำเนินอยู่
3. สามารถเปลี่ยนแปลงการตั้งค่าและการอัปเดตแอปพลิเคชันในขณะที่ระบบกำลังทำงาน โดยไม่ต้องหยุดเซอร์วิสหรือแอปพลิเคชัน

#### ข้อเสีย:

1. อาจมีความซับซ้อนสูง เนื่องจากต้องปรับค่าและกำหนดการตั้งค่าให้ถูกต้องเพื่อให้การอัปเดตและการจัดการเป็นไปได้ตามที่ต้องการ
2. การทำความเข้าใจและเรียนรู้เกี่ยวกับการใช้งาน การปรับใช้ Kubernetes อาจจะใช้เวลาและความพยายามมากกว่าการใช้บริการอื่น
3. การที่ระบบอัปเดตและการเปลี่ยนแปลงเกิดขึ้นบ่อยครั้งอาจจะเกิดข้อผิดพลาดในการจัดการและติดตามการเปลี่ยนแปลง
4. อาจจะต้องมีการใช้งานเครื่องมือและอุปกรณ์เสริมเพิ่มเติมเพื่อให้การตั้งค่าและการจัดการ การปรับใช้ Kubernetes เป็นไปได้ตามต้องการ

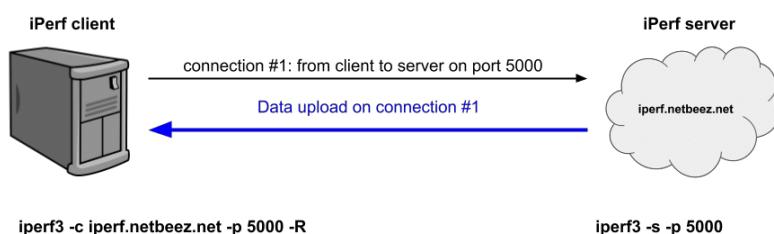


รูปที่ 2.10 การปรับใช้ด้วย Kubernetes [6]

### 2.1.8 Iperf3

Iperf3 เป็นเครื่องมือที่ใช้ในการทดสอบ และ วัดประสิทธิภาพของเครือข่ายคอมพิวเตอร์ โดยเฉพาะในการทดสอบความเร็วของการส่งและการรับข้อมูลระหว่างอุปกรณ์ภายในเครือข่าย Iperf3 ถูกพัฒนาขึ้นโดยกลุ่มวิศวกรรมคอมพิวเตอร์ที่มหาวิทยาลัย California และเป็นซอฟต์แวร์โอเพนซอร์ส ทำให้สามารถใช้งานได้ฟรีและสามารถปรับแต่งตามความต้องการของผู้ใช้ได้

Iperf3 สามารถทำการทดสอบแบบ ทิศทางเดียว หรือ แบบสองทิศทาง ระหว่างคอมพิวเตอร์ สามารถทดสอบความเร็วของเครือข่ายในรูปแบบที่แตกต่างกันได้ เช่น TCP หรือ UDP และมีความสามารถในการตั้งค่าต่าง ๆ เพื่อทดสอบระยะเวลา ขนาดของข้อมูล และพารามิเตอร์อื่น ๆ ตามต้องการของผู้ใช้ ช่วยให้ผู้ดูแลระบบหรือผู้พัฒนาโปรแกรมสามารถทำการทดสอบและปรับปรุงประสิทธิภาพของเครือข่ายได้โดยตรง โดยมีการวัดและรายงานผลการทดสอบที่มีความน่าเชื่อถือและเป็นที่ยอมรับในวงกว้างของอุตสาหกรรมและชุมชนนักพัฒนาโค้ดและการต่างๆทั่วโลก

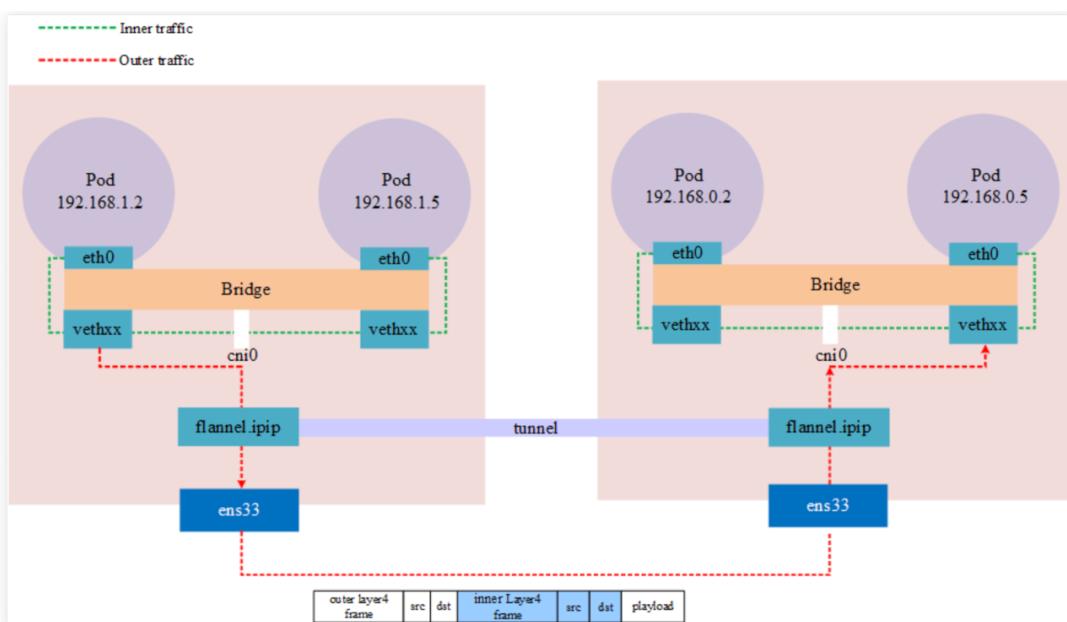


รูปที่ 2.11 หลักการทำงานของ Iperf3 [7]

## 2.1.9 โมเดลเครือข่าย CNI

### 2.1.9.1 เครือข่ายแบบห่อหุ้ม (Overlay Mode)

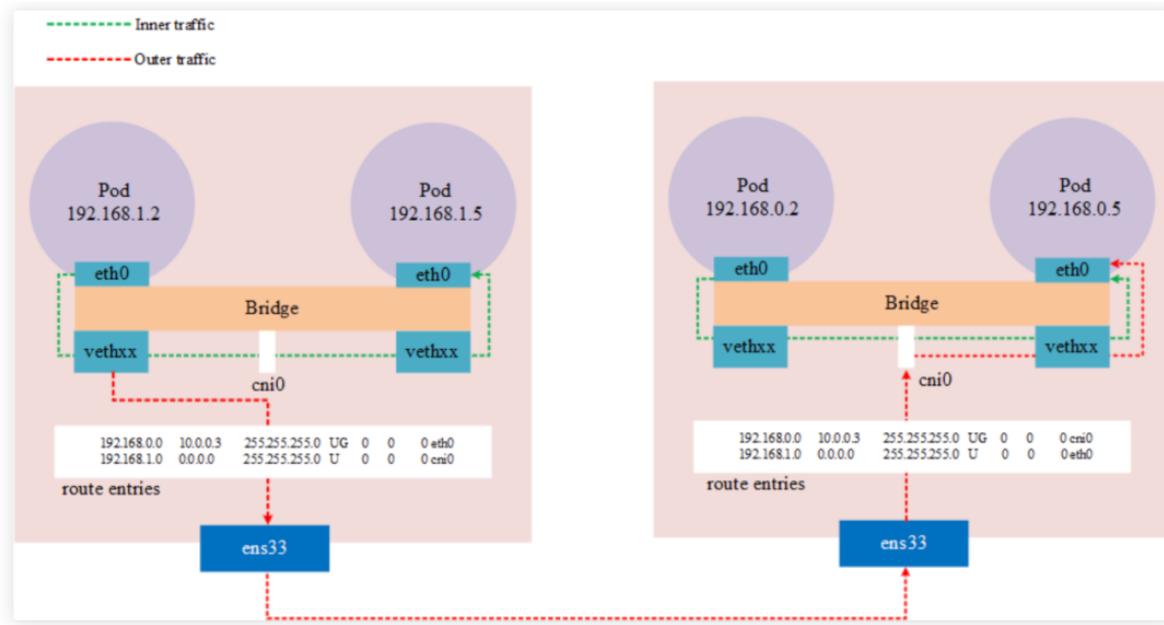
เครือข่ายชั้นทับ (Overlay Network) เป็นเครือข่ายเสมือนที่ชั้นทับบนโครงสร้างพื้นฐานทางกายภาพ ซึ่งช่วยเพิ่มความปลอดภัยและลดการพึ่งพาโครงสร้างทางกายภาพที่อยู่ข้างใต้ ไฮสต์ในเครือข่ายชั้นทับสื่อสารกันผ่านลิงก์เสมือนที่เรียกว่า อุโมงค์ชั้นทับ (Overlay Tunnels) แพ็คเก็ตที่เดินทางผ่านอุโมงค์ชั้นทับจะถูกห่อหุ้มด้วยส่วนหัวภายนอกตามโปรโตคอลชั้นทับที่เลือก เช่น VXLAN หรือ GRE ในขณะที่เทคโนโลยีชั้นทับลดการพึ่งพาโครงสร้างพื้นฐานที่อยู่ข้างใต้ แต่ก็อาจทำให้การติดตามแพ็คเก็ตท้าทายมากขึ้นและอาจนำไปสู่การลดthonประสิทธิภาพการทำงาน



รูปที่ 2.12 โหมด Overlay บน CNI [8]

### 2.1.9.2 เครือข่ายที่ไม่มีการห่อหุ้ม (Underlay Mode)

เครือข่าย Underlay เป็นเพียงเลเยอร์ Datalink หรือเลเยอร์-3 (IP) สำหรับการส่งต่อแพ็คเก็ต เครือข่าย Underlay จัดทำการใช้มต่อการกำหนดเส้นทางดังเดิมระหว่างไฮสต์ต่างๆ ในกลุ่ม ซึ่งโดยทั่วไปจะต้องใช้ BGP โดยที่ไฮสต์จะทำหน้าที่เป็นเพียร์ BGP และแบ่งปันข้อมูลการกำหนดเส้นทางระหว่างกันเพื่อร่วบกันกำหนดเส้นทางระหว่างไฮสต์โดยไม่จำเป็นต้องมีการห่อหุ้มเครือข่าย Underlay เป็นพื้นฐานสำหรับเครือข่ายชั้นทับ เครือข่ายชั้นทับจะสร้างเครือข่ายเสมือนส่วนบนของเครือข่าย Underlay ที่มีอยู่ เครือข่ายชั้นทับสามารถใช้เพื่อเพิ่มประสิทธิภาพและความสามารถในการปรับขนาดของเครือข่าย ตลอดจนเพิ่มความปลอดภัยของเครือข่าย



รูปที่ 2.13 โหมด Underlay บน CNI [8]

## 2.1.10 CNI ที่ใช้ในโครงการ

### 2.1.10.1 Calico

Calico เป็นโซลูชันเครือข่ายและความปลอดภัยเครือข่ายโอเพนซอร์สที่ได้รับความนิยมอย่างกว้างขวางสำหรับ Kubernetes, Virtual Machines (VM) และ Bare-Metal Workloads Calico ให้บริการหลักสองประการสำหรับแอปพลิเคชัน Cloud Native:

- การเข้ามต่อเครือข่ายระหว่างภาระงาน
- การบังคับใช้นโยบายความปลอดภัยเครือข่ายระหว่างภาระงาน

สถาปัตยกรรมที่ยืดหยุ่นของ Calico รองรับตัวเลือกการใช้งานที่หลากหลาย โดยใช้ส่วนประกอบและเทคโนโลยีแบบโมดูลาร์ รวมถึง:

- Calico รองรับเทคโนโลยี Data Plane หลากหลายประการ เช่น eBPF, Standard Linux, Windows HNS และ VPP ผู้ใช้สามารถเลือกเทคโนโลยี Data Plane ที่เหมาะสมกับสภาพแวดล้อมการใช้งานของตนเอง
- Calico รองรับฟีเจอร์นโยบายความปลอดภัยเครือข่าย Kubernetes ทั้งหมด นอกจากนี้ Calico ยังรองรับนโยบายความปลอดภัยเครือข่ายของ Calico ซึ่งมีฟีเจอร์ที่หลากหลายกว่านโยบายความปลอดภัยเครือข่าย Kubernetes
- Calico ใช้ eBPF เพื่อปรับแต่งการใช้งานบริการ Kubernetes ให้มีประสิทธิภาพสูงสุด

- Calico สามารถสนับสนุนเข้ากับเซิร์ฟเวอร์ API Kubernetes เพื่อใช้ในการจัดการการกำหนดค่า Calico และนโยบายความปลอดภัยเครือข่าย Calico
- Calico รองรับทั้งเครือข่ายแบบ Non-Overlay และเครือข่ายแบบ Overlay (ผ่าน IPIP หรือ VXLAN) ใน การใช้งานทั้งแบบคลาวด์สาธารณะและภายในองค์กร
- Calico มี CNI Plugin สำหรับ Kubernetes ซึ่งช่วยให้สามารถจัดเตรียมเครือข่าย Pod และการจัดการที่อยู่ IP ที่มีประสิทธิภาพสูงสุด
- Calico มีปลั๊กอิน Neutron ML2 ซึ่งช่วยให้สามารถจัดเตรียมเครือข่าย VM สำหรับ OpenStack
- Calico มี BGP Routing Stack ซึ่งช่วยให้สามารถโฆษณาเส้นทางสำหรับที่อยู่ IP ของภาระงานและบริการ ไปยังโครงสร้างพื้นฐานเครือข่ายภายในภาระ

Overlay จะใช้เป็น VXLAN, IP in IP or None สามารถเปลี่ยนได้

Underlay จะใช้เป็น BGP ไม่สามารถเปลี่ยนได้

### 2.1.10.2 Flannel

Flannel ทำงานโดยรันเอเจนต์ขนาดเล็กที่เรียกว่า flanneld บนแต่ละโภสต์ โดยมีหน้าที่รับผิดชอบจัดสรรการเช่าซับเน็ตให้กับแต่ละโภสต์จากพื้นที่ที่อยู่ที่กำหนดค่าไว้ล่วงหน้าที่มีขนาดใหญ่กว่า Flannel ใช้ Kubernetes API หรือ etcd โดยตรงเพื่อเก็บข้อมูลการกำหนดค่าเครือข่าย ซับเน็ตที่จัดสรร และข้อมูลเสริมอื่นๆ (เช่น IP สาธารณะของโภสต์) แพ็กเกจจะถูกส่งต่อโดยใช้กลไกแบ็กเอนด์แบบหนึ่ง ซึ่งรวมถึง VXLAN และการผสานรวมกับคลาวด์ต่างๆ

แพลตฟอร์มต่างๆ เช่น Kubernetes ตั้งสมมติฐานว่าคอนเทนเนอร์แต่ละตัวจะมี IP เอกพาะที่สามารถกำหนดเส้นทางได้ภายในกลุ่ม ข้อดีของโมเดลนี้คือช่วยจัดความซับซ้อนของการแมปพอร์ตที่เกิดจากการเชื่อมต่อ IP โภสต์เดียว

Flannel มีหน้าที่รับผิดชอบในการจัดเตรียมเครือข่าย IPv4 เลเยอร์ 3 ระหว่างโหนดต่างๆ ในกลุ่ม Flannel ไม่ได้ควบคุมวิธีการเชื่อมต่อคอนเทนเนอร์กับโภสต์ แต่ควบคุมเฉพาะวิธีการรับส่งข้อมูลระหว่างโภสต์เท่านั้น อย่างไรก็ตาม Flannel มีปลั๊กอิน CNI สำหรับ Kubernetes และแนวทางการรวมเข้ากับ Docker

**Overlay** จะใช้เป็น VXLAN

**Underlay** ในส่วนของ Underlay นั้นไม่มีจะมีแค่ส่วนของ Overlay

#### 2.1.10.3 Cilium

Cilium เป็นโซลูชันเครือข่าย การสังเกตการณ์ และความปลอดภัยที่มี Data Plane แบบ eBPF Cilium จัดเตรียมเครือข่าย Layer 3 แบบแบนเรียบง่ายที่มีความสามารถในการขยายไปยังหลายกลุ่มในโหมดการกำหนดเส้นทางแบบเน็ตฟิวส์โหมดโควอร์เลอร์ Cilium รองรับโปรโตคอล L7 และสามารถบังคับใช้นโยบายเครือข่ายบน L3 - L7 โดยใช้โมเดลความปลอดภัยตามอัตลักษณ์ที่แยกออกจากโครงสร้างพื้นฐานของเครือข่าย

Cilium ใช้การกระจายโหลดбалานซ์สำหรับการรับส่งข้อมูลระหว่างผู้ใช้ และ ไปยังบริการภายนอก และ สามารถแทนที่ Kube-Proxy ได้อย่างสมบูรณ์ โดยใช้ตารางแซฟท์แวร์ที่มีประสิทธิภาพใน eBPF ซึ่งรองรับการปรับขนาดเกือบไม่จำกัด Cilium ยังรองรับฟังก์ชันการทำงานขั้นสูง เช่น เกตเวย์ขาเข้าและขาออกแบบบูรณาการ การจัดการแบบดิจิตอล และเซอร์วิสเมช และมีความสามารถในการตรวจสอบเครือข่ายและความปลอดภัยอย่างลึกซึ้ง

**Overlay** จะใช้เป็น VXLAN ไม่สามารถเปลี่ยนได้

**Underlay** จะใช้เป็น BGP ไม่สามารถเปลี่ยนได้

## 2.2 งานวิจัยที่เกี่ยวข้อง

คณบุญจัดทำได้ทำการศึกษาค้นคว้าข้อมูลงานวิจัยที่สำคัญต่อการดำเนินโครงการเพื่อนำความรู้ผ่านความเข้าใจจากความสำเร็จของงานวิจัยที่เกี่ยวข้อง ที่เป็นประโยชน์ต่อโครงการหรือเป็นประโยชน์ต่อการดำเนินโครงการทั้งข้อมูลอันเป็นประโยชน์และวิธีการแก้ปัญหาของโครงการที่พับเจอกองงานวิจัยโดยผ่านการแนะนำของอาจารย์ที่ปรึกษาของคณบุญจัดทำได้ดังนี้

### 2.2.1 A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications [9]

Kang, Z., An, K., Gokhale, A., & Pazandak, P. N. (2021: 4970-4973) กล่าวว่าการใช้งานเทคโนโลยีคอนเทนเนอร์ (Docker) และการจัดการโดยใช้ Kubernetes (K8s) เพื่อแก้ไขปัญหาความยุ่งยากในการปรับใช้งานบริการที่ใช้ Data Distribution Service (DDS) ในโลกของ Internet of Things (IoT) ที่กระจายอยู่บนพื้นที่ของคลาวด์และเอดจ์ โดยการใช้งานเทคโนโลยีคอนเทนเนอร์ช่วยลดความซับซ้อนและทำให้เป็นไปได้ในการปรับใช้งานและจัดการแอปพลิเคชันและบริการแบบกระจายได้อย่างมีประสิทธิภาพและมีประสิทธิภาพนี้เป็นเรื่องที่มีความสำคัญเนื่องจาก Internet of Things กำลังเติบโตอย่างรวดเร็วและมีความต้องการในการจัดการและปรับใช้งานที่มีประสิทธิภาพ การใช้งานเทคโนโลยีคอนเทนเนอร์และ Kubernetes จึงเป็นทางเลือกที่มีความน่าสนใจสำหรับการจัดการและปรับใช้งาน Internet of Things แบบทั่วไปที่อยู่ในอุตสาหกรรมต่าง ๆ ที่สำคัญ

ผลจากการศึกษาวิจัยจากมุมมองของประสิทธิภาพนั้น การใช้ K8s host-mode ในสถานการณ์ต่าง ๆ ให้ผลลัพธ์ที่ดีที่สุดเนื่องจากสามารถลดความสูญเสียในเครือข่ายเมื่อไม่ได้ แต่ในเวลาเดียวกัน การใช้ host-mode จะไม่สามารถแยกช่องเครือข่ายของคอนเทนเนอร์ได้ ซึ่งอาจทำให้เกิดปัญหาด้านความปลอดภัย พร้อมกับการจัดการพอร์ตที่ซับซ้อนอาจลดความสามารถในการขยายมากของระบบได้ เนื่องจากความเสียหายที่เป็นไปได้ในโหมด host-mode จึงได้รับการพัฒนา CNIs ของ Kubernetes มากมาย เอกสารวิจัยนี้วิเคราะห์ประสิทธิภาพของแอปพลิเคชัน DDS ที่จัดตั้งในคอนเทนเนอร์บนพื้นฐานแพลตฟอร์มแบบสมมติ (ARM+AMD) ของกลุ่ม K8s เมื่อรวมกับ Flannel, WeaveNet, และ Kube-router การใช้งานเทคโนโลยี CNI Flannel-Hostgw หรือ Kube-Router ให้ผลลัพธ์ประสิทธิภาพที่ดีกว่าในการสื่อสารแบบยูนิกาสต์ที่เชื่อมต่อได้ การเปิดใช้งาน DDS BestEffort QoS นั้นเป็นวิธีที่ดีในการปรับปรุงเวลาความสามารถในการตอบสนองและการใช้งาน CNIs ประเภท L2 มีการปรับปรุงประสิทธิภาพที่ดีกว่า Flannel-Hostgw และ Kube-router แม้ว่า WeaveNet จะเป็น CNI ที่จำลองแบบ L2 เพียงอย่างเดียว แต่ความเสียหายที่เกิดจากการครอบคลุมด้วย VXLAN อาจทำให้เกิดผลกระทบต่อประสิทธิภาพของการใช้งานแบบมัลติแคสต์เมื่อทำงานในกลุ่มน้ำดเล็ก

แต่มีความสามารถในเรื่องของความขยายได้ดีกว่า Flannel-Hostgw และ Kube-router เมื่อใช้งานในกรณี การสมาชิกแบบหลายคนขนาดใหญ่ เพื่อป้องกันจากการโจมตีที่เป็นอันตรายต่อแพคเก็ตของ DDS ส่วนขยาย ความปลอดภัยของ DDS นั้นให้ความคุ้มครองที่ยืดหยุ่นและละเอียดถี่ถ้วนกว่าวิธีการ IPSec ที่รองรับโดย Flannel และ WeaveNet ในกรณีการสื่อสารแบบบูนิค่าสต์ ในการสื่อสารแบบ 1 ต่อ 1 DDS Security มี ประสิทธิภาพดีกว่า Flannel-IPSec และ WeaveNet-Encrypt เมื่อขนาดของข้อมูลน้อยกว่า MTU แต่จะมี ความเสียหายมากขึ้นเมื่อส่งตัวอย่างข้อมูลขนาดใหญ่ อย่างไรก็ตามในกรณีการสมาชิกแบบหลายคน DDS Security เป็นวิธีการที่เหมาะสมที่สุดในการบรรลุประสิทธิภาพและขนาดของระบบที่ใหญ่ สังเกตได้ว่า Flannel-IPsec ดำเนินการดีกว่า WeaveNet-Encrypt ในการสื่อสารแบบบูนิค่าสต์ นอกจากนี้ ปลั๊กอินของ DDS Security มีประสิทธิภาพดีกว่าวิธีการ IPSec ในกรณีการสมาชิกแบบหลายคน ทำให้มีประสิทธิภาพและ ขนาดของระบบที่ดีกว่า

