การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์

วิเชพ ใจบุญ

วิทยานิพนธ์เสนอบัณฑิตวิทยาลัย มหาวิทยาลัยนเรศวร
เพื่อเป็นส่วนหนึ่งของการศึกษา หลักสูตรปรัชญาดุษฎีบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
ปีการศึกษา 2565

การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์

วิเชพ ใจบุญ

วิทยานิพนธ์เสนอบัณฑิตวิทยาลัย มหาวิทยาลัยนเรศวร
เพื่อเป็นส่วนหนึ่งของการศึกษา หลักสูตรปรัชญาดุษฎีบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
ปีการศึกษา 2565

วิทยานิพนธ์ เรื่อง "การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์"

ของ วิเชพ ใจบุญ

ได้รับการพิจารณาให้นับเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

คณะกรรมการสอบวิทยานิพนธ์

---------- ประธานกรรมการสอบวิทยานิพนธ์

(รองศาสตราจารย์ ดร.เอกรัฐ บุญเชียง)

---------- ประธานที่ปรึกษาวิทยานิพนธ์

(ผู้ช่วยศาสตราจารย์ ดร.วินัย วงษ์ไทย)

---------- กรรมการผู้ทรงคุณวุฒิภายใน

(รองศาสตราจารย์ ดร.จักรกฤษณ์ เสน่ห์ นมะหุต)

---------- กรรมการผู้ทรงคุณวุฒิภายใน

(ผู้ช่วยศาสตราจารย์ ดร.จันทร์จิรา พยัคฆ์เพศ)

---------- กรรมการผู้ทรงคุณวุฒิภายใน

(ผู้ช่วยศาสตราจารย์ ดร.ธนะธร พ่อค้า)

---------- กรรมการผู้ทรงคุณวุฒิภายใน

(ผู้ช่วยศาสตราจารย์ ดร.เกรียงศักดิ์ เตมีย์)

อนุมัติ

(รองศาสตราจารย์ ดร.กรองกาญจน์ ชูทิพย์)

คณบดีบัณฑิตวิทยาลัย

ชื่อเรื่อง การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์

ผู้วิจัย วิเชพ ใจบุญ

ประธานที่ปรึกษา ผู้ช่วยศาสตราจารย์ ดร.วินัย วงษ์ไทย

ประเภทสารนิพนธ์ วิทยานิพนธ์ ปร.ด. วิทยาการคอมพิวเตอร์, มหาวิทยาลัยนเรศวร, 2565

คำสำคัญ ระบบบันทึกเหตุการณ์, การประมวลผลแบบกลุ่มเมฆ, การให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์, โอเพนสแต็ก

บทคัดย่อ

งานวิจัยนี้มีวัตถุประสงค์เพื่อสร้างต้นแบบและปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามทางไซเบอร์ที่อาจส่งผลกระทบต่อระบบการประมวลผลแบบกลุ่มเมฆตามที่ระบุโดยองค์กร Cloud Security Alliance ผู้วิจัยมีแนวคิดในการดำเนินการวิจัย 1) เพื่อประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยก่อนหน้า ให้สามารถทำงานบนสภาพแวดล้อมของโอเพนสแต็ก 2) เพื่ออภิปรายผลลัพธ์การออกแบบและการทำให้เกิดผลของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็ก 3) เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับสำหรับสภาพแวดล้อมของโอเพนสแต็ก และ 4) เพื่อทดสอบและประเมินประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็ก ผลการวิจัยพบว่า 1) สามารถประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยก่อนหน้า ให้สามารถทำงานบนสภาพแวดล้อมของโอเพนสแต็กได้ โดยสร้างสภาพแวดล้อมนี้จากการติดตั้งระบบปฏิบัติการคลาวด์โอเพนสแต็กบนคอมพิวเตอร์กายภาพจำนวนสองเครื่องและเชื่อมต่อกันผ่านระบบเครือข่าย 2) มีการอภิปรายผลลัพธ์การออกแบบและการทำให้เกิดผล รวมทั้งมีการปรับปรุงประสิทธิของระบบบันทึกเหตุการณ์ให้มีประสิทธิภาพดีขึ้นกว่าเดิม โดยการประยุกต์ใช้หลักการเขียนโปรแกรมซ้อกเก็ตเพื่อทำให้ระบบบันทึกเหตุการณ์สามารถแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์กายภาพที่อยู่ในเครือข่ายเดียวกันได้ ประยุกต์ใช้หลักการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรดเพื่อทำให้ระบบบันทึก