### 2.2.2 Overview of Kubernetes Cni Plugins Performance [10]

atkus, A., & Tamulevičius, A. (2022: 13-23) กล่าวว่าสถาปัตยกรรมของไมโครเซอร์วิสต้องการ จำนวนคอนเทนเนอร์ที่สูงที่ต้องได้รับการจัดการด้วยเครื่องมือต่าง ๆ เช่น Kubernetes, Docker Swarm, Apache Mesos การจัดการคอนเทนเนอร์ เช่น Kubernetes และ Apache Mesos ใช้ Container Network Interface (CNI) สำหรับการดำเนินการเครือข่ายระหว่างคอนเทนเนอร์ CNI นำเสนอปลั๊กอินที่หลากหลายซึ่ง แก้ไขปัญหาการเครือข่ายเช่นเดียวกันในวิธีที่แตกต่างกัน และพลิเคชันที่พื้นฐานบนสถาปัตยกรรมไมโครเซอร์วิสสามารถถูกจัดการบนเซิร์ฟเวอร์ทางกายภาพหรือเพื่อความยืดหยุ่นและความคุ้มค่าสามารถถูกจัดการ ในสภาพแวดล้อมศูนย์ข้อมูล โดยได้ทดสอบประสิทธิภาพของเลขบล็อกอิน CNI ก่อนที่สภาพแวดล้อมเครื่องจำลอง เสมือน และเปรียบเทียบกับประสิทธิภาพของเลขบล็อกอิน CNI ในสภาพแวดล้อมทางกายภาพ ในท้ายที่สุด ผลลัพธ์ทั้งสองถูกเปรียบเทียบกับผลลัพธ์ของปั๊มน้ำมาร์คเซิร์ฟเวอร์ที่มีโครงสร้าง TCP และ HTTP ถูกเลือก เนื่องจากเป็นโปรโตคอลที่ได้รับความนิยมในแอปพลิเคชันไมโครเซอร์วิสในปัจจุบัน ส่วนที่เหลือของงานวิจัยนี้ จัดเรียงอย่างเร้าใจและสนับสนุน ส่วนที่เกี่ยวข้องกับงานวิจัยนำเสนอผลการประเมินประสิทธิภาพของเลขบล็อกอิน Kubernetes CNI สภาพแวดล้อมทดสอบถูกอธิบายในส่วนของการตั้งค่าสภาพแวดล้อมในศูนย์ข้อมูล ในส่วน ของการวิเคราะห์ผลการทดสอบจะนำเสนอผลการทดสอบ ในส่วนสุดท้ายเป็นสรุปผลสุดท้ายและงานที่จะทำ ในอนาคต

ผลจากการดำเนินการวิเคราะห์ได้สร้างกลุ่ม Kubernetes ที่ประกอบด้วย 3 โนนดในศูนย์ข้อมูล 3 แห่งที่แยกกันด้วยระยะทางประมาณ 10 กิโลเมตรและเชื่อมต่อกันโดยใช้เชื่อมต่อไฟเบอร์ที่กำหนดไว้เป็นพิเศษ ผลการทดลองยืนยันว่าปลั๊กอิน CNI ทั้งหมดในกรณีของโปรโตคอล TCP และ HTTP ในสภาพแวดล้อมเครือข่าย จำลองเสมือนมีประสิทธิภาพที่มากกว่าในสภาพแวดล้อมทางกายภาพเนื่องจากการเพิ่มภาระการประมวลผล เพิ่มเติมที่เกิดจากไฮเปอร์ไวเซอร์ การใช้ค่า MTU ที่สูงขึ้น (9,000B ต่อ 1,500B) เพิ่มประสิทธิภาพของปลั๊กอิน CNI ในสภาพแวดล้อมเครือข่าย จำลองเสมือนเนื่องจากการเพิ่มภาระการประมวลผลของข้อมูลโปรโตคอลน้อยลง อย่างไรก็ตามถึงแม้ว่าปลั๊กอิน CNI ใน Kubernetes จะมีความสามารถในการจัดการไฟเบอร์ที่ซับซ้อน แต่การสูญเสียประสิทธิภาพในการรับส่งข้อมูลสำหรับปลั๊กอินที่ใช้กันอยู่ยังแพร่หลายไม่มี ความสำคัญมาก และสภาพแวดล้อมเครือข่าย จำลองเสมือนมีความยืดหยุ่นสูงกว่า ในงานวิจัยในอนาคต เราจะ ศึกษาและเปรียบเทียบประสิทธิภาพของปลั๊กอิน CNI ใน Kubernetes ในสภาพแวดล้อมเครือข่าย จำลองเสมือน อื่น ๆ (เช่น KVM, Xen, Hyper-V) และดูว่าประสิทธิภาพของปลั๊กอิน CNI ขึ้นอยู่กับลักษณะของเครือข่าย จำลอง เสมือน

## บทที่ 3

### วิธีการดำเนินงาน

#### 3.1 เครื่องมือที่ใช้ในโครงการ

##### 3.1.1 ฮาร์ดแวร์ที่ใช้

- เครื่องคอมพิวเตอร์ Acer Nitro AN515-45

ทำหน้าที่เป็นเครื่องคอมพิวเตอร์ที่ไว้ใช้สำหรับควบคุมโหนด Control และ โหนด Worker ที่ใช้ในการทดลอง

- Switch WS-C3650-24PS-L (Cisco Catalyst 3650 24 Port PoE 4x1G Uplink LAN Base)

[24]

ทำหน้าที่เป็นศูนย์กลางเชื่อมต่อระบบ ให้อุปกรณ์ Raspberry Pi 4 และ Lenovo ThinkSystem SR530 Server เพื่อรับส่งข้อมูลในรูปแบบ IP ระหว่างอุปกรณ์แต่ละตัวในระบบเข้าด้วยกัน ผ่านสายแลน

- Raspberry Pi 4 Model B

ทำหน้าที่เป็นเครื่องคอมพิวเตอร์สำหรับเป็นโหนด Control และโหนด Worker, โดยการใช้ Microk8s ในการกำหนดค่า

- Lenovo ThinkSystem SR530 Server [23]

ทำหน้าที่เป็นเซิร์ฟเวอร์ในการสร้างเครื่องจำลองสมือนี้ที่เป็นเครื่องคอมพิวเตอร์สำหรับ โหนด Control และ โหนด Worker, โดยการใช้ Microk8s ในการกำหนดค่า

##### 3.1.2 ซอฟต์แวร์ที่ใช้

- Docker Container

ใช้สำหรับรันแอปพลิเคชันหรือบริการบนเซิร์ฟเวอร์ใดๆ ก็ได้ โดยไม่ต้องติดตั้งแอปพลิเคชัน หรือบริการนั้นบนเซิร์ฟเวอร์ทั้งหมด

### - Container Network Interface (CNI)

ใช้สำหรับเชื่อมต่อคอนเนนเนอร์เข้าด้วยกันเพื่อที่สามารถทำการสื่อสารกันระหว่างและส่งข้อมูลไปยังคอนเนนเนอร์ตัวอื่นๆได้

### - Microk8s

Microk8s เป็นเวอร์ชันที่เล็กกว่าของ Kubernetes ใช้สำหรับการจัดเก็บและจัดการคอนเนนเนอร์ อีกทั้งยังใช้เพื่อปรับใช้ จัดการ และปรับขนาดแอปพลิเคชันที่รันบนคอนเนนเนอร์

### - Iperf3

เป็นซอฟแวร์ที่ใช้เพื่อวัดแบบดิจิตที่สูงสุดที่พร้อมใช้งาน หน่วงเวลา และอัตราเฟรม อีกทั้งยังใช้เป็นเครื่องมือทดสอบประสิทธิภาพของเครือข่าย CNI

## 3.2 ขั้นตอนการดำเนินงาน

### 3.2.1 ขั้นตอนการติดตั้งระบบปฏิบัติการบน Raspberry Pi 4

#### 3.2.1.1 ดาวน์โหลดและติดตั้ง Raspberry Pi Imager

**Install Raspberry Pi OS using Raspberry Pi Imager**

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)

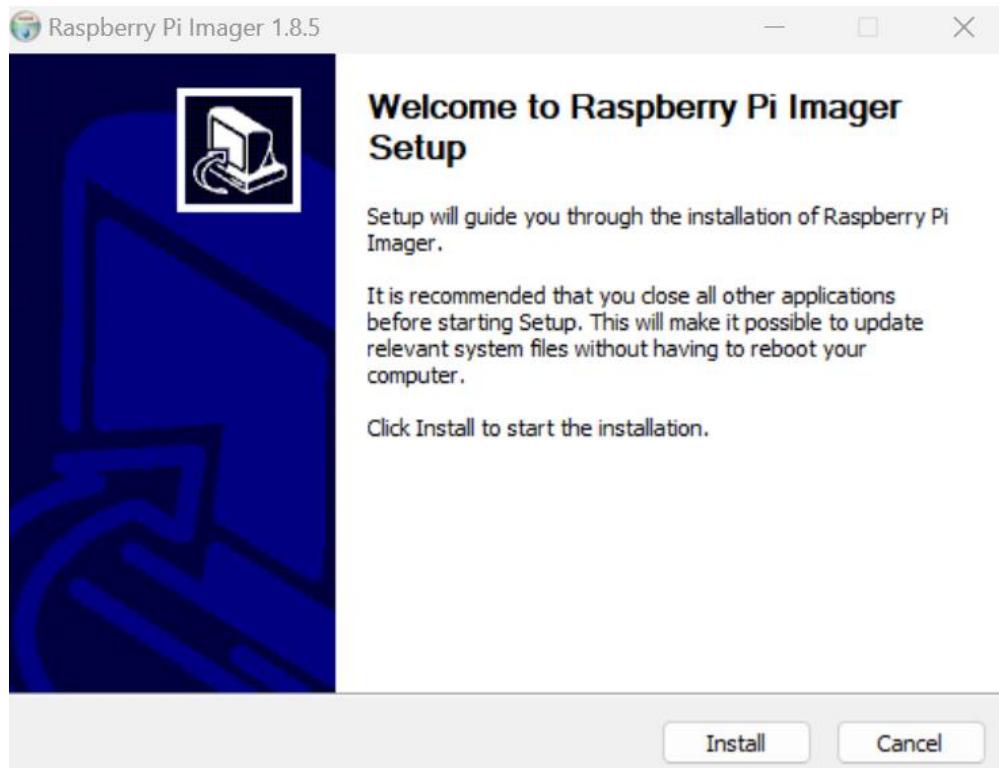
[Download for macOS](#)

[Download for Ubuntu for x86](#)

To install on **Raspberry Pi OS**, type  
`sudo apt install rpi-imager`  
in a Terminal window.

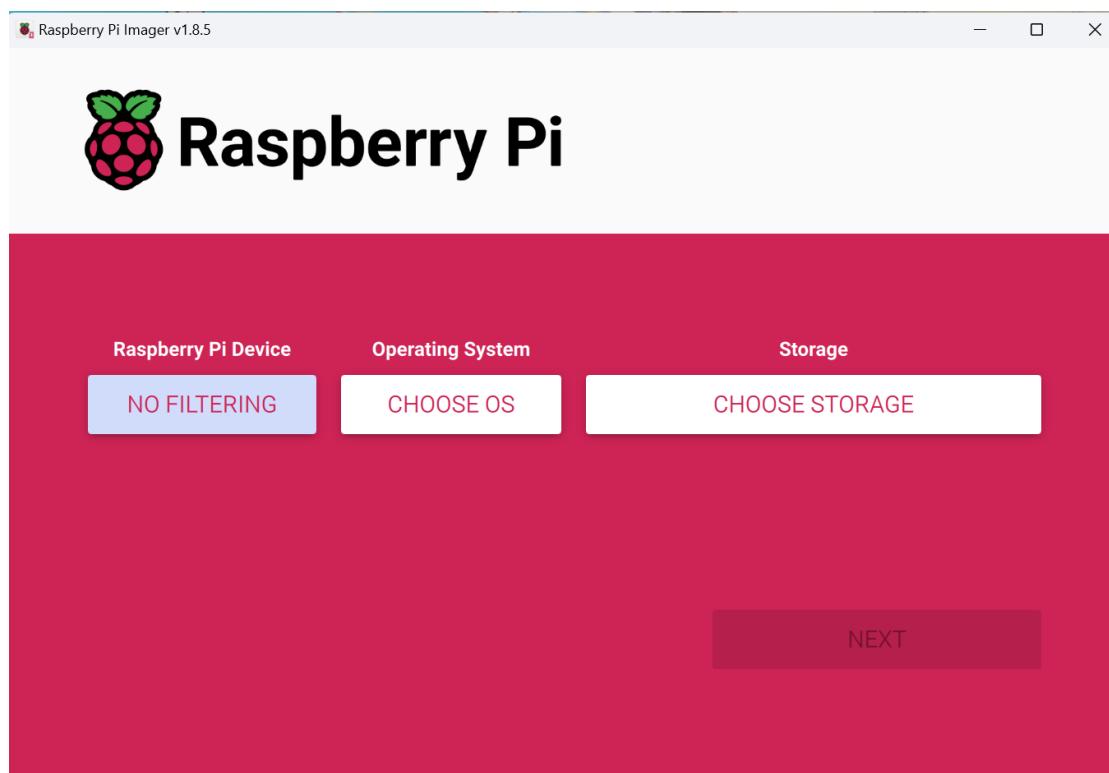


รูปที่ 3.1 ดาวน์โหลด Raspberry Pi Imager



รูปที่ 3.2 ติดตั้ง Raspberry Pi Imager

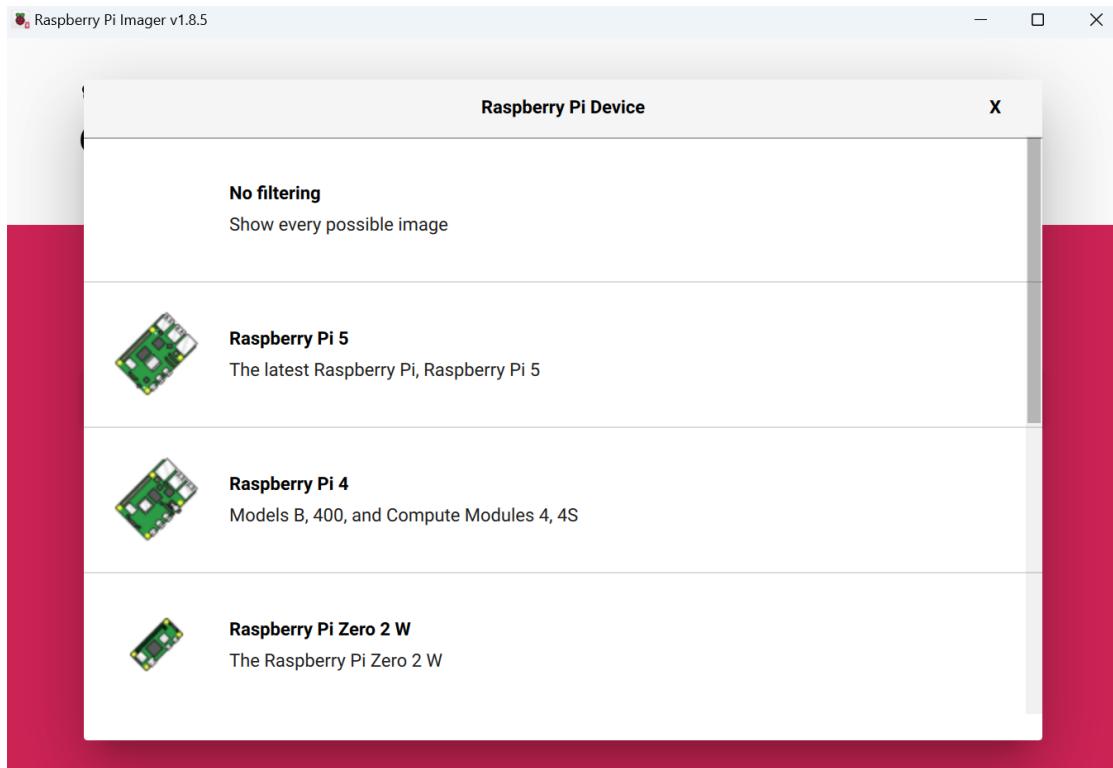
### 3.2.1.2 ติดตั้งระบบปฏิบัติการบน Raspberry Pi 4



รูปที่ 3.3 หน้าโปรแกรม Raspberry Pi Imager

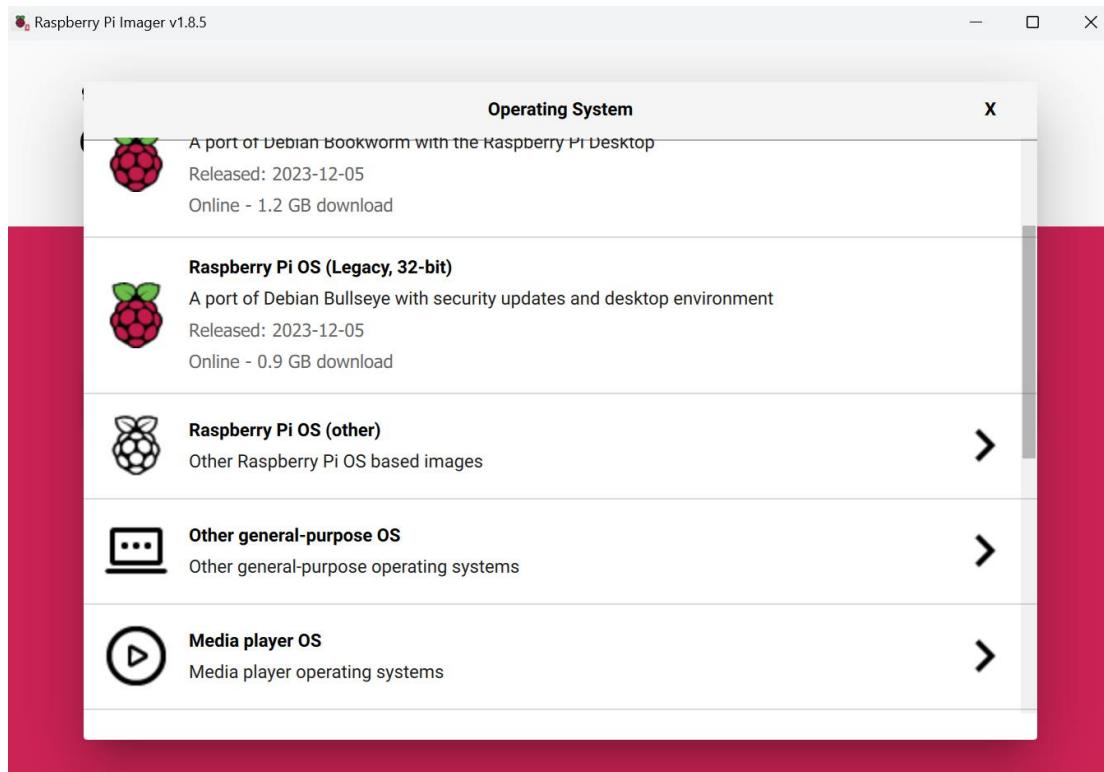
วิธีการใช้โปรแกรม Raspberry Pi Imager ในการติดตั้งระบบปฏิบัติการบน Raspberry Pi 4 จำเป็นต้องใช้ SD Card 32 GB C10 เป็นขั้นต่ำ

Raspberry Pi Device คือ อุปกรณ์ของ Raspberry Pi ที่ต้องการนำระบบปฏิบัติการไปใช้ Operating System คือ ระบบปฏิบัติการของ Raspberry Pi Imager ที่สามารถเลือกใช้ได้ Storage คือ อุปกรณ์เก็บข้อมูลที่ต้องจัดเก็บบันทึก

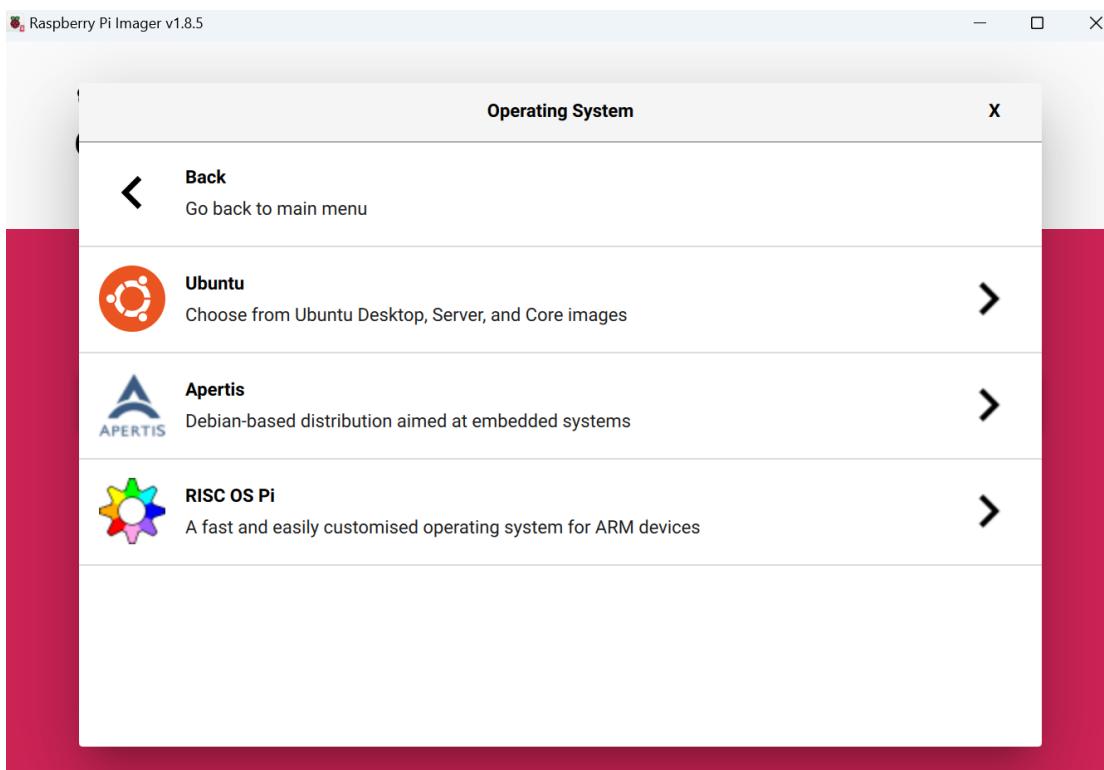


รูปที่ 3.4 อุปกรณ์ของ Raspberry Pi Imager

อุปกรณ์ของ Raspberry Pi Imager ณ เวอร์ชัน V1.8.5 รองรับถึง Raspberry Pi 5 ซึ่งสามารถเลือกได้ตามต้องการ ในการทดสอบได้เลือกใช้ Raspberry Pi 4 ตามอุปกรณ์ที่ได้นำมาใช้งาน

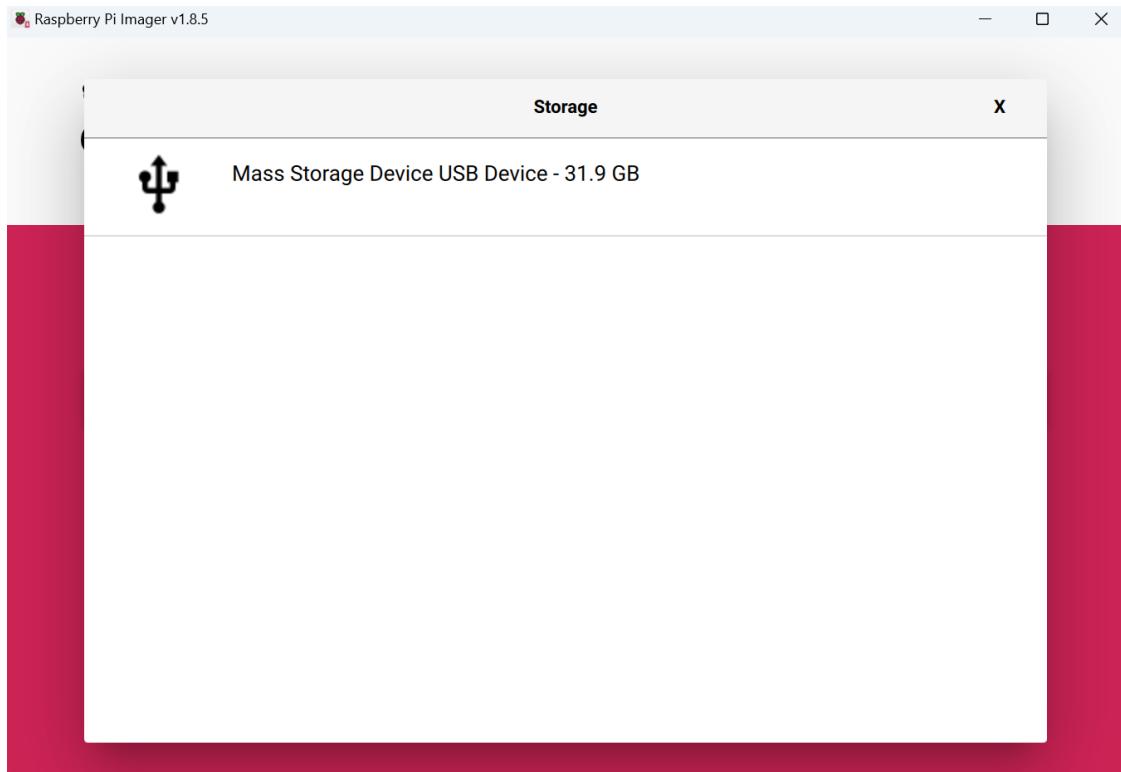


รูปที่ 3.5 ระบบปฏิบัติการของ Raspberry Pi Imager



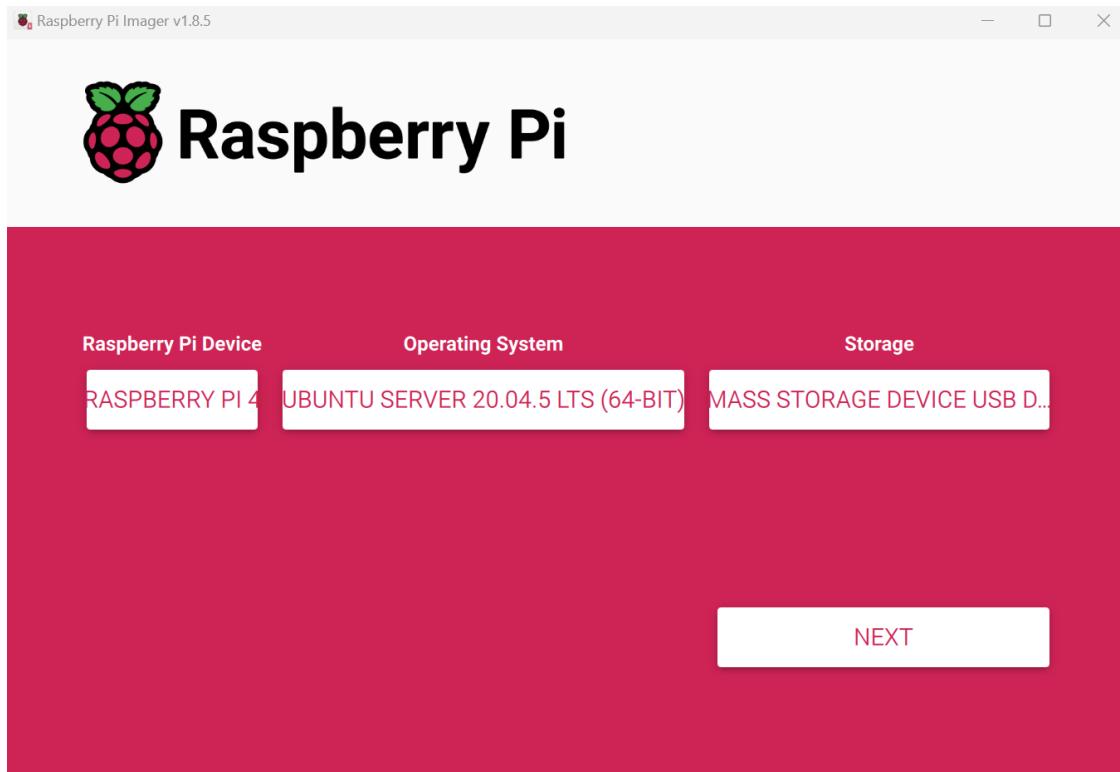
รูปที่ 3.6 ระบบปฏิบัติการของ Other General-Purpose

ระบบปฏิบัติการของ Raspberry Pi Imager มีให้ใช้หลากหลาย ใน การทดสอบได้ทำการเลือกใช้ Ubuntu Server 20.04.5 LTS (64 bit)



รูปที่ 3.7 อุปกรณ์จัดเก็บข้อมูลของ Raspberry Pi Imager

อุปกรณ์เก็บข้อมูลของ Raspberry Pi Imager จะแสดงก็ต่อเมื่อมี อุปกรณ์ที่เชื่อมต่ออยู่กับคอมพิวเตอร์ของคุณ เลือกใช้ให้ถูกต้องและดูพื้นที่จัดเก็บของอุปกรณ์ที่นำมาใช้งาน หากไม่พอใช้จะทำให้เกิดการใช้งาน Raspberry Pi Imager ไม่สมบูรณ์และใช้งานไม่ได้



รูปที่ 3.8 การตั้งค่าของโปรแกรม Raspberry Pi Imager

### 3.2.2 ขั้นตอนการติดตั้งระบบปฏิบัติการบนเครื่องจำลองเสมือน

สำหรับการติดตั้งระบบปฏิบัติการและกำหนดประสิทธิภาพของเครื่องจำลองเสมือนมีขั้นตอนดังต่อไปนี้

**Type:** Linux

**Version:** 6.x-2.6 Kernel

**ISO image:** ubuntu-20.04.6-live-server-amd64.iso

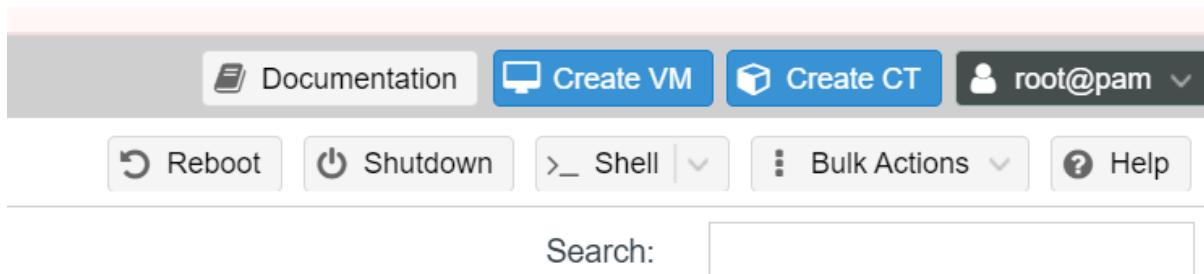
**Sockets:** 1 **Cores:** 4

**Memory (MiB):** 4096

การติดตั้งระบบปฏิบัติการบนเครื่องจำลองเสมือนผ่าน Proxmox Virtual Environment

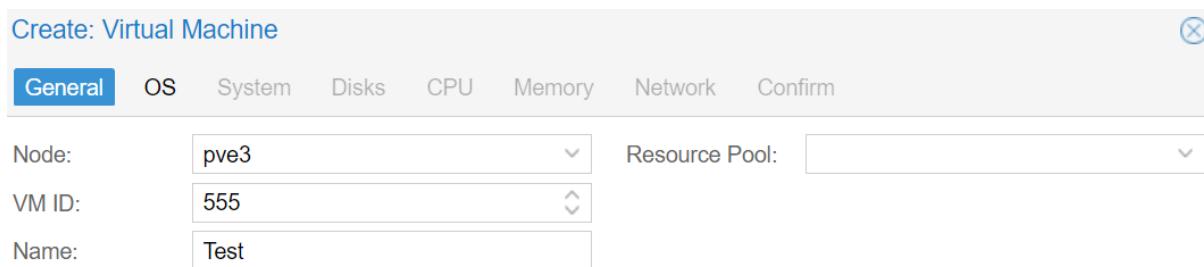
7.4.3 มีขั้นตอนดังนี้

### 3.2.2.1 สร้างเครื่องจำลองเสมือน (Create VM)



รูปที่ 3.9 การสร้างเครื่องจำลองเสมือนบน Proxmox

### 3.2.2.2 กำหนดรายละเอียดทั่วไป



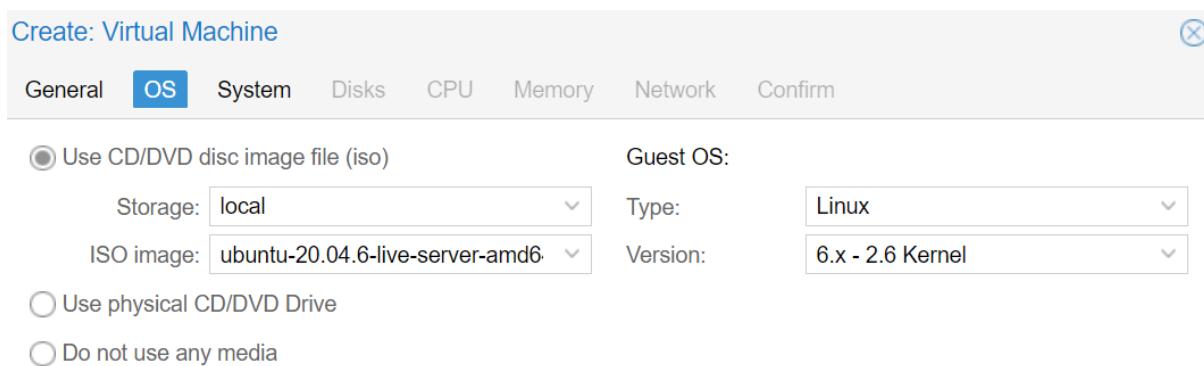
รูปที่ 3.10 รายละเอียดเครื่องจำลองเสมือน

**Node:** ชื่อเซิร์ฟเวอร์ที่ต้องการสร้างเครื่องจำลองเสมือน

**VM ID:** ID เฉพาะของเครื่องจำลองเสมือน

**Name:** ชื่อเครื่องจำลองเสมือน

### 3.2.2.3 กำหนดระบบปฏิบัติการ



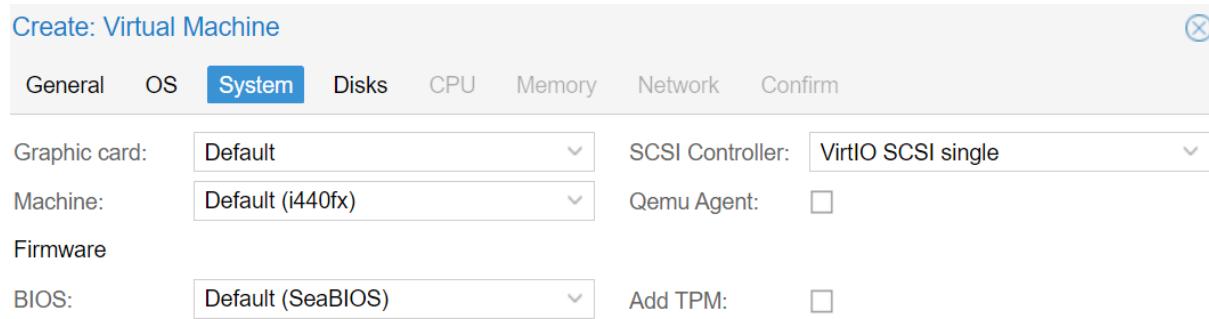
รูปที่ 3.11 รายละเอียดระบบปฏิบัติการเครื่องจำลองเสมือน

**Type:** Linux

**Version:** 6.x-2.6 Kernel

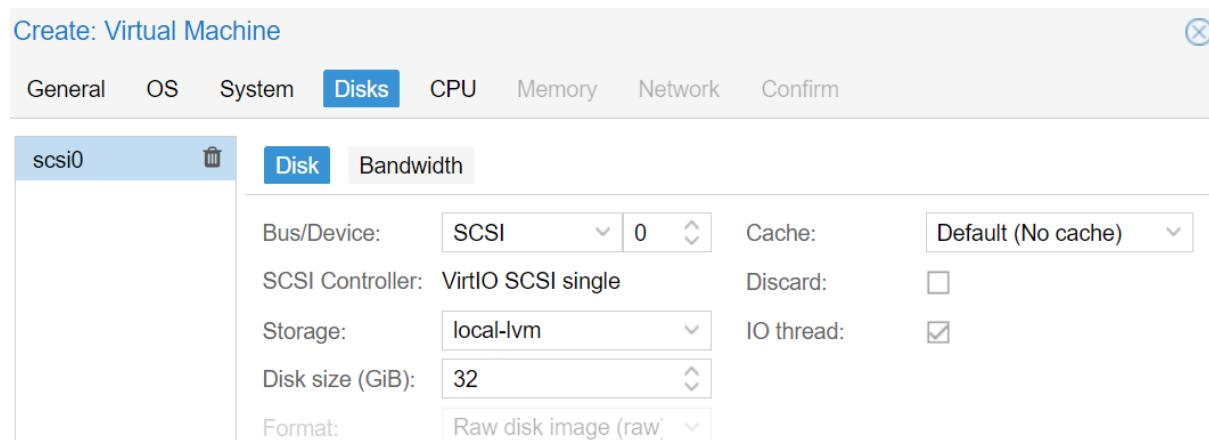
**ISO image:** ubuntu-20.04.6-live-server-amd64.iso

3.2.2.4 กำหนดระบบเป็นค่าเริ่มต้นทั้งหมด



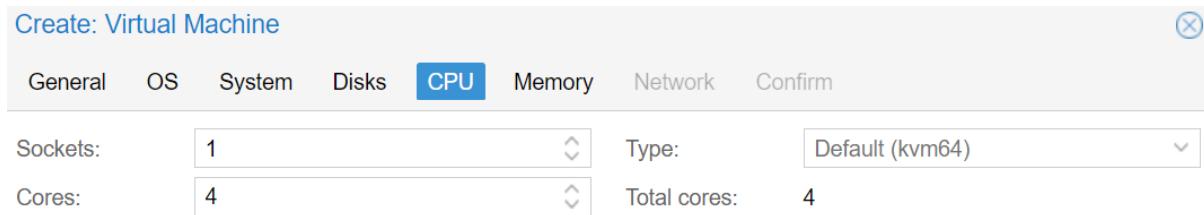
รูปที่ 3.12 รายละเอียดการกำหนดระบบ

3.2.2.5 กำหนดดิสก์เป็นค่าเริ่มต้นทั้งหมด



รูปที่ 3.13 รายละเอียดการกำหนดดิสก์

### 3.2.2.6 กำหนดหน่วยประมวลผลกลาง



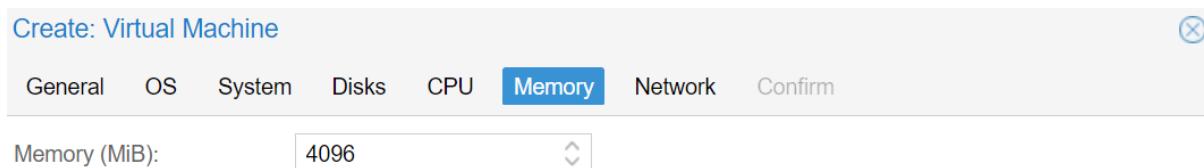
รูปที่ 3.14 รายละเอียดการกำหนดหน่วยประมวลผลกลาง

**Sockets:** 1

**Cores:** 4

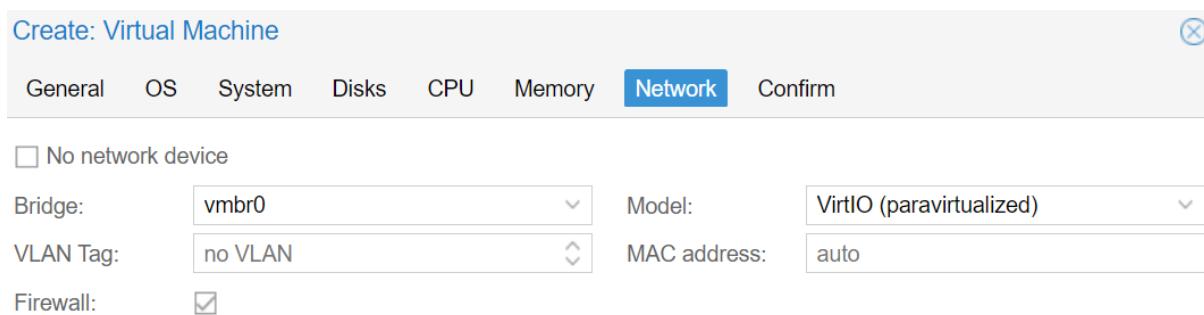
**Type:** ค่าเริ่มต้น

### 3.2.2.7 กำหนดหน่วยความจำ 4096 MiB

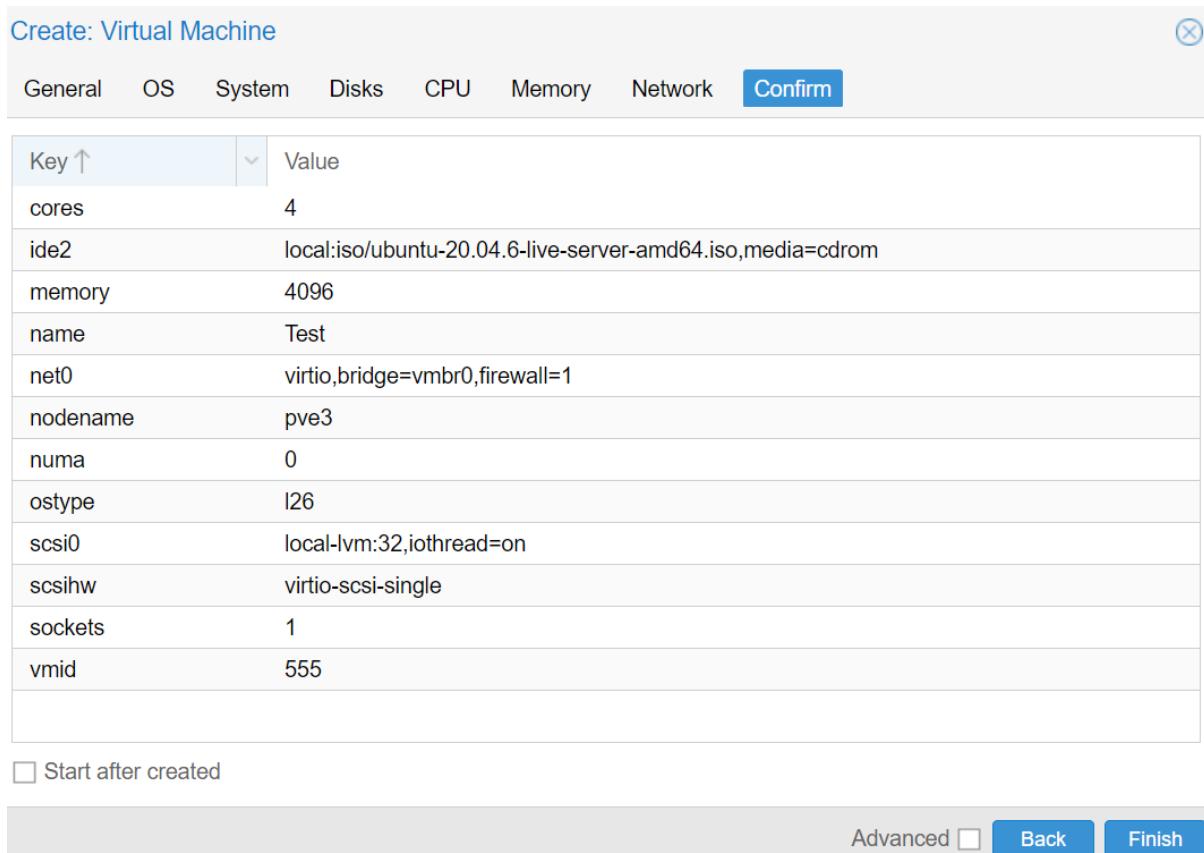


รูปที่ 3.15 รายละเอียดการกำหนดหน่วยความจำ

### 3.2.2.8 กำหนดเครือข่ายเป็นค่าเริ่มต้นทั้งหมด



รูปที่ 3.16 รายละเอียดการกำหนดเครือข่าย



รูปที่ 3.17 รายละเอียดรูปแบบปฏิบัติการบนเครื่องจำลองเสมือน

### 3.2.3 ขั้นตอนการติดตั้ง MicroK8s บนระบบปฏิบัติการ Linux

#### 3.2.3.1 ติดตั้ง MicroK8s

```
sudo snap install microk8s --classic
```

```
Last login: Thu Oct  5 07:21:55 2023 from 192.168.50.127
user@control:~$ sudo snap install microk8s --classic
[sudo] password for user:
microk8s (1.27/stable) v1.27.5 from Canonical✓ installed
user@control:~$ █
```