เหตุการณ์สามารถตรวจสอบเครื่องเสมือนที่อยู่บนโฮสต์เดี่ยวกันได้มากกว่าหนึ่งเครื่องพร้อมกัน และ 3) ผลการวิจัยนี้ยังสามารถบอกถึงประสิทธิภาพและข้อจำกัดของระบบบันทึกเหตุการณ์ เพื่อเป็นแนวทางการประยุกต์ใช้งานระบบบันทึกเหตุการณ์ในเชิงพาณิชย์

Title	LOGGING SYSTEMS PERFORMANCE IMPROVEMENT FOR OPENSTACK ENVIRONMENT IN INFRASTRUCTURE AS A SERVICE CLOUD
Author	Wichep Jaiboon
Advisor	Assistant Professor Dr. Winai Wongthai, Ph.D.
Academic Paper	Ph.D. Dissertation in Computer Science - (Type 2.1), Naresuan University, 2022
Keywords	Logging system, Cloud computing, Infrastructure as a service cloud, OpenStack

ABSTRACT

This research aims to create a prototype and improve the efficiency of a logging system for the OpenStack environment in Infrastructure as a Service. The aim is to mitigate risks associated with cyber threats that may impact cloud computing systems, as identified by the Cloud Security Alliance (CSA). The researcher has the following research plan: 1) to apply the logging system developed from previous research to work on the OpenStack environment, 2) to discuss the results of the design and implementation of the logging system for the OpenStack environment, 3) to improve the efficiency of the logging system to be compatible with the OpenStack environment, and 4) to test and evaluate the performance of the logging system in the OpenStack environment. The results indicate that 1) the previous research's logging system can be applied to work in an OpenStack environment. This environment can be achieved by deploying a cloud operating system on two physical computers and connecting them through a network system. 2) There is a discussion on the results of the design and implementation, along with the improvement of the efficiency of the logging system to perform better than before. The improvement is achieved by applying the principle of socket programming to allow the logging system to exchange data between physical computers within the same network. In addition, the principles of parallel programming are also applied, enabling the logging system to monitor multiple virtual machines on a single host simultaneously. Furthermore, 3) This research result also provides the performance and limitations of the logging system, serving as a guideline for applying the event logging system in commercial applications.

ประกาศคุณูปการ

ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงในความกรุณาของประธานที่ปรึกษาวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร.วินัย วงษ์ไทย ที่ได้สละเวลาอันมีค่าในการอบรมสั่งสอน แนะนำองค์ความรู้ และให้คำปรึกษาด้านเทคนิคที่เกี่ยวข้องกับงานวิจัย ตลอดจนข้อเสนอแนะต่าง ๆ สำหรับการทำวิทยานิพนธ์เล่มนี้ จนเสร็จสมบูรณ์เป็นอย่างดียิ่ง