รูปที่ 3.18 วิธีติดตั้ง MicroK8s

### 3.2.3.2 ตรวจสอบการทำงานของ MicroK8s

microk8s status

```
user@control:~$ microk8s status
Insufficient permissions to access MicroK8s.
You can either try again with sudo or add the user user to the 'microk8s' group:
  sudo usermod -a -G microk8s user
  sudo chown -R user ~/.kube

After this, reload the user groups either via a reboot or by running 'newgrp microk8s'.
user@control:~$ sudo usermod -a -G microk8s user
user@control:~$ newgrp microk8s
user@control:~$
```

รูปที่ 3.19 ผลลัพธ์เมื่อ MicroK8s ไม่ทำงาน

### 3.2.3.3 หากมีการแสดงผลลัพธ์เหมือนรูปที่ 3.10 ให้ใช้คำสั่งตามที่ระบุไว้ในเอกสารพูดหลังจากนั้นตรวจสอบอีกรอบ

```
user@control:~$ microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
addons:
  enabled:
    dns          # (core) CoreDNS
    ha-cluster   # (core) Configure high availability on the current node
    helm         # (core) Helm - the package manager for Kubernetes
    helm3        # (core) Helm 3 - the package manager for Kubernetes
  disabled:
    cert-manager # (core) Cloud native certificate management
    community    # (core) The community addons repository
    dashboard    # (core) The Kubernetes dashboard
    host-access  # (core) Allow Pods connecting to Host services smoothly
    hostpath-storage # (core) Storage class; allocates storage from host directory
    ingress      # (core) Ingress controller for external access
    kube-ovn     # (core) An advanced network fabric for Kubernetes
    mayastor     # (core) OpenEBS MayaStor
    metallb     # (core) Loadbalancer for your Kubernetes cluster
    metrics-server # (core) K8s Metrics Server for API access to service metrics
    minio        # (core) MinIO object storage
    observability # (core) A lightweight observability stack for logs, traces and metrics
    prometheus   # (core) Prometheus operator for monitoring and logging
    rbac         # (core) Role-Based Access Control for authorisation
    registry     # (core) Private image registry exposed on localhost:32000
    storage      # (core) Alias to hostpath-storage add-on, deprecated
```

รูปที่ 3.20 สถานะ MicroK8s

3.2.3.4 ใช้คำสั่งตั้งค่านามแฝงให้กับคำสั่ง microk8s.kubectl ให้เป็น kubectl แทน เพื่อให้สามารถเรียกใช้คำสั่ง kubectl ได้โดยไม่จำต้องพิมพ์ microk8s. นำหน้า

```
sudo snap alias microk8s.kubectl kubectl
```

### 3.2.3.5 ตรวจสอบสถานะของโหนด

```
kubectl get nodes
```

```
user@control:~$ sudo snap alias microk8s.kubectl kubectl
Added:
- microk8s.kubectl as kubectl
user@control:~$ kubectl get nodes
NAME     STATUS   ROLES      AGE    VERSION
control   Ready    <none>   88s   v1.27.5
```

รูปที่ 3.21 ตั้งค่านามแฝงและตรวจสอบสถานะของโหนด

### 3.2.3.6 ติดตั้ง MicroK8s เพิ่มอีก 7 โหนด

#### 3.2.4 ขั้นตอนการเพิ่มโหนดเข้าไปในกลุ่มของ Kubernetes

##### 3.2.4.1 ใช้คำสั่งต่อไปนี้เพื่อสร้าง Token ให้โหนด Worker ใช้เพื่อเข้าสู่กลุ่มของโหนด

Control Plane

```
sudo microk8s.add-node
```

```
user@control:~$ sudo microk8s.add-node
From the node you wish to join to this cluster, run the following:
microk8s join 192.168.50.209:25000/cde6b9e2ffbdd29ed53483aeb476f134/8f78ef239056

Use the '--worker' flag to join a node as a worker not running the control plane, e.g:
microk8s join 192.168.50.209:25000/cde6b9e2ffbdd29ed53483aeb476f134/8f78ef239056 --worker

If the node you are adding is not reachable through the default interface you can use one of the following:
microk8s join 192.168.50.209:25000/cde6b9e2ffbdd29ed53483aeb476f134/8f78ef239056
user@control:~$
```

รูปที่ 3.22 Token สำหรับโหนด Worker

### 3.2.4.2 นำ Token ที่ได้ไปใส่ในหน้า Worker ที่ต้องการเพิ่มเข้าไปในกลุ่ม

```
user@workervm1:~$ microk8s join 192.168.50.209:25000/bee32b9cf178635e24d711ea69627f79/8f78ef239056 --worker
[sudo] password for user:
Contacting cluster at 192.168.50.209

The node has joined the cluster and will appear in the nodes list in a few seconds.

This worker node gets automatically configured with the API server endpoints.
If the API servers are behind a loadbalancer please set the '--refresh-interval'
to '0s' in:
    /var/snap/microk8s/current/args/apiserver-proxy
and replace the API server endpoints with the one provided by the loadbalancer in:
    /var/snap/microk8s/current/args/traefik/provider.yaml

user@workervm1:~$ █
```

รูปที่ 3.23 ผลลัพธ์เมื่อหน้า Worker เข้าร่วมกลุ่ม

### 3.2.4.3 ตรวจสอบสถานะหน้าภายนอกกลุ่ม

```
user@control:~$ kubectl get node
NAME      STATUS   ROLES   AGE     VERSION
workervm3  Ready    <none>  4d     v1.27.5
workervm4  Ready    <none>  4d     v1.27.5
workervm2  Ready    <none>  4d     v1.27.5
control    Ready    <none>  4d1h   v1.27.5
worker2    Ready    <none>  4d1h   v1.27.5
worker3    Ready    <none>  4d1h   v1.27.5
worker1    Ready    <none>  91s    v1.27.5
workervm1  Ready    <none>  33s    v1.27.5
user@control:~$ █
```

รูปที่ 3.24 ตรวจสอบสถานะหน้าภายนอกกลุ่ม

### 3.2.5 ขั้นตอนการทดสอบด้วย Iperf3

#### 3.2.5.1 สร้างไฟล์ Yaml สำหรับใช้การปรับใช้ Iperf3

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iperf3-deployment
spec:
  replicas: 8
  selector:
    matchLabels:
      app: iperf3
  template:
    metadata:
      labels:
        app: iperf3
    spec:
      containers:
        - name: iperf3
          image: taoyou/iperf3-alpine
        ports:
          - containerPort: 5201
```

### 3.2.5.2 ใช้งานการปรับใช้ไฟล์ Yaml ที่มีการสร้าง Iperf3 ขึ้นมา

```
kubectl apply -f deployment.yaml
```

```

user@control:~/CNI_Project
user@control:~$ ls
CNI_Project  snap
user@control:~$ cd CNI_Project/
user@control:~/CNI_Project$ ls
README.md  cmdline.txt  commands.txt  iperf3-deployment.yaml  network-config
user@control:~/CNI_Project$ kubectl apply -f iperf3-deployment.yaml
deployment.apps/iperf3-deployment created
user@control:~/CNI_Project$ █

```

รูปที่ 3.25 การปรับใช้ไฟล์ Yaml

### 3.2.5.3 แสดงรายละเอียดของรายการ Pods ในกลุ่มของ Kubernetes

```
kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
iperf3-deployment-5b98b455df-z7hvt	1/1	Running	0	45s	10.1.210.69	workervm2	<none>	<none>
iperf3-deployment-5b98b455df-q8sj6	1/1	Running	0	45s	10.1.123.1	workervm1	<none>	<none>
iperf3-deployment-5b98b455df-nr5zl	1/1	Running	0	45s	10.1.36.5	workervm3	<none>	<none>
iperf3-deployment-5b98b455df-czz72	1/1	Running	0	45s	10.1.189.69	worker2	<none>	<none>
iperf3-deployment-5b98b455df-gzwnq	1/1	Running	0	45s	10.1.119.197	workervm4	<none>	<none>
iperf3-deployment-5b98b455df-f4sl2	1/1	Running	0	45s	10.1.235.129	worker1	<none>	<none>
iperf3-deployment-5b98b455df-f179p	1/1	Running	0	45s	10.1.182.5	worker3	<none>	<none>

รูปที่ 3.26 รายการ Pods ในกลุ่มของ Kubernetes

### 3.2.5.4 สร้างไฟล์ Python สำหรับการทดสอบ

```

import os
import subprocess
import time

def run_command(command, output_file):
    try:
        with open(output_file, "a") as file:
            subprocess.run(command, shell=True, check=True, stdout=file, stderr=file)
    except subprocess.CalledProcessError as e:
        with open(output_file, "a") as file:

```

```

print(f"Error running command: {e}", file=file)

def run_kubectl_command(pod_name, server_pod_ip, package_size, output_file):
    command = f"kubectl exec -it {pod_name} -- iperf3 -c {server_pod_ip} -p 12345 -f k -n {package_size} -l {package_size}"
    run_command(command, output_file)

def run_ping_command(pod_name, server_pod_ip, package_size_ping, output_file):
    command = f"kubectl exec -it {pod_name} -- ping -c 1 {server_pod_ip} -s {package_size_ping}"
    run_command(command, output_file)

#def run_tests(pod_name, server_pod_ip, package_sizes, output_file, localhost):
def run_tests(pod_name, server_pod_ip, package_sizes, output_file,):
    for size in package_sizes:
        counter = 1
        for _ in range(10):
            if "ping" in size:
                run_ping_command(pod_name, server_pod_ip, size.split('_')[0], output_file)
            else:
                run_kubectl_command(pod_name, server_pod_ip, size, output_file)
            counter += 1
        print(f"Test completed with package size {size} Bytes")

if __name__ == "__main__":
    pod_name = input("ใส่ชื่อ pod ที่จะเข้าไปใช้คำสั่ง: ")
    server_pod_ip = input("ใส่ IP ของ Server Iperf: ")
    output_file = input("ใส่ชื่อ file: ")

    iperf3_package_sizes = ["100", "200", "400", "800", "1600", "3200", "6400", "12800", "25600",
                           "51200", "102400"]

```

```

ping_package_sizes = ["100_ping", "200_ping", "400_ping", "800_ping", "1600_ping",
"3200_ping", "6400_ping", "12800_ping", "25600_ping", "51200_ping"]

# ตรวจสอบว่ามีหลาย Pods หรือไม่
alls_name_pod = pod_name.split()
lens_pod = len(alls_name_pod)

# ตรวจสอบว่ามีหลายไฟล์ผลลัพธ์หรือไม่
all_output_file = output_file.split()
lens_output = len(all_output_file)

# ทดสอบและรันโค้ด ตามจำนวน Pods หรือใช้ Default
if lens_pod > 1 and lens_output > 1 and lens_output == lens_pod:
    for i in range(lens_pod):
        run_tests(alls_name_pod[i], server_pod_ip, iperf3_package_sizes +
ping_package_sizes, all_output_file[i])
        print("Sleep 10 S .......")
        time.sleep(10)
        print("Awake!!")
else:
    run_tests(pod_name, server_pod_ip, iperf3_package_sizes + ping_package_sizes,
output_file)

```

### 3.2.5.5 ใช้คำสั่งเริ่มต้นเซิร์ฟเวอร์ Iperf3

```
kubectl exec -it { pod-name } -- iperf3 -s -p 12345
```

```

user@control:~/CNI_Project$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE   NOMINATED NODE   READINESS GATES
iperf3-deployment-5b98b455df-z7hvt  1/1    Running   0          45s    10.1.210.69  workervm2  <none>      <none>
iperf3-deployment-5b98b455df-q8sj6  1/1    Running   0          45s    10.1.123.1   workervm1  <none>      <none>
iperf3-deployment-5b98b455df-nr5z1  1/1    Running   0          45s    10.1.36.5    workervm3  <none>      <none>
iperf3-deployment-5b98b455df-czz72  1/1    Running   0          45s    10.1.189.69  worker2     <none>      <none>
iperf3-deployment-5b98b455df-qzwng  1/1    Running   0          45s    10.1.119.197 workervm4  <none>      <none>
iperf3-deployment-5b98b455df-f4s12  1/1    Running   0          45s    10.1.235.129 worker1     <none>      <none>
iperf3-deployment-5b98b455df-f179p  1/1    Running   0          45s    10.1.182.5   worker3     <none>      <none>
user@control:~/CNI Project$ kubectl exec -it iperf3-deployment-5b98b455df-f4s12 -- iperf3 -s -p 12345
-----
Server listening on 12345 (test #1)
-----
```

รูปที่ 3.27 กำหนดเซิร์ฟเวอร์ Iperf3

### 3.2.5.6 ทำการรันโค้ด Python ที่ได้สร้างไว้

Python3 <File Name.py>

```
cni@cni:~/CNI_Project$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP
GATES
iperf3-deployment-5d949677c9-lrnwt  1/1     Running   0          36s    10.1.196.75
iperf3-deployment-5d949677c9-nskgj  1/1     Running   0          36s    10.1.196.76
iperf3-deployment-5d949677c9-zqmlj  1/1     Running   0          36s    10.1.196.78
iperf3-deployment-5d949677c9-vlswg  1/1     Running   0          36s    10.1.196.77
iperf3-deployment-5d949677c9-6nh8g  1/1     Running   0          37s    10.1.162.197
iperf3-deployment-5d949677c9-h9r5k  1/1     Running   0          36s    10.1.162.198
iperf3-deployment-5d949677c9-ptn5s  1/1     Running   0          36s    10.1.162.196
iperf3-deployment-5d949677c9-gk9nb  1/1     Running   0          36s    10.1.162.199
cni@cni:~/CNI_Project$ python3 testCNI1.py
ใส่ชื่อ pod ที่จะเข้าไปใช้งาน : iperf3-deployment-5d949677c9-gk9nb
ใส่ IP ของ Server Iperf: 10.1.196.75
ใส่ชื่อ file: TestDemo.txt
Test completed with package size 100 Bytes
Test completed with package size 200 Bytes
Test completed with package size 400 Bytes
Test completed with package size 800 Bytes
Test completed with package size 1600 Bytes
Test completed with package size 3200 Bytes
Test completed with package size 6400 Bytes
Test completed with package size 12800 Bytes
Test completed with package size 25600 Bytes
Test completed with package size 51200 Bytes
Test completed with package size 102400 Bytes
Test completed with package size 100_ping Bytes
Test completed with package size 200_ping Bytes
Test completed with package size 400_ping Bytes
Test completed with package size 800_ping Bytes
Test completed with package size 1600_ping Bytes
Test completed with package size 3200_ping Bytes
Test completed with package size 6400_ping Bytes
Test completed with package size 12800_ping Bytes
Test completed with package size 25600_ping Bytes
Test completed with package size 51200_ping Bytes
File uploaded successfullyUpload completed
cni@cni:~/CNI_Project$
```

รูปที่ 3.28 การใช้งาน Code Python

สำหรับการใช้งาน Code Python ดังกล่าว จำเป็นต้องกำหนดข้อมูลดังต่อไปนี้ให้ครบถ้วนสมบูรณ์

- ใส่ชื่อ Pod ที่จะเข้าไปใช้คำสั่ง: เลือก Pod ที่ทำหน้าที่เป็นตัวส่งข้อมูล
- ใส่ IP ของเซิร์ฟเวอร์ Iperf: เซิร์ฟเวอร์ IP ของ Iperf3 ที่กำหนดในขั้นตอนที่ 3.2.5.5
- ใส่ชื่อ File: กำหนดชื่อไฟล์ Text ที่ต้องการ (จำเป็นต้องใส่นามสกุลไฟล์)

### 3.2.6 วิธีการเปลี่ยน CNI เป็น Calico

3.2.6.1 ถ้าใช้คำสั่งในข้อที่ 3.2.3.1 นั้น CNI Calico จะถูกติดตั้งมาเป็นค่าเริ่มต้นอยู่แล้ว โดยสามารถเช็คว่า CNI Calico ทำงานแล้วได้โดยคำสั่งด้านล่างนี้

```
kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-864597b5fd-pq4dj	1/1	Running	1 (9m30s ago)	28m
kube-system	calico-node-4bfpc	1/1	Running	0	3m16s
kube-system	calico-kube-controllers-77bd7c5b-75bbm	1/1	Running	0	3m16s

รูปที่ 3.29 วิธีการตรวจสอบการใช้งาน CNI Calico

3.2.6.2 เช็คว่ามีอินเตอร์เฟซของ CNI Calico ถูกสร้างขึ้นมาหรือไม่ โดยใช้คำสั่งดังนี้

```
ip a
```

```
91: vxlan.calico: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UNKNOWN group default
    link/ether 66:51:20:96:ad:44 brd ff:ff:ff:ff:ff:ff
      inet 10.1.162.192/32 scope global vxlan.calico
        valid_lft forever preferred_lft forever
      inette fe80::6451:20ff:fe96:ad44/64 scope link
        valid_lft forever preferred_lft forever
```

รูปที่ 3.30 อินเตอร์เฟซของ CNI Calico

3.2.6.3 สามารถหยุดใช้งาน CNI Calico ได้โดยการดึงหนนดทุกตัวออกจากกลุ่ม หลังจากนั้นให้ใช้คำสั่งนี้กับทุกๆ โนนด

```
microk8s kubectl delete -f /var/snap/microk8s/current/args/cni-network/cni.yaml
```

```
cni@cni:~/CNI_Project$ microk8s kubectl delete -f /var/snap/microk8s/current/args/cni-network/cni.yaml
poddisruptionbudget.policy "calico-kube-controllers" deleted
serviceaccount "calico-kube-controllers" deleted
serviceaccount "calico-node" deleted
configmap "calico-config" deleted
customresourcedefinition.apiextensions.k8s.io "bgpconfigurations.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "bgppeers.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "blockaffinities.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "caliconodestatuses.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "clusterinformations.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "felixconfigurations.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "globalnetworkpolicies.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "globalnetworksets.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "hostendpoints.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "ipamblocks.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "ipamconfigs.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "ipamhandles.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "ippools.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "ipreservations.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "kubecontrollersconfigurations.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "networkpolicies.crd.projectcalico.org" deleted
customresourcedefinition.apiextensions.k8s.io "networksets.crd.projectcalico.org" deleted
clusterrole.rbac.authorization.k8s.io "calico-kube-controllers" deleted
clusterrole.rbac.authorization.k8s.io "calico-node" deleted
clusterrolebinding.rbac.authorization.k8s.io "calico-kube-controllers" deleted
clusterrolebinding.rbac.authorization.k8s.io "calico-node" deleted
daemonset.apps "calico-node" deleted
deployment.apps "calico-kube-controllers" deleted
```

รูปที่ 3.31 วิธีการหยุดใช้งาน CNI Calico

### 3.2.7 วิธีการเปลี่ยน CNI เป็น Cilium

#### 3.2.7.1 ทำการดึงโหนดทุกตัวออกจากกลุ่ม

#### 3.2.7.2 ใช้คำสั่งด้านล่างนี้ในโหนดแต่ละตัว ให้ครับทุกตัวที่จะทำการเพิ่มเข้ากลุ่ม

microk8s enable community

microk8s enable cilium

```
customresourcedefinition.apirextensions.k8s.io "globalnetworksets.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "hostendpoints.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "ipamblocks.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "ipamconfigs.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "ipamhandles.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "ippools.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "ipreservations.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "networkcontrollersconfigurations.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "networkpolicies.crd.projectcalico.org" deleted
customresourcedefinition.apirextensions.k8s.io "networksets.crd.projectcalico.org" deleted
clusterrole.rbac.authorization.k8s.io "calico-kube-controllers" deleted
clusterrole.rbac.authorization.k8s.io "calico-node" deleted
clusterrolebinding.rbac.authorization.k8s.io "calico-kube-controllers" deleted
clusterrolebinding.rbac.authorization.k8s.io "calico-node" deleted
daemonset.apps "calico-node" deleted
deployment.apps "calico-kube-controllers" deleted
Enabling Cilium
Fetching cilium version v1.13.4.
  % Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
               Dload  Upload   Total Spent   Left Speed
  0     0    0     0    0     0      0 --:--:-- --:--:-- --:--:--    0
100 36.6M  0 36.6M  0     0  7217k      0 --:--:-- 0:00:05 --:--:-- 8098k
"cilium" already exists with the same configuration, skipping
NAME: cilium
LAST DEPLOYED: Fri Mar  1 21:58:45 2024
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
You have successfully installed Cilium with Hubble.

Your release version is 1.13.4.

For any further help, visit https://docs.cilium.io/en/v1.13/gettinghelp
Waiting for cilium. This may take several minutes.
Waiting for daemon set "cilium" rollout to finish: 0 of 1 updated pods are available...
Waiting for daemon set "cilium" rollout to finish: 0 of 1 updated pods are available...
daemon set "cilium" successfully rolled out
Installing Cilium CLI binary
  % Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
               Dload  Upload   Total Spent   Left Speed
  0     0    0     0    0     0      0 --:--:-- 0:00:04 --:--:--    0
100 30.6M  100 30.6M  0     0  3922k      0  0:00:08  0:00:08 --:--:-- 13.5M
Cilium is enabled

You can now enable hubble with:
  sudo microk8s cilium hubble enable
```

รูปที่ 3.32 การติดตั้ง CNI Cilium

### 3.2.7.3 ทำการเช็คว่า CNI Cilium ได้ทำงานแล้วโดยใช้คำสั่งด้านล่าง

```
kubectl get pods -A
```

```
cni@cni:~/CNI_Project$ kubectl get pods -A
NAMESPACE      NAME            READY   STATUS    RESTARTS   AGE
kube-system    cilium-operator-f5b4997d6-vn84l   1/1     Running   1 (2m1s ago)   2m13s
kube-system    coredns-864597b5fd-pq4dj        1/1     Running   1 (2m1s ago)   21m
kube-system    cilium-fxbnb                   1/1     Running   1 (2m1s ago)   2m13s
```

รูปที่ 3.33 วิธีการตรวจสอบการใช้งาน CNI Cilium

### 3.2.7.4 เช็คว่ามีอินเตอร์เฟชของ CNI Cilium ถูกสร้างขึ้นมาหรือไม่ โดยใช้คำสั่งดังนี้

```
ip a
```

```
73: cilium_net@cilium_host: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
  link/ether ce:6c:b4:4d:94:73 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::cc6c:b4ff:fed4:9473/64 scope link
      valid_lft forever preferred_lft forever
74: cilium_host@cilium_net: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
  link/ether 72:d9:24:8f:4a:9b brd ff:ff:ff:ff:ff:ff
    inet 10.1.0.17/32 scope global cilium_host
      valid_lft forever preferred_lft forever
    inet6 fe80::70d9:24ff:fea8:4a9b/64 scope link
      valid_lft forever preferred_lft forever
94: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default qlen 1000
  link/ether fa:1d:c2:a9:06:84 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::f81d:c2ff:fea9:684/64 scope link
      valid_lft forever preferred_lft forever
```

รูปที่ 3.34 อินเตอร์เฟชของ CNI Cilium

### 3.2.7.5 ทำการเพิ่มโหนดเข้ากลุ่มใหม่อีกเดียว

3.2.7.6 สามารถหยุดใช้งาน CNI Cilium ได้โดยการดึงโหนดทุกตัวออกจากกลุ่มหลังจากนั้นให้ใช้คำสั่งนี้กับทุกๆ โหนด

microk8s disable cilium

```
cni@cni:~/CNI_Project$ microk8s disable cilium
Infer repository community for addon cilium
Disabling Cilium
release "cilium" uninstalled
75: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 7a:2b:76:c3:f4:88 brd ff:ff:ff:ff:ff:ff
Deleting old cilium_vxlan link
Restarting default cni
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apirextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/irpresaervations.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apirextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
Cilium is terminating
```

รูปที่ 3.35 วิธีการหยุดใช้งาน CNI Cilium

### 3.2.8 วิธีการเปลี่ยน CNI เป็น Flannel

3.2.8.1 ทำการดึงโหนดทุกตัวออกจากกลุ่ม

3.2.8.2 ยกเลิกการใช้งาน CNI ที่มีอยู่ในเครื่องทั้งหมด

3.2.8.3 ใช้คำสั่งด้านล่างนี้ในโหนดแต่ละตัวให้ครบทุกตัวที่จะทำการเพิ่มเข้ากลุ่ม

microk8s disable ha-cluster --force

```
cni@cni:~/CNI_Project$ microk8s disable ha-cluster --force
Infer repository core for addon ha-cluster
Reverting to a non-HA setup
Enabling flanneld and etcd
HA disabled
```

รูปที่ 3.36 วิธีการติดตั้ง CNI Flannel

3.2.8.4 เราไม่สามารถใช้คำสั่ง kubectl get pods -A เพื่อเช็คว่า CNI Flannel ทำงานแล้ว หรือยังเหมือนกับ CNI 2 ตัวก่อนหน้า แต่เราสามารถใช้คำสั่งด้านล่างนี้ เพื่อเช็คว่ามี อินเตอร์เฟซ ของ CNI Flannel ถูกสร้างขึ้นมาหรือไม่

```
ip a
```

```
93: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 02:99:0c:97:dc:3b brd ff:ff:ff:ff:ff:ff
        inet 10.1.21.0/32 scope global flannel.1
            valid_lft forever preferred_lft forever
        inet6 fe80::99:cff:fe97:dc3b/64 scope link
            valid_lft forever preferred_lft forever
```

รูปที่ 3.37 อินเตอร์เฟซของ CNI Flannel

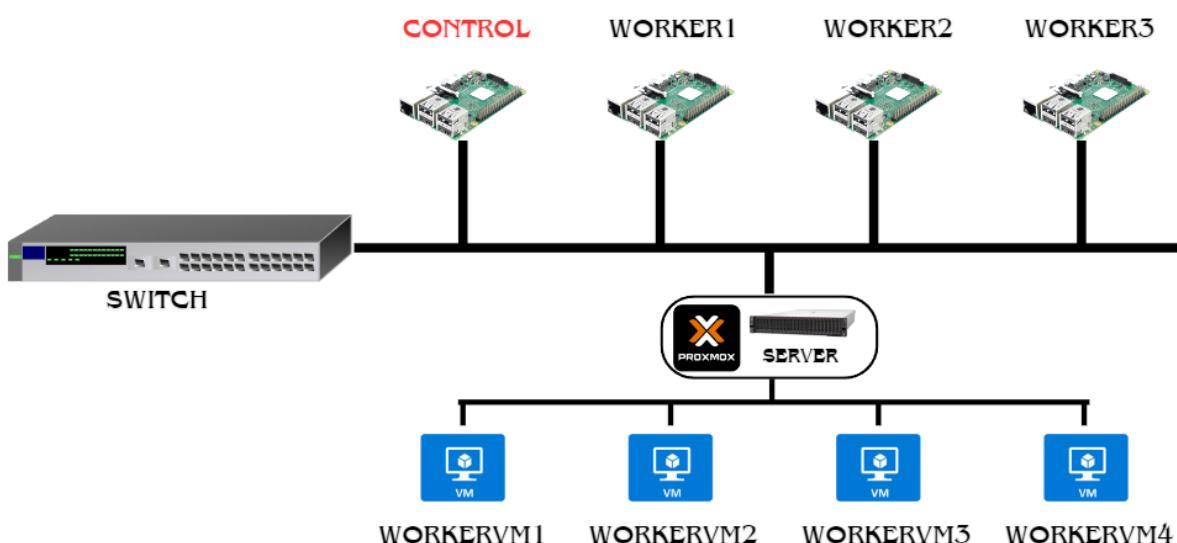
3.2.8.5 สามารถหยุดใช้งาน CNI Flannel ได้โดยการตึงโหนดทุกด้วยกันจากกลุ่มหลังจากนั้น ให้ใช้คำสั่งนี้กับทุกๆ โหนด