ขอกราบขอบพระคุณท่านคณาจารย์กรรมการสอบวิทยานิพนธ์ทุกท่าน ประกอบด้วย ประธานกรรมการสอบวิทยานิพน์และกรรมการผู้ทรงคุณวุฒิภายนอก รองศาสตราจารย์ ดร.เอกรัฐ บุญเชียง ประธานที่ปรึกษาวิทยานิพนธ์ผู้ช่วยศาสตราจารย์ ดร. วินัย วงษ์ไทย ผู้ทรงคุณวุฒิภายใน รองศาสตราจารย์ ดร. จักรกฤษณ์ เสน่ห์ นมะหุต ผู้ช่วยศาสตราจารย์ ดร. เกรียงศักดิ์ เตมีย์ ผู้ช่วยศาสตราจารย์ ดร. จันทร์จิรา พยัคฆ์เพศ และผู้ช่วยศาสตราจารย์ ดร. ธนะธร พ่อค้า ที่ได้กรุณาให้คำแนะนำ ข้อเสนอแนะ ปรับปรุงพัฒนางานวิจัยด้วยความเอาใจใส่จนทำให้วิทยานิพนธ์เล่มนี้สำเร็จลุล่วงอย่างสมบูรณ์

ขอขอบพระคุณรุ่นพี่ เพื่อน และผู้ที่เกี่ยวข้องทุกท่านในภาควิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ คณะวิทยาศาสตร์ มหาวิทยาลัยนเรศวร ที่มีส่วนร่วมในการทำวิทยานิพนธ์เล่มนี้จนประสบผลสำเร็จ

คุณค่าและคุณประโยชน์อันพึงจะมีจากวิทยานิพนธ์เล่มนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน และหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์เล่มนี้จะเป็นประโยชน์ต่อผู้ที่สนใจนำไปใช้ได้เป็นอย่างดียิ่ง หากมีข้อบกพร่องประการใดที่อาจจะเกิดขึ้นในวิทยานิพนธ์นั้น ผู้วิจัยขอน้อมรับและยินดีรับฟังคำแนะนำอันจะเป็นประโยชน์ในการพัฒนาต่อไปในอนาคต

วิเชพ ใจบุญ

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

คำอภิธานศัพท์ที่ใช้ในการวิจัย

คำศัพท์ภาษาไทย	คำศัพท์ภาษาอังกฤษ	คำอธิบาย	ที่มา	คำที่ใช้ในงานวิจัย
การประมวลผลแบบกลุ่มเมฆ	Cloud Computing	แบบจำลองสำหรับการใช้ทรัพยากรคอมพิวเตอร์ เช่น เครือข่าย เครื่องเซิร์ฟเวอร์ หน่วยเก็บข้อมูล โปรแกรมประยุกต์หรือแอปพลิเคชัน และบริการอื่น ๆ ซึ่งทรัพยากรเหล่านี้สามารถเข้าถึงได้อย่างสะดวกจากอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ โดยเชื่อมต่อผ่านเครือข่ายได้จากทุกที่ การขอใช้หรือยกเลิกการใช้ทรัพยากรเหล่านี้สามารถทำได้อย่างรวดเร็วและปรับเปลี่ยนไปตามความต้องการ โดยใช้ความพยายามในการจัดการหรือติดต่อกับผู้ให้บริการน้อยสุด	(Mell & Grance, 2011)	คลาวด์
ระบบการประมวลผลแบบ	Infrastructure as a Service	ในงานวิจัยนี้จะถึงระบบการประมวลผล	-	IaaS คลาวด์

คำศัพท์ภาษาไทย	คำศัพท์ภาษาอังกฤษ	คำอธิบาย	ที่มา	คำที่ใช้ในงานวิจัย
กลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์	Public Cloud	แบบกลุ่มเมฆสาธารณะประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ ว่าเป็นการจัดเตรียมหน่วยประมวลผลกลาง หน่วยเก็บข้อมูล เครือข่าย และทรัพยากรคอมพิวเตอร์พื้นฐานอื่น ๆ ซึ่งผู้ใช้บริการสามารถปรับใช้ตามความต้องการ รวมถึงควบคุมระบบปฏิบัติการ การจัดเก็บ การปรับใช้แอปพลิเคชัน และส่วนประกอบของเครือข่ายบางส่วนของคอมพิวเตอร์แม่งาน (Host Computer) หรือโฮสต์ไว้สำหรับการใช้งานแบบเปิดโดยให้ผู้ใช้บริการทั่วไปใช้งาน		
ไฟล์ข้อมูลที่มี	Critical File	ไฟล์ต่าง ๆ ที่เก็บไว้ใน	(W. Wongthai,	ไฟล์ที่มี