```
microk8s enable ha-cluster
```

```
cni@cni:~/CNI_Project$ microk8s enable ha-cluster
Infer repository core for addon ha-cluster
Enabling HA
Upgrading the network CNI
Waiting for the cluster to restart
```

รูปที่ 3.38 วิธีการหยุดใช้งาน CNI Flannel

3.2.9 สภาพแวดล้อมโดยรวมทั้งหมดที่ใช้ในการทดสอบ



รูปที่ 3.39 สภาพแวดล้อมโดยรวมทั้งหมดที่ใช้ในการทดสอบ

ในสภาพแวดล้อมการทดสอบที่ใช้นั้น ตัวของ Raspberry Pi 4 และ เครื่องจำลองเสมือน (VM) ที่อยู่บน Lenovo ThinkSystem SR530 Server จะได้รับการติดตั้ง Microk8s เวอร์ชัน 1.28 เมมีอนกันหมวดในทุกๆ โหนดในสภาพแวดล้อมการทดสอบ

ในการทดสอบบันทึกผลเก็บค่า Throughput (Kbits/sec) และ Latency (ms) นั้นในแต่ละขนาดแพ็คเก็ตที่ได้กำหนดไว้ 11 ขนาด คือ 100 ไบต์, 200 ไบต์, 400 ไบต์, 800 ไบต์, 1,600 ไบต์, 3,200 ไบต์, 6,400 ไบต์, 12,800 ไบต์, 25,600 ไบต์, 51,200 ไบต์ และ 102,400 ไบต์ ต้องทดสอบจำนวน 50 รอบในแต่ละขนาดแพ็คเก็ตเพื่อนำค่าที่ได้มาคำนวณและเป็นค่าเฉลี่ยที่เกิดขึ้น

ฮาร์ดแวร์ที่ใช้ทดสอบในสภาพแวดล้อมต่างกัน 2 ชนิด คือ Raspberry Pi 4 และ เครื่องจำลองเสมือน (VM) ที่อยู่บน Lenovo ThinkSystem SR530 Server ดังนั้นการทดสอบจำเป็นต้องมีการจับคู่และสลับหน้าที่ในการทดสอบเพื่อเปรียบเทียบผลลัพธ์ที่เกิดขึ้น

ค่าปริมาณงาน หรือ Throughput (Kbits/sec) และ เวลาแฟรงหรือ Latency (ms) ที่ได้จะนำมาพล็อตเป็นกราฟเส้นเพื่อดูและเปรียบเทียบผลลัพธ์ที่ได้ในแต่ละ CNI และในแต่ละการจับคู่ของฮาร์ดแวร์ที่แตกต่างกัน

## บทที่ 4

### ผลการดำเนินโครงการ

การศึกษาและทดสอบการทำงานของ Container Network Interface (CNI) ทั้ง 3 ตัว คือ Calico, Flannel และ Cilium ซึ่งได้ผลลัพธ์มาดังนี้

#### 4.1 ตารางผลการทดสอบ

กำหนดให้  $(S)$  = ตัวส่งข้อมูล

$(R)$  = ตัวรับข้อมูล

##### 4.1.1 Calico

###### 4.1.1.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

ตารางที่ 4.1 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R) ของ CNI: Calico

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	3932.84	0.8724	
200	8009.34	0.8729	
400	15975.64	0.8731	
800	29025.48	0.8956	
1600	53046.42	1.1832	
3200	98621.1	1.335	
6400	170910.9	1.5639	
12800	244658.16	2.1822	
25600	443753.2	3.3599	
51200	768883.88	5.786	
102400	1391122.12	-	

#### 4.1.1.2 Raspberry Pi 4 (S) กับ VM (R)

ตารางที่ 4.2 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ VM (R) ของ CNI: Calico

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	4738.88	0.7779	
200	9962.04	0.7774	
400	19343.74	0.8109	
800	37524.78	0.8106	
1600	66986.92	0.9719	
3200	117072	1.0773	
6400	200957.02	1.2014	
12800	277410.3	1.4069	
25600	545092.9	1.949	
51200	870079.88	3.1047	
102400	1511059.98	-	

#### 4.1.1.3 VM (S) กับ Raspberry Pi 4 (R)

ตารางที่ 4.3 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ Raspberry Pi 4 (R) ของ CNI: Calico

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	4993.58	0.9139	
200	8919.62	0.9224	
400	17556.24	0.9152	
800	31235.68	0.9319	
1600	63048.92	1.1551	
3200	118388.8	1.2573	
6400	236974.56	1.5121	
12800	325037.62	2.1007	
25600	739893.46	3.1306	
51200	1485473.34	5.2209	
102400	3330186.42	-	

#### 4.1.1.4 VM (S) กับ VM (R)

ตารางที่ 4.4 ผลการทดสอบเบลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ VM (R) ของ CNI: Calico

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	6915.1	0.7109	
200	15792.18	0.7043	
400	29578.4	0.6938	
800	50047.02	0.7205	
1600	104500.18	0.7783	
3200	186787.86	0.8302	
6400	327404.7	0.8742	
12800	488900.46	0.9668	
25600	949452.3	1.2473	
51200	1757853.16	1.8843	
102400	3536451.6	-	

จากตารางผลการทดสอบของ Calico ในทุกการจับคู่ชาร์ดแวร์ จะเห็นได้ว่าค่าเมื่อเพิ่มขนาดแพ็คเก็ตมากขึ้นค่า Throughput ก็จะมากขึ้นไปด้วย แต่ Latency ในขนาดแพ็คเก็ตที่เล็กในหลักร้อยมีค่าที่ไม่แตกต่างกันมากเท่าไหร่

#### 4.1.2 Flannel

##### 4.1.2.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

ตารางที่ 4.5 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R) ของ CNI: Flannel

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
	100	3712.74	0.9191
	200	6829.98	0.9299
	400	13796.84	0.9185
	800	27881.2	0.9452
	1600	54626.58	1.2554
	3200	104402.78	1.4073
	6400	188960.94	1.6606
	12800	286269.72	2.1976
	25600	531054.76	3.6218
	51200	939737.16	6.2884
	102400	1671924.68	-

##### 4.1.2.2 Raspberry Pi 4 (S) กับ VM (R)

ตารางที่ 4.6 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ VM (R) ของ CNI: Flannel

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
	100	4676.06	0.8816
	200	9534.5	0.8764
	400	18327.24	0.8939
	800	37033.08	0.8995
	1600	67409.7	1.0692
	3200	125895.2	1.1407
	6400	240059.88	1.2699
	12800	358994.04	1.5007
	25600	660766.18	2.0502
	51200	1026713.54	3.2092
	102400	1842409.92	-

#### 4.1.2.3 VM (S) กับ Raspberry Pi 4 (R)

ตารางที่ 4.7 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ Raspberry Pi 4 (R) ของ CNI: Flannel

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	3829.12	0.9539	
200	8727.16	0.9725	
400	14692.84	0.99	
800	31677	0.9953	
1600	61551.68	1.2065	
3200	111025.54	1.3707	
6400	242744.92	1.618	
12800	410495.36	2.1391	
25600	949163.48	3.4313	
51200	1708908.2	5.8127	
102400	3414643.64	-	

#### 4.1.2.4 VM (S) กับ VM (R)

ตารางที่ 4.8 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ VM (R) ของ CNI: Flannel

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	5970.32	0.7622	
200	13233.284	0.7664	
400	29651.04	0.7582	
800	48005.1	0.7657	
1600	89476.1	0.8619	
3200	202787.1	0.9257	
6400	387658.4	0.9694	
12800	563735.12	1.1069	
25600	1041341.26	1.4114	
51200	1975507.92	2.159	
102400	3444423.66	-	

จากการทดสอบของ Cilium ในทุกการจับคู่ยาร์ดแวร์ จะเห็นได้ว่าค่าเมื่อเพิ่มขนาดแพ็คเก็ตมากขึ้นค่า Throughput ก็จะมากขึ้นไปด้วย แต่ Latency ในขนาดแพ็คเก็ตที่เล็กในหลักร้อยมีค่าที่ไม่แตกต่างกันมากเท่าไหร่ซึ่งผลลัพธ์มีแนวโน้มคล้ายคลึงกับ Calico

#### 4.1.3 Cilium

##### 4.1.3.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

**ตารางที่ 4.9** ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R) ของ CNI: Cilium

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	3730.72	1.0088	
200	7033.3	0.9793	
400	16001.36	0.9643	
800	30714.82	0.9829	
1600	56680.16	1.2734	
3200	107485.62	1.3912	
6400	200997.48	1.6183	
12800	322294.12	2.2099	
25600	615128.64	3.4925	
51200	209375	6.3866	
102400	277064.22	-	

#### 4.1.3.2 Raspberry Pi 4 (S) กับ VM (R)

ตารางที่ 4.10 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง Raspberry Pi 4 (S) กับ VM (R) ของ CNI: Cilium

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	4608.16	0.9302	
200	9637.9	0.9378	
400	19642.18	0.9252	
800	38519.52	0.9338	
1600	69988.76	1.0657	
3200	130612.28	1.1415	
6400	249385.92	1.2446	
12800	379788.46	1.5452	
25600	694562.64	2.048	
51200	438870.74	3.1928	
102400	590468.22	-	

#### 4.1.3.3 VM (S) กับ Raspberry Pi 4 (R)

ตารางที่ 4.11 ผลการทดสอบเฉลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ Raspberry Pi 4 (R) ของ CNI: Cilium

Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	3982.6	1.0283	
200	9194.96	0.9883	
400	18510.08	1.0077	
800	31932.64	1.0396	
1600	67800.46	1.2042	
3200	129794.7	1.3696	
6400	256276.22	1.6422	
12800	487348.4	2.2158	
25600	1105208.6	3.5055	
51200	373133.54	5.7559	
102400	627650.78	-	

#### 4.1.3.4 VM (S) กับ VM (R)

ตารางที่ 4.12 ผลการทดสอบแลี่ยของแต่ละขนาดแพ็คเก็ตระหว่าง VM (S) กับ VM (R) ของ CNI: Cilium

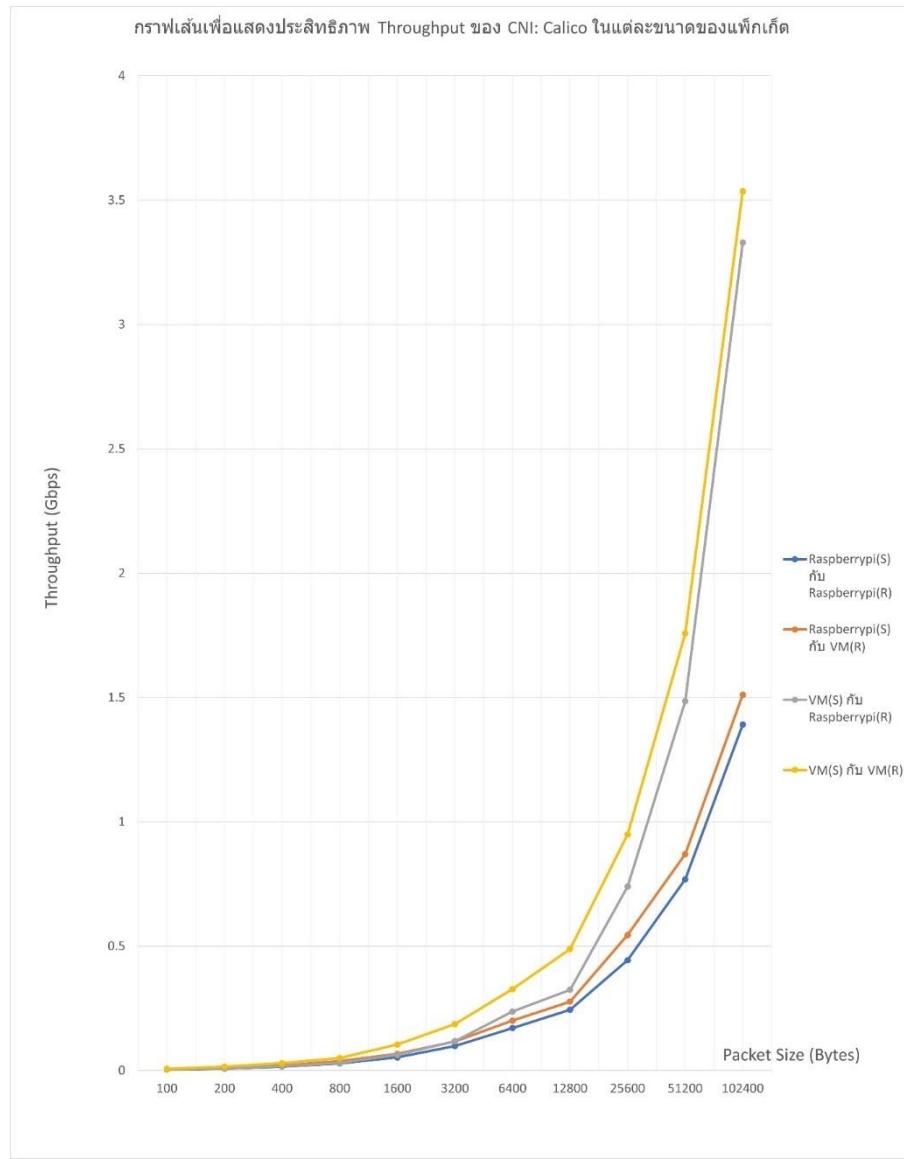
Avg	Packet Size (Bytes)	Throughput (Kbits/sec)	Latency(ms)
100	4470.26	0.8835	
200	10523.264	0.8475	
400	23656.46	0.8524	
800	46040.42	0.8528	
1600	85244.76	0.9539	
3200	148469.9	0.9885	
6400	302952.56	1.0904	
12800	477647.88	1.2045	
25600	974040.1	1.5881	
51200	570752.38	2.351	
102400	975524.06	-	

จากการทดสอบของ Cilium ในทุกการจับคู่ชาร์ดแวร์ จะเห็นได้ว่าค่าเมื่อเพิ่มขนาดแพ็คเก็ตมากขึ้นค่า Throughput ก็จะมากขึ้นไปด้วยจนถึงขนาดแพ็คเก็ต 25,600 ไบต์ เมื่อสั้นเกตแพ็คเก็ต 51,200 ไบต์ มีค่า Throughput ที่ต่ำลงอย่างมากและก็เพิ่มขึ้นเมื่อแพ็คเก็ต 102,400 ไบต์ และ Latency ยังมีผลลัพธ์คล้ายคลึงและแนวโน้มเดียวกันกับ Calico และ Cilium

## 4.2 กราฟแสดงประสิทธิภาพของ CNI ในแต่ละขนาดของแพ็คเก็ต

### 4.2.1 Calico

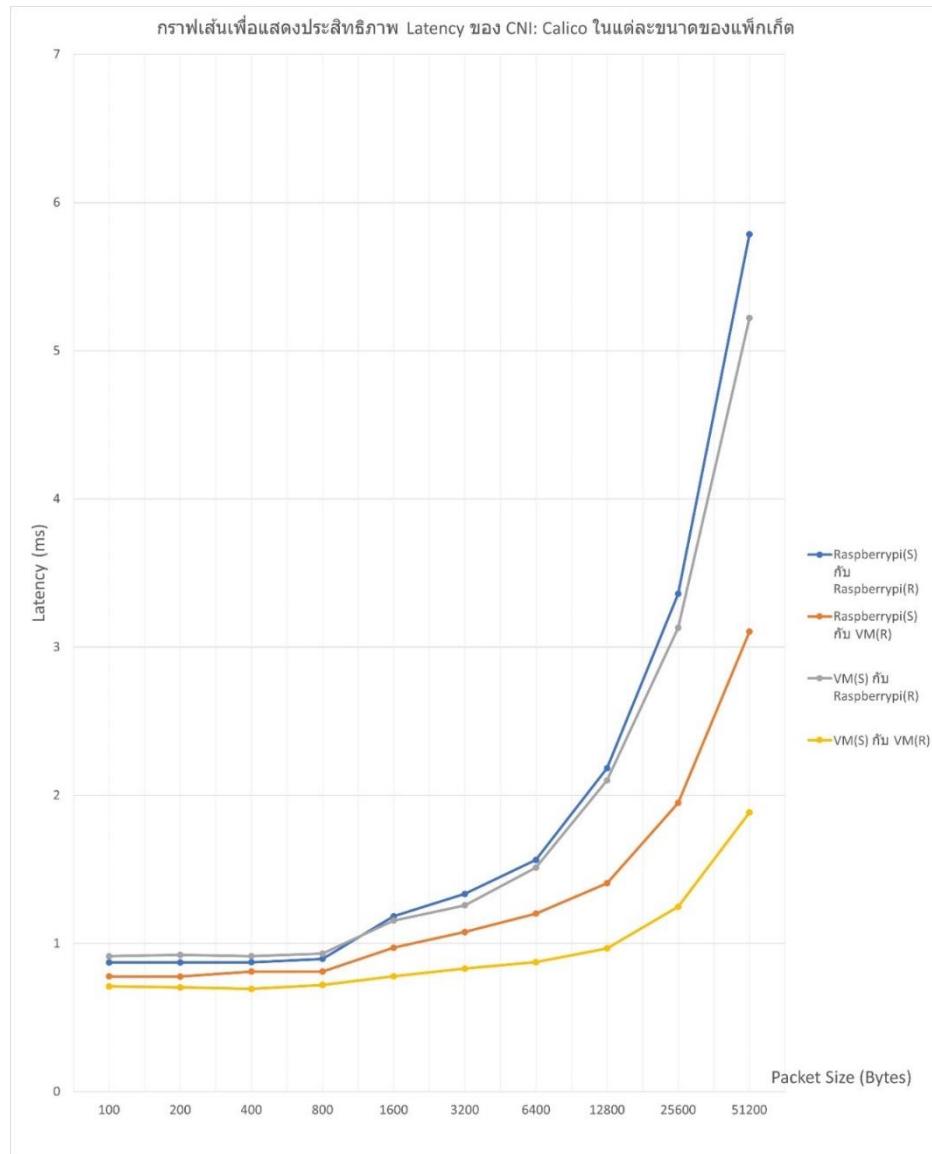
#### 4.2.1.1 Throughput



รูปที่ 4.1 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Calico ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้ส่งข้อมูลเป็น VM จะมีค่า Throughput ที่สูงกว่า แต่ถ้าหากเป็น Raspberry Pi 4

#### 4.2.1.2 Latency

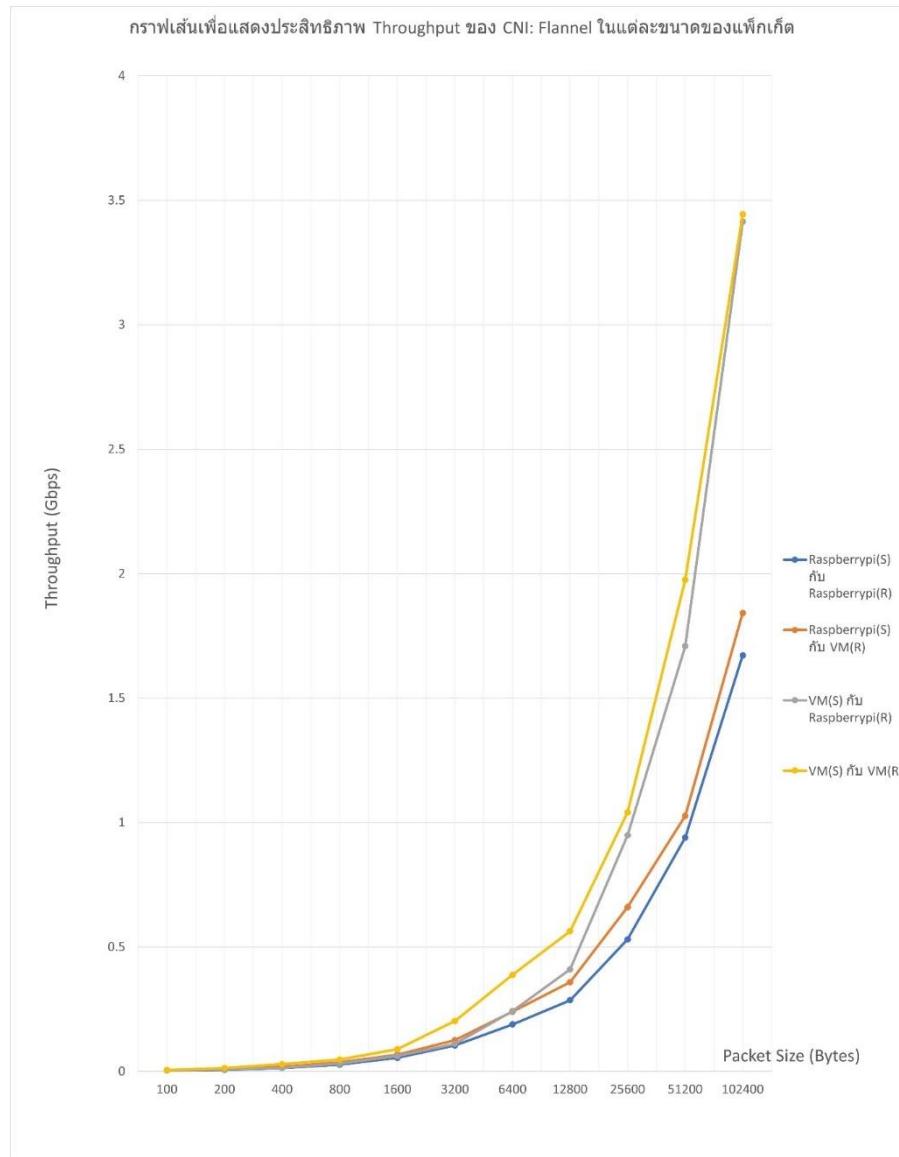


รูปที่ 4.2 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Calico ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4

## 4.2.2 Flannel

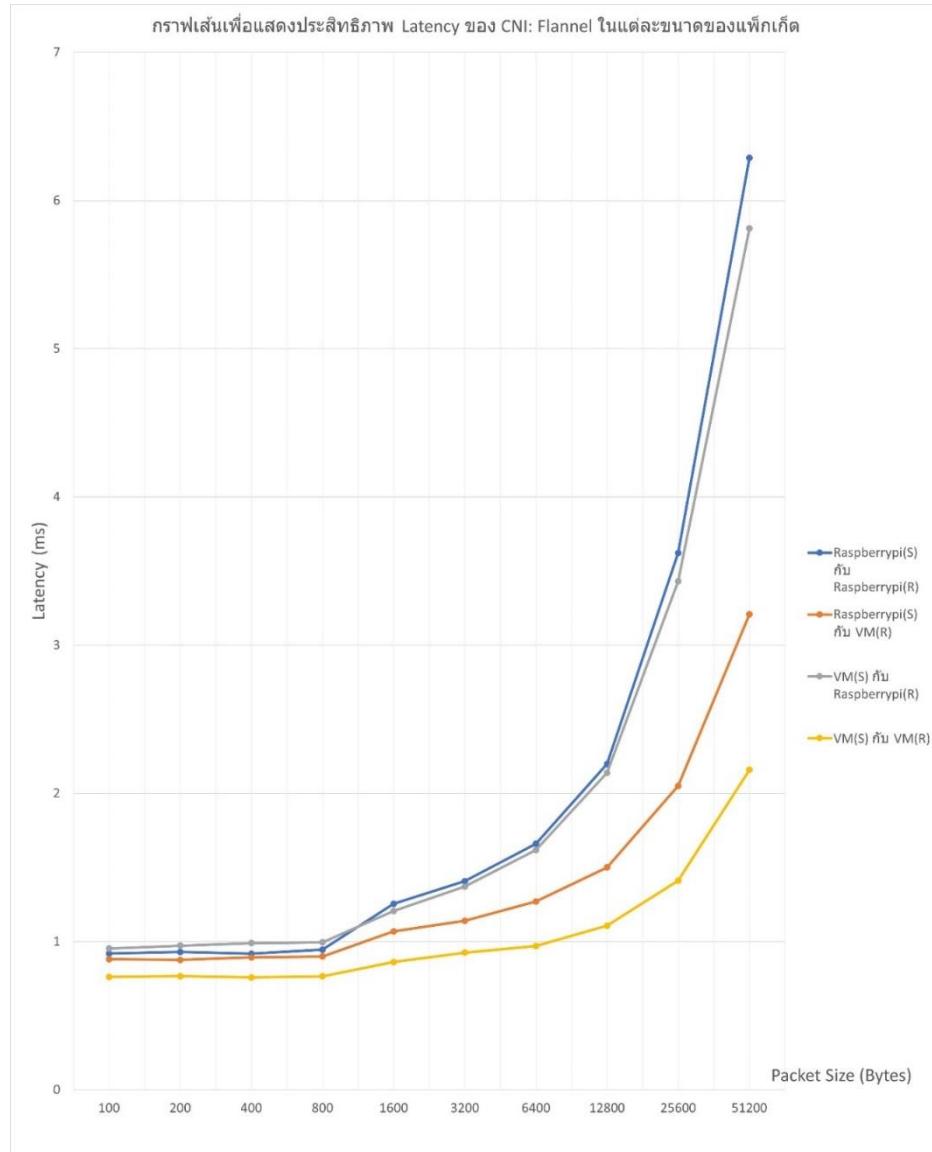
### 4.2.2.1 Throughput



รูปที่ 4.3 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Flannel ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้ส่งข้อมูลเป็น VM จะมีค่า Throughput ที่สูงกว่า แต่ถ้าหากเป็น Raspberry Pi 4

#### 4.2.2.2 Latency

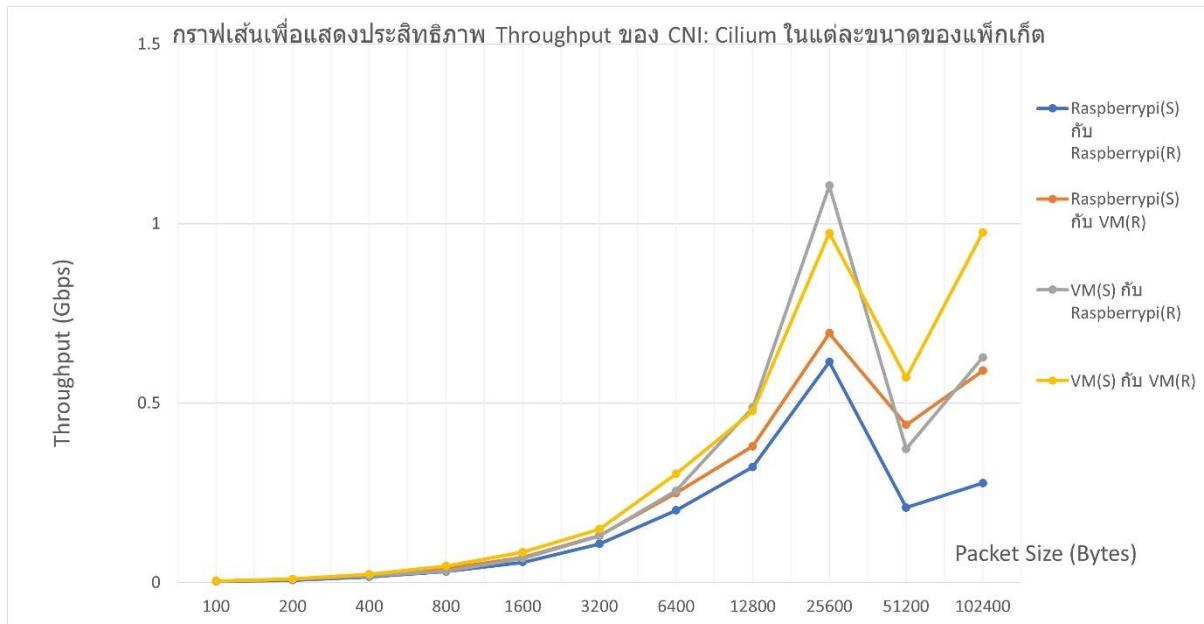


รูปที่ 4.4 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Flannel ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4

### 4.2.3 Cilium

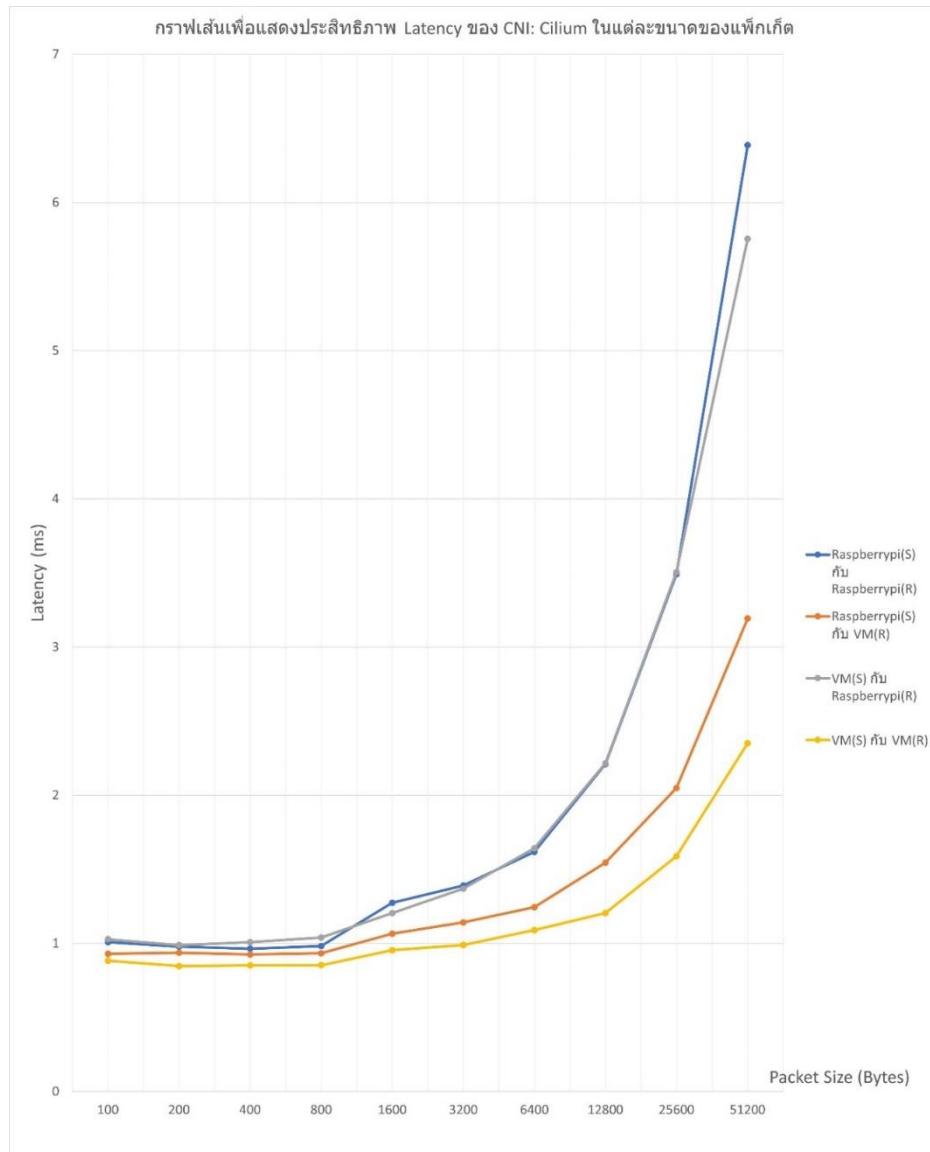