คำศัพท์ภาษาไทย	คำศัพท์ภาษาอังกฤษ	คำอธิบาย	ที่มา	คำที่ใช้ในงานวิจัย
ละเอียดอ่อนหรือมีความสำคัญของผู้ใช้บริการ		คอมพิวเตอร์เสมือน (Virtual Machine) ของผู้ใช้บริการ ซึ่งไฟล์เหล่านี้สามารถเป็นไฟล์ชนิดใดๆ ก็ได้ ตัวอย่างเช่น ไฟล์ข้อความ ไฟล์นามสกุล exe หรือไฟล์ฐานข้อมูล ไฟล์เหล่านี้ถือเป็นทรัพย์สินที่สำคัญและมีมูลค่าทางธุรกิจของผู้ใช้บริการ ที่ไม่ต้องการให้บุคคลใดเข้าถึงไฟล์นอกจากตนเองหรือผู้ที่ได้รับอนุญาตเท่านั้น รวมทั้งไม่ต้องการให้สูญหายหรือรั่วไหล	F. Rocha, & A. Van Moorsel, 2013a)	ความสำคัญของผู้ใช้บริการ
ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภท	IaaS OpenStack Logging System (IOLS)	ระบบบันทึกเหตุการณ์ที่ติดตั้งในสภาพแวดล้อมของโอเพนสแต็ก (OpenStack) ในระบบการประมวลผลแบบกลุ่มเมฆสาธารณะประเภทการ	-	IaaS OpenStack Logging System หรือ IOLS

คำศัพท์ภาษาไทย	คำศัพท์ภาษาอังกฤษ	คำอธิบาย	ที่มา	คำที่ใช้ในงานวิจัย
การให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์		ให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์		
ภาระความรับผิด	Accountability	ในงานวิจัยนี้จะหมายความว่า ภาระความรับผิดเมื่อนำมาใช้ในคลาวด์ คือ สถานการณ์ที่ผู้ตรวจสอบทราบถึงตัวบุคคลที่ต้องสงสัยว่ามีส่วนต้องรับผิดชอบต่อเหตุการณ์ที่เกิดขึ้นแล้ว และเป็นสาเหตุของภัยคุกคามต่อคลาวด์ตามรายงานขององค์กร Cloud Security Alliance (CSA) หรือภัยคุกคามอื่น และสามารถจัดเตรียมหลักฐานเพื่ออธิบายเหตุการณ์เหล่านั้นที่เกิดขึ้นแล้ว	(ปกรณ์ จันทร์อินทร์, 2561)	ภาระความรับผิด

บทที่ 1

บทนำ

สำหรับเนื้อหาในบทที่ 1 ผู้วิจัยจะกล่าวถึงภาพรวมทั้งหมดในวิทยานิพนธ์เล่มนี้เพื่อให้ผู้อ่านได้เข้าใจอย่างง่ายก่อนที่จะกล่าวถึงรายละเอียดเชิงลึกของเอกสารและเนื้อหาที่เกี่ยวข้อง รวมถึงงานวิจัยต่าง ๆ ในบทที่ 2 ต่อไป วิทยานิพนธ์เรื่อง การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ มีการแบ่งเนื้อหาและรายละเอียดที่แสดงให้เห็นถึงภาพรวมความเป็นมาและความสำคัญของปัญหา ประเด็นปัญหาวิจัยหลัก รวมถึงเป้าหมายของงานวิจัย โดยแยกเป็นหัวข้อย่อยดังต่อไปนี้

1. ความเป็นมาและความสำคัญของปัญหา
2. ปัญหาวิจัยหลัก
3. เป้าหมายของงานวิจัย
4. วัตถุประสงค์ของการศึกษา
5; ขอบเขตของงานวิจัย
6. ประโยชน์ที่คาดว่าจะได้รับ