#### 4.2.3.1 Throughput



รูปที่ 4.5 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI: Cilium ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาจนถึงขนาดแพ็คเก็ตที่ 25,600 ไปต่อ จากนั้นค่าที่ได้จากขนาดแพ็คเก็ตที่เหลือไม่สามารถคาดเดาหรือสรุป

#### 4.2.3.2 Latency



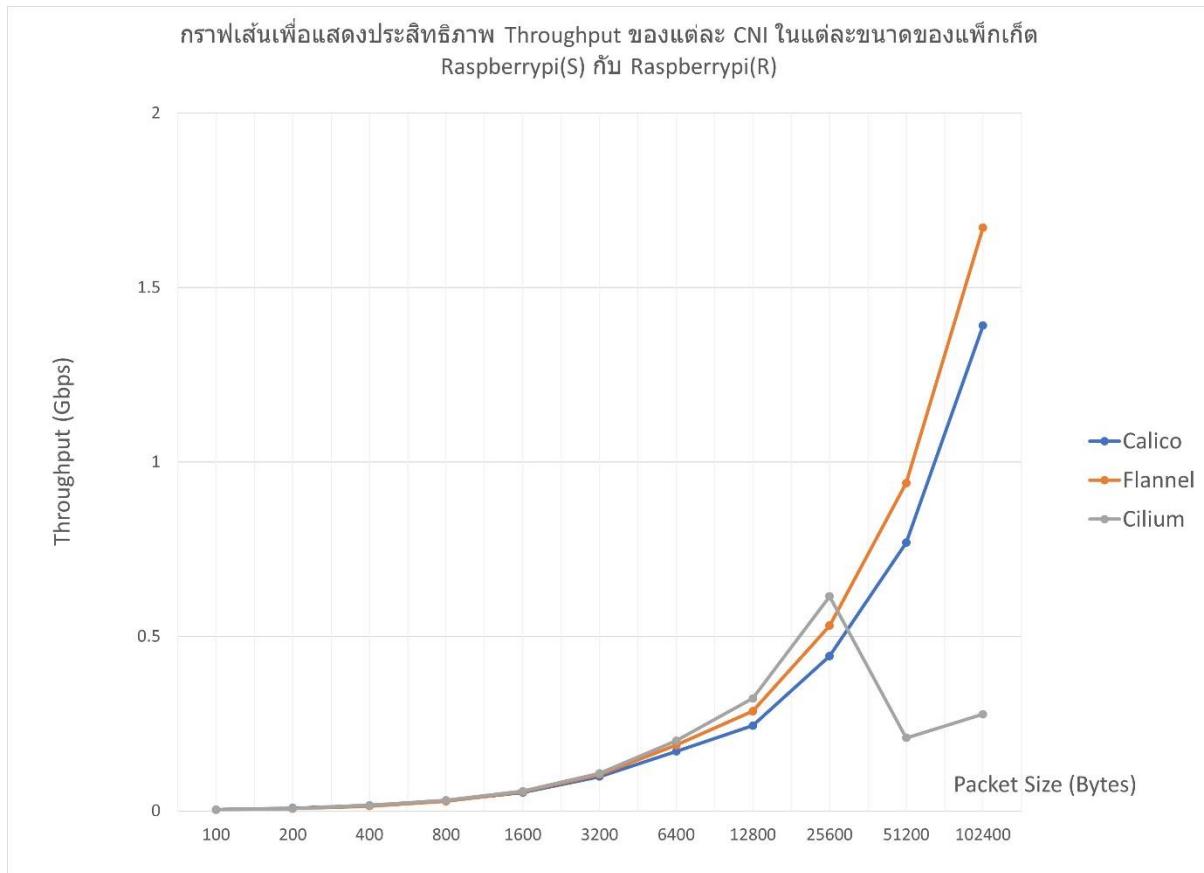
รูปที่ 4.6 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI: Cilium ในแต่ละขนาดของแพ็คเก็ต

กราฟเส้น Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4

4.3 กราฟแสดงการเปรียบเทียบประสิทธิภาพของ CNI ทั้ง 3 ตัวในด้านต่างๆ ตามชนิดของโหนดที่จับคู่กัน

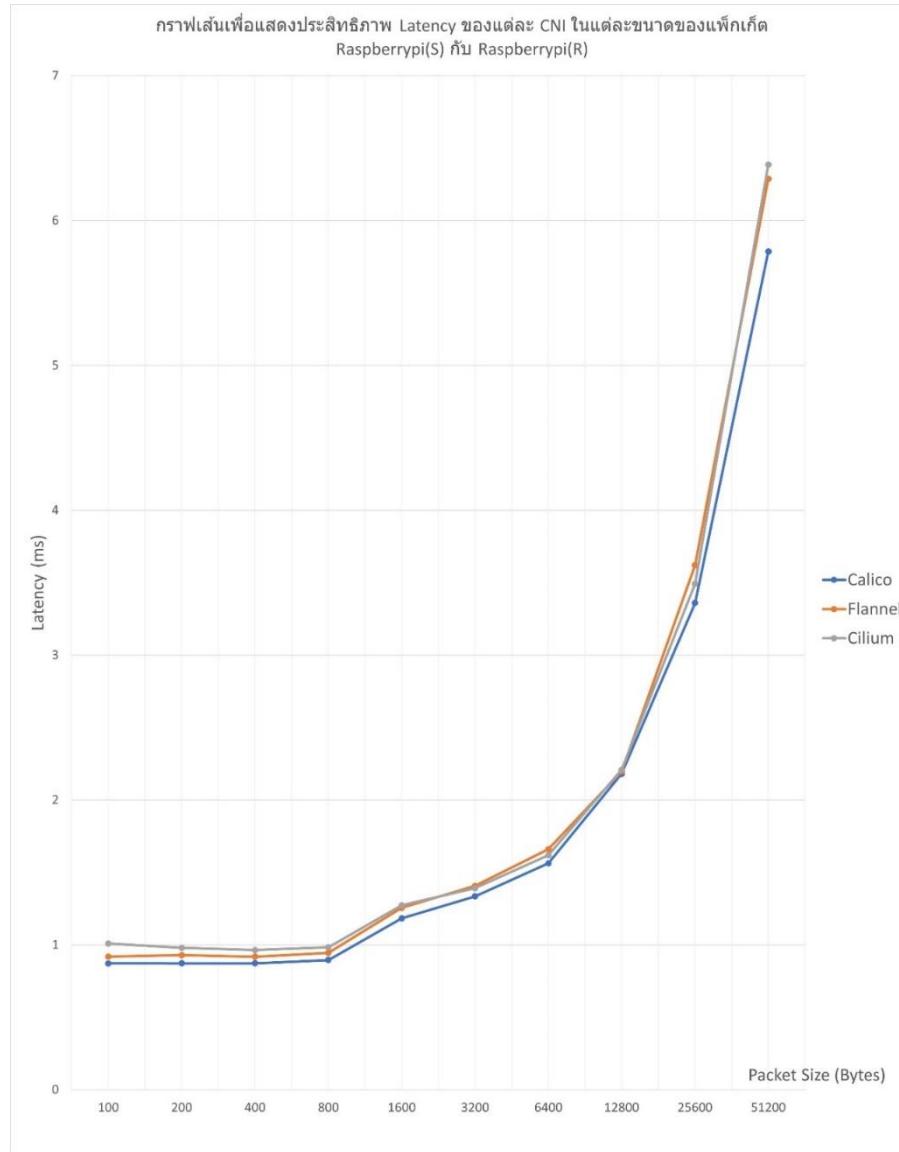
#### 4.3.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

##### 4.3.1.1 Throughput



รูปที่ 4.7 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

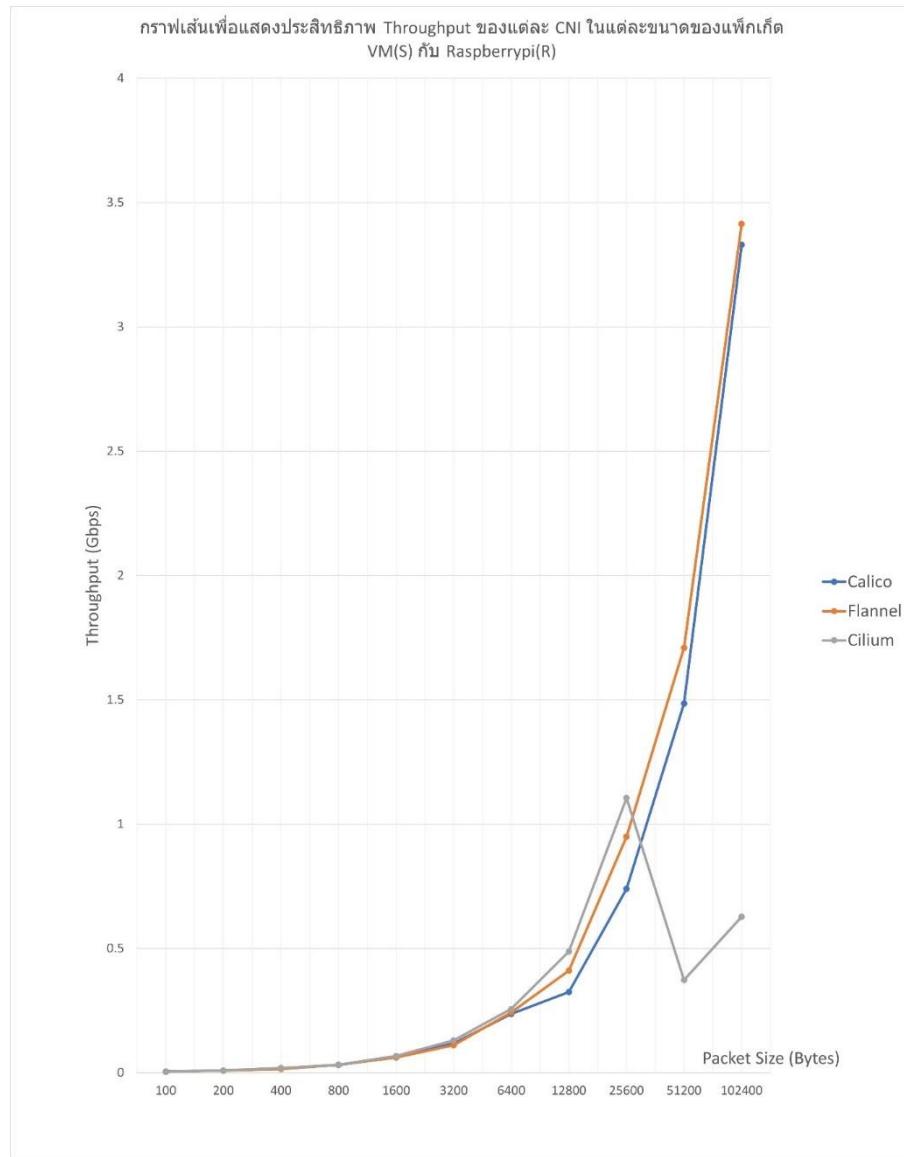
#### 4.3.1.2 Latency



รูปที่ 4.8 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R)

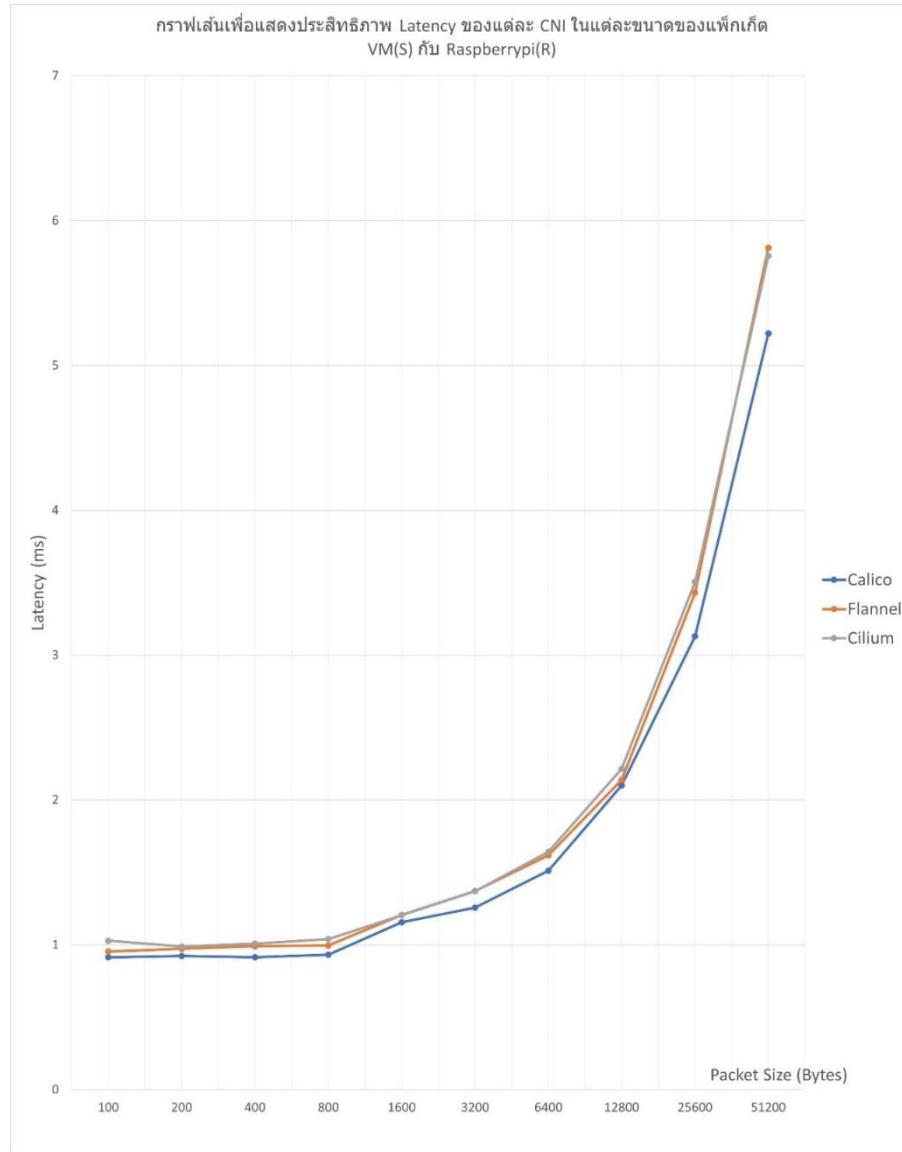
### 4.3.2 VM (S) กับ Raspberry Pi 4 (R)

#### 4.3.2.1 Throughput



รูปที่ 4.9 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ Raspberry Pi 4 (R)

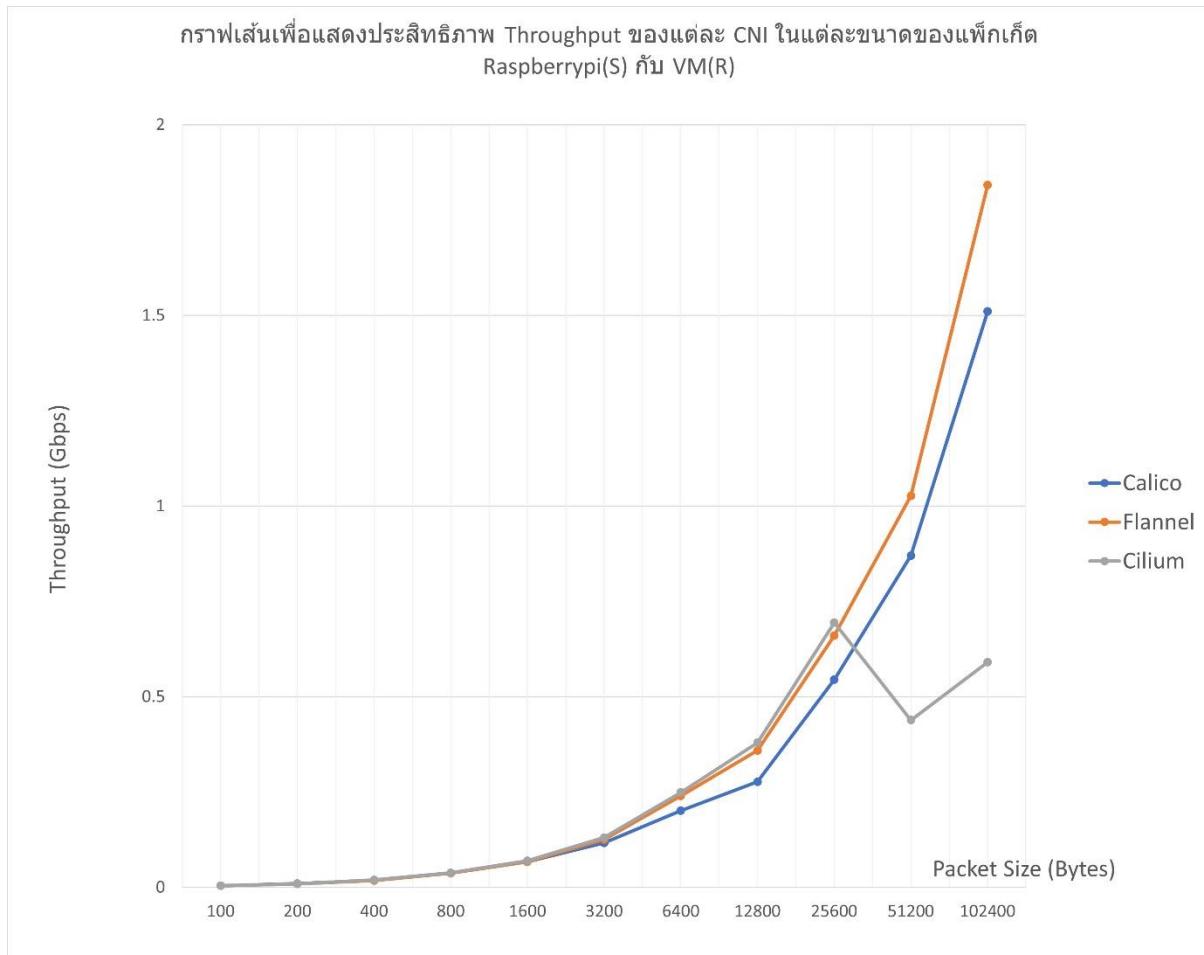
#### 4.3.2.2 Latency



รูปที่ 4.10 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ Raspberry Pi 4 (R)

### 4.3.3 Raspberry Pi 4 (S) กับ VM (R)

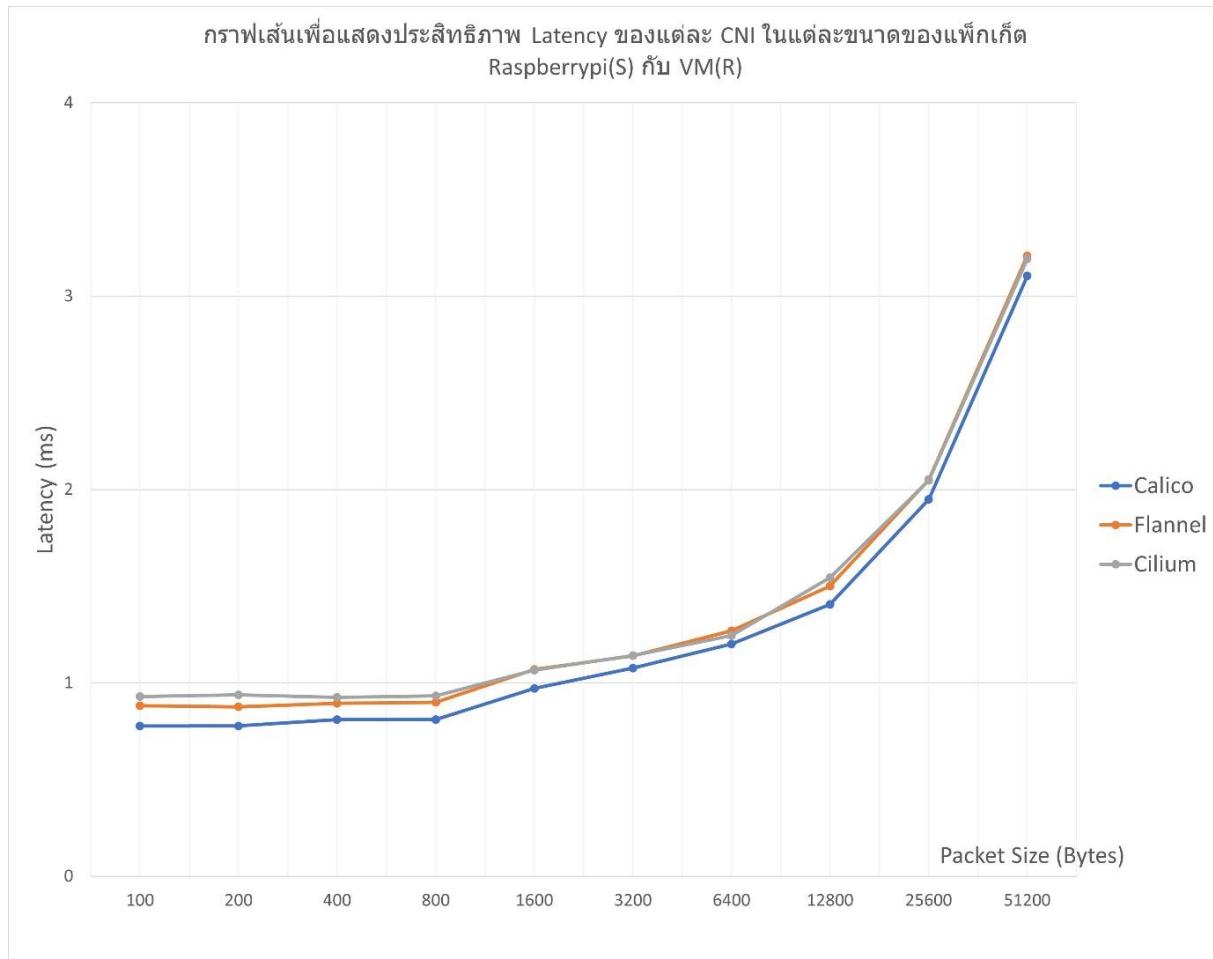
#### 4.3.3.1 Throughput



รูปที่ 4.11 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ VM (R)

Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R), VM (S) กับ Raspberry Pi 4 (R) และ Raspberry Pi 4 (S) กับ VM (R) มีกราฟเส้น Throughput ของ Calico และ Flannel มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา แต่ Cilium มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาถึงขนาดแพ็คเก็ตที่ 25,600 ไบต์ หลังจากนั้นมีค่าที่ลดลงอย่างมากไม่สามารถคาดเดาได้ จะได้เห็นได้ Flannel มีค่า Throughput มากที่สุด เมื่อมีขนาดแพ็คเก็ตที่มากกว่า 25,600 ไบต์ รองลงมาเป็น Calico แต่ Cilium

#### 4.3.3.2 Latency

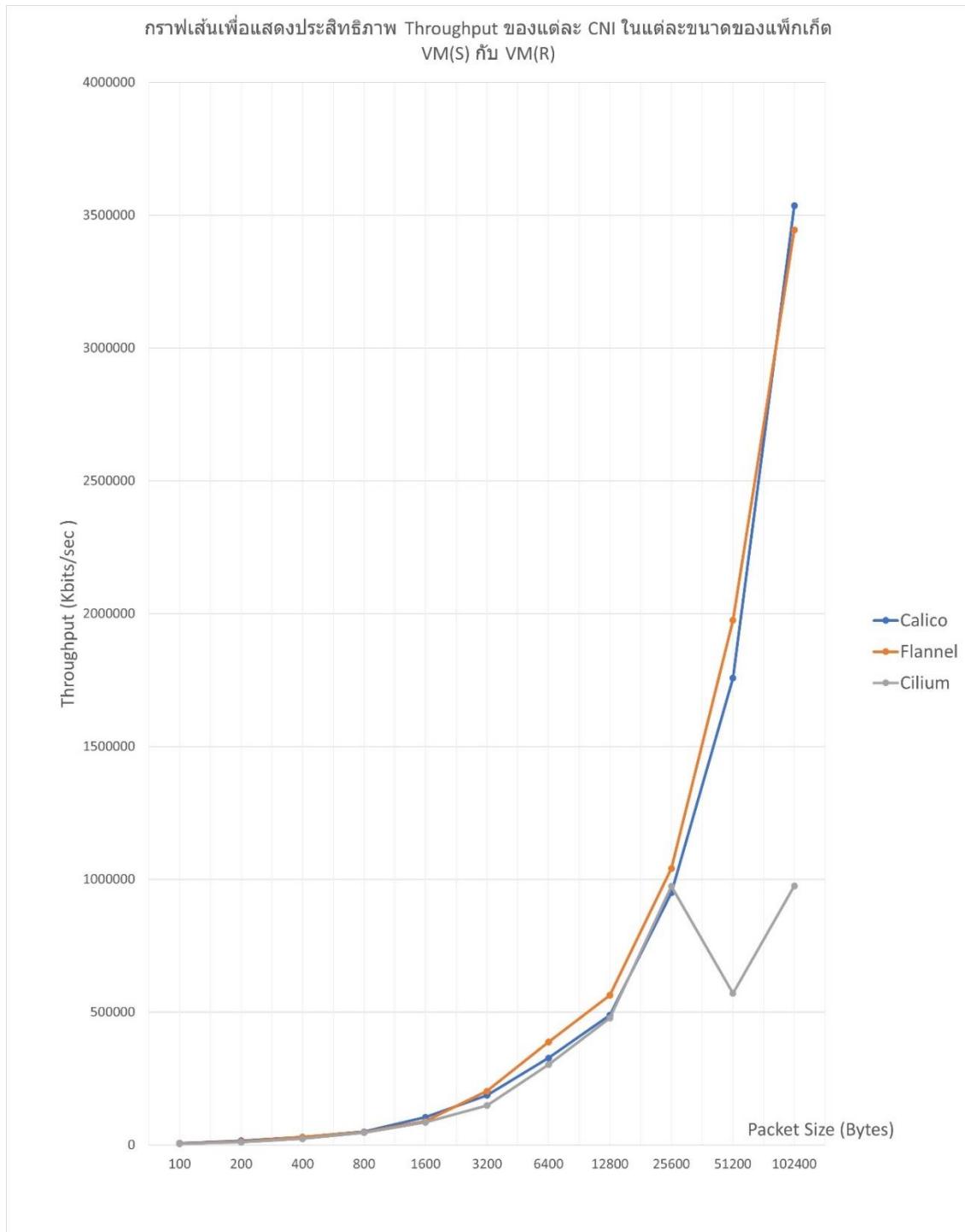


รูปที่ 4.12 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี Raspberry Pi 4 (S) กับ VM (R)

Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R), VM (S) กับ Raspberry Pi 4 (R) และ Raspberry Pi 4 (S) กับ VM (R) มี Latency ลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาทั้ง 3 CNI จะเห็นได้ว่า Calico มีค่า Latency ที่ดีกว่าทั้ง 2 ตัวที่มีผลใกล้เคียงกันโดย Cilium มี Latency เนลี่ยแย่ที่สุด

#### 4.3.4 VM (S) กับ VM (R)

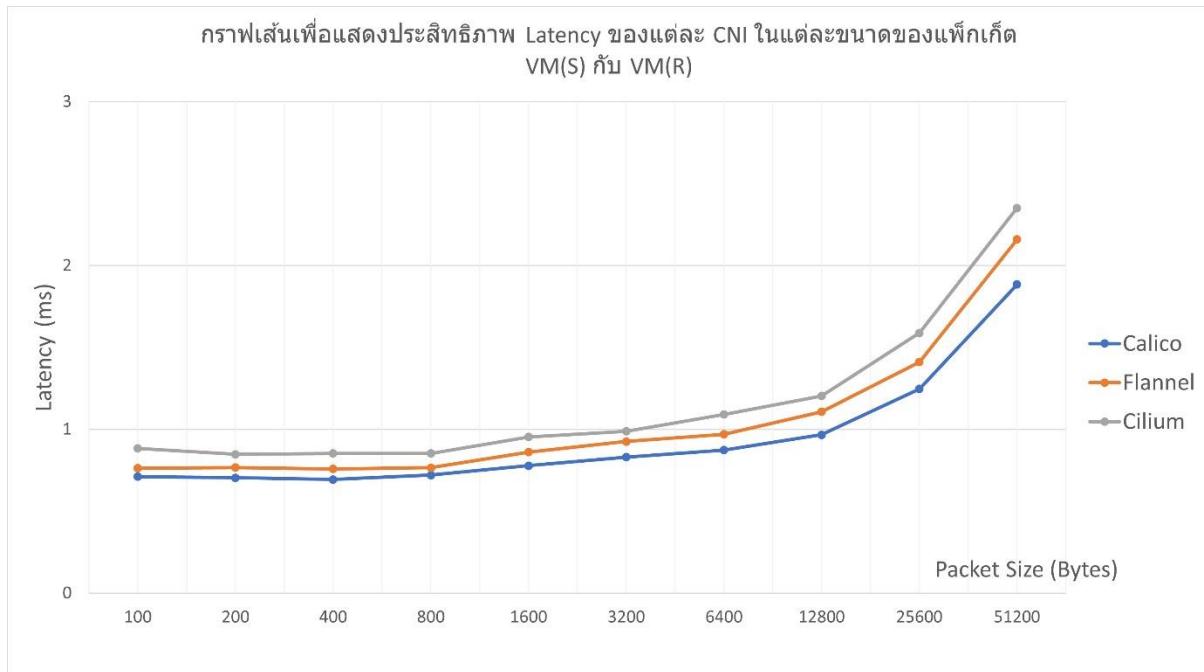
##### 4.3.4.1 Throughput



รูปที่ 4.13 กราฟเส้นแสดงประสิทธิภาพ Throughput ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ VM (R)

กราฟเส้น Throughput ของ Calico และ Flannel มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา แต่ Cilium มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาถึงขนาดแพ็คเก็ตที่ 25,600 ไปต่ หลังจากนั้นมีค่าที่ลดลงอย่างมากไม่สามารถคาดเดาได้ซึ่งผลลัพธ์ที่มีค่าน้อยที่สุดทุกขนาดแพ็คเก็ต จะได้เห็นได้ Flannel มีค่า Throughput มากที่สุดในทุกขนาดแพ็คเก็ต

#### 4.3.4.2 Latency



รูปที่ 4.14 กราฟเส้นแสดงประสิทธิภาพ Latency ของ CNI ในแต่ละขนาดของแพ็คเก็ต กรณี VM (S) กับ VM (R)

Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาทั้ง 3 CNI จะเห็นได้ชัดเจนว่า Calico มีค่า Latency ที่ดีกว่าและ Cilium มี Latency ที่แย่ที่สุด

## 4.4 ผลของการจับคู่ฮาร์ดแวร์ในการทดสอบในแต่ละ Container Network Interface (CNI)

### 4.4.1 Calico

จากการทดสอบเพื่อเก็บค่าเฉลี่ย Throughput (Kbits/sec) และ Latency (ms) ในแต่ละขนาดแพ็คเก็ตจะเห็นได้ว่ากราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้ส่งข้อมูลเป็น VM จะมีค่า Throughput ที่สูงกว่า แต่ถ้าหากเป็น Raspberry Pi 4 ในส่วนของค่า Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4 ซึ่งสามารถวิเคราะห์ Calico ได้ดังนี้ จากการจับคู่ฮาร์ดแวร์ VM เป็นผู้ส่งข้อมูล กับ VM เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพมากที่สุดในการทดสอบ และการจับคู่ฮาร์ดแวร์ Raspberry Pi 4 เป็นผู้ส่งข้อมูล กับ Raspberry Pi 4 เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพแย่ที่สุดในการทดสอบ

### 4.4.2 Flannel

จากการทดสอบเพื่อเก็บค่าเฉลี่ย Throughput (Kbits/sec) และ Latency (ms) ในแต่ละขนาดแพ็คเก็ตจะเห็นได้ว่าผลลัพธ์ที่ได้จะคล้ายๆกับ Calico กราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้ส่งข้อมูลเป็น VM จะมีค่า Throughput ที่สูงกว่าหากเป็น Raspberry Pi 4 ในส่วนของค่า Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4 ซึ่งสามารถวิเคราะห์ Flannel ได้ดังนี้ จากการจับคู่ฮาร์ดแวร์ VM เป็นผู้ส่งข้อมูล กับ VM เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพมากที่สุดในการทดสอบ และการจับคู่ฮาร์ดแวร์ Raspberry Pi 4 เป็นผู้ส่งข้อมูล กับ Raspberry Pi 4 เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพแย่ที่สุดในการทดสอบ

### 4.4.3 Cilium

จากการทดสอบเพื่อเก็บค่าเฉลี่ย Throughput (Kbits/sec) และ Latency (ms) ในแต่ละขนาดแพ็คเก็ตจะเห็นได้ว่ากราฟเส้น Throughput มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาจนถึงขนาดแพ็คเก็ตที่ 25,600 ไบต์ จากนั้นค่าที่ได้จากขนาดแพ็คเก็ตที่เหลือไม่สามารถคาดเดาหรือสรุปได้ข้อมูลยังไม่มากพอ จำเป็นต้องทดสอบเพิ่มจำนวนขนาดแพ็คเก็ตเพื่อวิเคราะห์เพิ่มเติม แต่สามารถวิเคราะห์ได้ว่าหากผู้ส่งข้อมูลเป็น VM จะมีค่า Throughput ที่สูงกว่าหากเป็น Raspberry Pi 4 ในส่วนของค่า Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา หากผู้รับข้อมูลเป็น VM จะมีค่า Latency ที่ดีกว่าหากเป็น Raspberry Pi 4 ซึ่งสามารถวิเคราะห์ Cilium ได้ดังนี้ จากการจับคู่ฮาร์ดแวร์ VM เป็นผู้ส่งข้อมูล กับ VM เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพ

มากที่สุดในการทดสอบ และการจับคู่ฮาร์ดแวร์ Raspberry Pi 4 เป็นผู้ส่งข้อมูล กับ Raspberry Pi 4 เป็นผู้รับข้อมูล ให้ผลประสิทธิภาพแยกต่างหากที่สุดในการทดสอบ

#### 4.5 ผลของการเปรียบเทียบแต่ละ Container Network Interface (CNI) ในแต่ละการจับคู่ฮาร์ดแวร์

##### 4.5.1 Raspberry Pi 4 (S) กับ Raspberry Pi 4 (R), VM (S) กับ Raspberry Pi 4 (R) และ Raspberry Pi 4 (S) กับ VM (R)

จากการทดสอบเพื่อเก็บค่าเฉลี่ย Throughput (Kbits/sec) และ Latency (ms) ในแต่ละขนาดแพ็คเก็ตจะเห็นได้ว่ากราฟแสดง Throughput ของ Calico และ Flannel มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา แต่ Cilium มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาถึงขนาดแพ็คเก็ตที่ 25,600 ไบต์ หลังจากนั้นมีค่าที่ลดลงอย่างมากไม่สามารถคาดเดาได้ จะได้เห็นได้ Flannel มีค่า Throughput มากที่สุดเมื่อมีขนาดแพ็คเก็ตที่มากกว่า 25,600 ไบต์ รองลงมาเป็น Calico แต่ Cilium นั้นแม้พลลัพธ์จะดูไม่เด่นเท่าไหร่แต่ในขนาดแพ็คเก็ตที่เล็กตั้งแต่ 100 ไบต์ ขึ้นไปกลับให้ค่า Throughput ที่ดีกว่าทุกๆ ตัวจนถึง 25,600 ไบต์ จึงมีค่า Throughput ที่แย่ที่สุดและคาดเดาจากข้อมูลยังไม่มากพอจำเป็นต้องทดสอบเพิ่มจำนวนขนาดแพ็คเก็ต เพื่อวิเคราะห์เพิ่มเติม ในส่วน Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาทั้ง 3 CNI จะเห็นได้ว่า Calico มีค่า Latency ที่ดีกว่าทั้ง 2 ตัวที่มีผลใกล้เคียงกันโดย Cilium มี Latency เฉลี่ยแย่ที่สุด

##### 4.5.2 VM (S) กับ VM (R)

จากการทดสอบเพื่อเก็บค่าเฉลี่ย Throughput (Kbits/sec) และ Latency (ms) ในแต่ละขนาดแพ็คเก็ต จะเห็นได้ว่ากราฟแสดง Throughput ของ Calico และ Flannel มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลา แต่ Cilium มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาถึงขนาดแพ็คเก็ตที่ 25,600 ไบต์ หลังจากนั้นมีค่าที่ลดลงอย่างมากไม่สามารถคาดเดาได้ จะได้เห็นได้ Flannel มีค่า Throughput มากที่สุดในทุกขนาดแพ็คเก็ต รองลงมาเป็น Calico แต่ Cilium นั้นผลลัพธ์แย่ที่สุดในทุกรณีและข้อมูลยังไม่มากพอจำเป็นต้องทดสอบเพิ่มจำนวนขนาดแพ็คเก็ต เพื่อวิเคราะห์เพิ่มเติม ในส่วน Latency มีลักษณะเพิ่มขึ้นเป็นรูปแบบพาราโบลาทั้ง 3 CNI จะเห็นได้ชัดเจนว่า Calico มีค่า Latency ที่ดีกว่าและ Cilium มี Latency ที่แย่ที่สุด

## บทที่ 5

### สรุปผลการดำเนินการศึกษา

#### 5.1 สรุปผลการดำเนินงาน

การศึกษาและทดสอบการทำงานของ Container Network Interface (CNI) 3 ประเภท คือ Calico, Flannel และ Cilium มีประสิทธิภาพในการทำงานที่แตกต่างกันในด้าน Throughput โดยที่ Flannel จะมีค่าเฉลี่ยที่ดีที่สุดจากการเปรียบเทียบทั้ง 3 ประเภทในทุก ๆ การจับคู่ของฮาร์ดแวร์ Calico มีผลลัพธ์ที่รองลงมา แต่ใกล้เคียงกับ Flannel อย่างมาก ในส่วน Cilium จะมีค่าเฉลี่ยที่แย่ แต่หากขนาดแพ็คเก็ตที่เล็กตั้งแต่ 100 ไบต์ ขึ้นไปมีผลลัพธ์ Throughput ที่ดีกว่าทุกๆ CNI จนถึง 25,600 ไบต์ ในทุกๆ การจับของฮาร์ดแวร์ยกเว้น สภาพแวดล้อมที่เป็นเครื่องจำลองเสมือน หรือ Virtual Machines (VM) ที่ CNI ประเภทอื่นให้ผลที่ดีกว่าในทุก ขนาดแพ็คเก็ตส่วนในด้าน Latency ทั้ง 3 ประเภทให้ผลที่ไม่แตกต่างกันมากแต่หากเปรียบเทียบ Calico มี ผลลัพธ์ที่ดีที่สุด ต่อมาคือ Flannel และผลลัพธ์ที่แย่ที่สุดคือ Cilium

Flannel CNI มีประสิทธิภาพในการทำงานที่โดดเด่นในด้าน Throughput ที่สูงและคงที่ Latency ที่ ดีในระดับที่ใช้งานได้อย่างยอดเยี่ยมเหมาะสมสำหรับการใช้งานในทุกๆ สภาพแวดล้อม

Calico CNI มีประสิทธิภาพในการทำงานที่โดดเด่นในด้าน Latency มีผลลัพธ์ที่ดีและคงที่ที่สุดแม้เพิ่ม ขนาดแพ็คเก็ตมากขึ้น ด้าน Throughput มีผลลัพธ์ที่ดี คงที่และสูงใกล้เคียงกับ Flannel เหมาะสำหรับการใช้ งานในทุกๆ สภาพแวดล้อม

Cilium CNI ข้อมูลยังไม่มากพอจำเป็นต้องทดสอบเพิ่มจำนวนขนาดแพ็คเก็ตเพื่อวิเคราะห์เพิ่มเติมแต่ เมื่อพิจารณา Cilium มีประสิทธิภาพในการทำงานที่โดดเด่นในด้าน Throughput ที่มีขนาดแพ็คเก็ตที่เล็ก ตั้งแต่ 100 ไบต์ ขึ้นไปให้ค่า Throughput ที่ดีกว่าทุกๆ CNI จนถึง 25,600 ไบต์ และ Latency มีผลลัพธ์ที่ ใกล้เคียงกับ Flannel แต่หากเป็นสภาพแวดล้อมที่เป็นเครื่องจำลองเสมือนนั้น CNI ตัวอื่นให้ผลลัพธ์ที่ดีกว่า

การศึกษาและทดสอบการทำงานของ CNI เพื่อหาผลลัพธ์ที่มีประสิทธิภาพดีที่สุด จะเห็นได้ว่าทุกๆ CNI มี จุดเด่นที่ต่างกัน หากต้องการผลลัพธ์ Throughput ที่ดีที่สุด มั่นคง Latency ไม่สูงและให้ผลลัพธ์ที่ดี Flannel จะเหมาะสม แต่หากต้องการ Latency ที่ดีที่สุด มั่นคง Throughput ให้ผลลัพธ์ที่ดี Calico จะเหมาะสมกว่า Flannel ซึ่งการเลือกใช้ Cilium จะเหมาะสมกับการนำไปใช้งานที่มีการใช้งานขนาดแพ็คเก็ตขนาดเล็ก

## 5.2 ปัญหาและอุปสรรค

ในการศึกษาการทำงานของ Container Network Interface พร้อมค้นคว้าหาข้อมูลมาประกอบจากบทความ งานวิจัย และหนังสือต่าง ๆ ซึ่งมีข้อมูลที่ยังไม่มากพอที่จะทำให้ผู้ดำเนินงานสามารถนำไปใช้ในการทดสอบและหัววิธีการทดสอบ จึงจำเป็นต้องมีเวลาในการศึกษารวมข้อมูลที่ใช้เวลานานพอสมควร

การกำหนดสภาพแวดล้อมในการทดสอบเป็นเรื่องที่ต้องตัดสินใจเป็นเวลานานเนื่องจากปัญหาด้านอุปกรณ์ที่ไม่เพียงพอจึงจำเป็นต้องหัววิธีปรับเปลี่ยนให้เหมาะสม นำมาสู่ความแตกต่างของประสิทธิภาพชาร์ดแวร์

การออกแบบการติดต่อสื่อสารภายในสภาพแวดล้อมเป็นปัญหาที่พิจารณาและแก้ไขเป็นเวลานานจาก การทดสอบและนำผลทดสอบเบื้องต้นมาเปรียบเทียบเพื่อจะกำหนดและเลือกรูปแบบการติดต่อสื่อสารกันโดย เลือกใช้สวิตซ์เป็นศูนย์กลางเชื่อมต่อระบบ ให้อุปกรณ์ Raspberry Pi 4 และ Lenovo ThinkSystem SR530 Server เพื่อรับส่งข้อมูล

การเลือกใช้ประเภทของ Container Network Interface นั้นมีหลากหลายตัวและมีโครงสร้างเครือข่ายที่แตกต่างการเลือกใช้จึงสำคัญ ผู้ดำเนินงานมีการเลือกใช้ Kubernetes เข้ามาช่วยในการทดสอบ เกิดปัญหาการนำมาปรับใช้ที่ยุ่งยากเนื่องจาก Kubernetes มีการอัปเดตอยู่ตลอดเวลาแต่ Container Network Interface บางตัวไม่สามารถรองรับและใช้งานได้ สาเหตุเกิดจากหลากหลายกรณีอาทิเช่น Container Network Interface บางตัวยังเป็นเวอร์ชันเก่าและไม่มีการอัปเดตเป็นเวลานาน การกำหนดค่าที่ซับซ้อนซึ่งไม่เหมาะสมกับการนำมหาทดสอบ

ระบบปฏิบัติการที่เลือกใช้เป็นเซิร์ฟเวอร์เพื่อให้ได้เห็นผลประสิทธิภาพและใกล้เคียงกับการใช้งานจริง ทำให้เครื่องมือซอฟต์แวร์ที่จะนำมหาทดสอบมีค่อนข้างจำกัด ต้องศึกษาหาเครื่องมือที่เหมาะสมและวิธีการใช้อย่างถูกต้องเพื่อออกแบบการทดสอบให้อกมาเห็นผลลัพธ์ได้อย่างชัดเจนและถูกต้องที่สุด

จากปัญหาและอุปสรรคข้างต้นทำให้ต้องใช้เวลาศึกษา ค้นคว้า มากกว่าทำการทดสอบและวิเคราะห์ ผลอย่างมากทำให้เกิดการล่าช้าในการนำผลลัพธ์มาแสดงและปรับปรุงการนำเสนอเพื่อผู้ที่สนใจได้เข้าใจอย่างง่ายที่สุด

### 5.3 ข้อเสนอแนะ

ในการทดสอบประสิทธิภาพของ Container Network Interface ควรมีสภาพแวดล้อมที่ซัพเจน มีฮาร์ดแวร์ที่มีประสิทธิภาพเหมือนกัน ทดสอบด้วยจำนวนขนาดแพ็คเก็ตและจำนวนรอบในการทดสอบเพิ่มมากขึ้น เพื่อที่จะทำให้สามารถวัดประสิทธิภาพที่เหมาะสมและเป็นธรรม ซึ่งจำเป็นต่อการสรุปผลลัพธ์ จะทำให้เห็นภาพรวมของผลการทดสอบที่ซัมมาอย่างขึ้น

ในการศึกษาข้อมูลเกี่ยวกับข้อดีข้อเสียที่เป็นต้องมีความรู้พื้นฐานถึงความเชี่ยวชาญด้านเครือข่าย และ Docker Container ในระดับหนึ่งก่อนที่จะทำการเข้าใจถูกที่ควรเริ่มในการทดสอบและเริ่มใช้ Container Network Interface

ศึกษาการใช้งาน Kubernetes ในทุก ๆ คำสั่งการใช้งานพร้อมศึกษาการทำงานของ Container Network Interface และ Docker Container ให้เข้าใจและชำนาญมากขึ้น เพื่อที่จะสามารถเขียนฟังก์ชันที่มีความซับซ้อนมากขึ้นและสามารถเลือกใช้ Container Network Interface ที่เหลือได้อีกมากมายที่มีความแตกต่างกันในด้านประสิทธิภาพและความแตกต่างกันของโครงสร้างเครือข่าย

## บรรณานุกรม

### References

- [1] Hausenblas, M. (2017). Container Networking. O'Reilly. Retrieved March 10, 2024, from <https://www.oreilly.com/library/view/container-networking/9781492036845/>
- [2] Cloud Native Computing Foundation. (2017, May 23). CNCF hosts container networking interface (CNI). Retrieved March 10, 2024, from <https://www.cncf.io/projects/container-network-interface-cni/>
- [3] Huawei. (2023, September 29). Kubernetes network model. Retrieved from <https://support.huawei.com/enterprise/en/doc/EDOC1100086966>
- [4] Cloudflare. (n.d.). What is BGP?. Retrieved March 10, 2024, from <https://developers.cloudflare.com/network-interconnect/set-up-cni/configure-bgp/>
- [5] Docker. (n.d.). What is a container?. Retrieved March 10, 2024, from <https://www.docker.com/>
- [6] DataCamp. (2023, September 29). Tutorial: Machine Learning Pipelines, MLOps, and Deployment. Retrieved March 10, 2024, from [https://www.datacamp.com/tutorial/tutorial-machine-learning-pipelines-mlops-deployment?utm\\_source=google&utm\\_medium=paid\\_search&utm\\_campaignid=19589720824&utm\\_adgroupid=143216588537&utm\\_device=c&utm\\_keyword&utm\\_matchtype&utm\\_network=g&utm\\_adpostion&utm\\_creative=665485585140&utm\\_targetid=dsa-1947282172981&utm\\_loc\\_interest\\_ms&utm\\_loc\\_physical\\_ms=9073383&utm\\_content=dsa~page~community-tuto&utm\\_campaign=230119\\_1-sea~dsa~tutorials\\_2-b2c\\_3-row-p2\\_4-prc\\_5-na\\_6-na\\_7-le\\_8-pdsh-go\\_9-na\\_10-na\\_11-na-Itsjul23&gclid=CjwKCAjww7KmBhAyEiwA5-PUSu5yJ\\_ALnATgYMVvhMm1ngsA1P0l5WtAMf7WoeXmBoBWOAHqJExvtRoCFxIQAvD\\_BwE&fbclid=IwAR2xKFFWLHb\\_XmdLWonsuU1do9UuSt6wZb0FJ4JCDO-0-VwzvtbKJzWWsug](https://www.datacamp.com/tutorial/tutorial-machine-learning-pipelines-mlops-deployment?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=143216588537&utm_device=c&utm_keyword&utm_matchtype&utm_network=g&utm_adpostion&utm_creative=665485585140&utm_targetid=dsa-1947282172981&utm_loc_interest_ms&utm_loc_physical_ms=9073383&utm_content=dsa~page~community-tuto&utm_campaign=230119_1-sea~dsa~tutorials_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-na_10-na_11-na-Itsjul23&gclid=CjwKCAjww7KmBhAyEiwA5-PUSu5yJ_ALnATgYMVvhMm1ngsA1P0l5WtAMf7WoeXmBoBWOAHqJExvtRoCFxIQAvD_BwE&fbclid=IwAR2xKFFWLHb_XmdLWonsuU1do9UuSt6wZb0FJ4JCDO-0-VwzvtbKJzWWsug)
- [7] Netbeez. (2023, February 14). iPerf2 vs iPerf3: เปรียบเทียบเครื่องมือวัดประสิทธิภาพเครือข่าย. Retrieved March 10, 2024, from <https://netbeez.net/blog/iperf2-vs-iperf3-2/>

- [8] ZenTaoPM. (2023, September 29). kubernetes network model. Retrieved March 10, 2024, from <https://www.zentao.pm/blog/kubernetes-network-model-1379.html>
- [9] Kang, Z., An, K., Gokhale, A., & Pazandak, P. N. (2021, April). A comprehensive performance evaluation of different Kubernetes CNI plugins for edge-based and containerized publish/subscribe applications. In 2021 15th IEEE International Conference on Cloud Engineering (IC2E) (pp. 4970-4973). IEEE.
- [10] atkus, A., & Tamulevičius, A. (2022). Overview of Kubernetes CNI plugins performance. Electronics and electrical engineering Elektronika ir elektros inžinerija, 24(1), 13-23.
- [11] Kubernetes Container Network Interface (CNI). Retrieved August 16, 2023, from <https://www.cni.dev/docs/>
- [12] Kubernetes. (n.d.). Objects in Kubernetes. Retrieved August 16, 2023, from <https://kubernetes.io/docs/concepts/overview/working-with-objects/>
- [13] Shanmugam, K. (2022). IoT Edge Computing with MicroK8s: A hands-on approach to building, deploying, and distributing production-ready Kubernetes on IoT and Edge platforms. Packt Publishing
- [14] McWhorter, P. (2021). Networking and Kubernetes: A practical guide to building, deploying, and managing Kubernetes networks. O'Reilly Media
- [15] Hausenblas, M. (2020). Container Networking: From Docker to Kubernetes. Google Books. ISBN 9781492036845.
- [16] Dugan, J. (2014). iPerf3: A new implementation of a high-performance network bandwidth measurement tool. [Online]. Available at: <https://iperf.fr/iperf-doc.php>
- [17] Ajarn WS. "Ping? RTT? Latency? Lag? & Jitter? มันคืออะไร?" Medium, Retrieved September 29, 2023 from <https://ajarn-ws.medium.com/ping-rtt-latency-lag-jitter-d7a37dab36c7>
- [18] Oap. "Kubernetes Cluster Network: CNI." Medium, 13 Apr. 2023, Retrieved September 29, 2023 from <https://medium.com/@oap.py/kubernetes-cluster-network-cni-ec0b7cab7b1e>

- [19] Alibaba Cloud. "Kubernetes CNIS และ Cni ปลั๊กอิน." Blog, Alibaba Cloud, Retrieved September 29, 2023 from [https://www.alibabacloud.com/blog/getting-started-with-kubernetes-%7C-kubernetes-cnis-and-cni-plug-ins\\_596330](https://www.alibabacloud.com/blog/getting-started-with-kubernetes-%7C-kubernetes-cnis-and-cni-plug-ins_596330)
- [20] ZenTaoPM. kubernetes network model. Retrieved September 29, 2023 from <https://www.zentao.pm/blog/kubernetes-network-model-1379.html>
- [21] TechTarget. "Explore Network Plugins for Kubernetes: CNI Explained." Retrieved September 30, 2023 from <https://www.techtarget.com/searchitoperations/tip/Explore-network-plugins-for-Kubernetes-CNI-explained>.
- [22] Tigera. "Kubernetes CNI Explained." Tigera, 16 Oct. 2023, Retrieved September 30, 2023 from <https://www.tigera.io/learn/guides/kubernetes-networking/kubernetes-cni/>.
- [23] Lenovo. "Lenovo ThinkSystem SR530 Server (Xeon SP Gen 1 / Gen 2) Product Guide (withdrawn product)." Lenovo Press, 12 Dec. 2023, Retrieved 30 Sep. 2023 from <https://lenovopress.lenovo.com/lp1045-thinksystem-sr530-server>.
- [24] Cisco Systems, Inc. (2016, September). Cisco Catalyst 3650 Switches Hardware Installation Guide [Text Part Number: OL-29734-01]. Retrieved from [https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3650/hardware/installation\\_guide/Cat3650hig\\_book.pdf](https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3650/hardware/installation_guide/Cat3650hig_book.pdf)