ความเป็นมาและความสำคัญของปัญหา

สถาบันมาตรฐานและเทคโนโลยีแห่งชาติ (National Institute of Standards and Technology: NIST) ได้ระบุเกี่ยวกับการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) หรือคลาวด์ (Cloud) ว่าเป็นแบบจำลองสำหรับการใช้ทรัพยากรคอมพิวเตอร์ เช่น เครือข่าย (Network) เครื่องเซิร์ฟเวอร์ (Server) หน่วยเก็บข้อมูล (Storage) โปรแกรมประยุกต์หรือแอปพลิเคชัน (Application) และบริการอื่น ๆ ซึ่งทรัพยากรเหล่านี้สามารถเข้าถึงได้อย่างสะดวกจากอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ โดยเชื่อมต่อผ่านเครือข่ายได้จากทุกที่ การขอใช้หรือยกเลิกการใช้ทรัพยากรเหล่านี้สามารถทำได้อย่างรวดเร็วและปรับเปลี่ยนไปตามความต้องการ โดยใช้ความพยายามในการจัดการหรือติดต่อกับผู้ให้บริการ (Provider) น้อยที่สุด (Mell & Grance, 2011)

Gartner (2017, 2022) ได้กล่าวว่า คลาวด์ถูกนำมาใช้งานมากขึ้นในองค์กรเนื่องจากความสามารถในการปรับขนาด (Scalability) ได้และมีความยืดหยุ่น (Elasticity) อีกทั้งคลาวด์ยังสามารถช่วยลดต้นทุนการจัดการเทคโนโลยีสารสนเทศในองค์กรตามที่กล่าวไว้ใน Albaroodi,

Manickam & Bawa (2014); Chan-In & Wongthai (2016, 2017); Runathong, Wongthai & Panithansuwan (2017); Wongthai (2014); Wongthai & Van Moorsel (2016a, 2016b, 2017) ปัจจุบันคลาวด์ยังคงได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ เห็นได้จากรายงานของ Gartner (2022) ซึ่งคาดการณ์มูลค่าใช้จ่ายของผู้ใช้บริการบนคลาวด์สาธารณะทั่วโลกจะสูงถึง 591.8 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2566 คิดเป็นร้อยละ 20.7 เพิ่มขึ้นจาก 490.3 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2565 และในปี พ.ศ. 2565 มีการเติมโตคิดเป็นร้อยละ 18.8 เพิ่มขึ้นจาก 412.6 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2564

Mell & Grance (2011) ได้แบ่งคลาวด์ตามลักษณะขอบเขตของการให้บริการ (Service Model) ไว้ 3 ประเภท ได้แก่ การให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ (Infrastructure as a Service: IaaS) การให้บริการแพลตฟอร์ม (Platform as a Service: PaaS) และการให้บริการซอฟต์แวร์ (Software as a Service: SaaS) และแบ่งคลาวด์ตามลักษณะขอบเขตของการจัดการ (Deployment Model) ไว้ 4 ประเภท ได้แก่ คลาวด์ภายในองค์กร (Private Cloud) คลาวด์สาธารณะ (Public Cloud) คลาวด์สำหรับผู้ใช้เฉพาะกลุ่ม (Community Cloud) และคลาวด์แบบผสมผสาน (Hybrid Cloud) สำหรับเนื้อหาในวิทยานิพนธ์เล่มนี้จะมุ่งเน้นที่คลาวด์ สาธารณะประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ โดยแบบจำลองการให้บริการ นี้จะเกี่ยวข้องกับการจัดเตรียมจำนวนหน่วยประมวลผลกลาง หน่วยเก็บข้อมูล เครือข่าย และทรัพยากรคอมพิวเตอร์พื้นฐานอื่น ๆ โดยผู้ให้บริการ (Provider) เป็นผู้จัดเตรียมไว้เพื่อให้บริการกับผู้ใช้บริการ (Consumer) ทั่วไป ดังนั้นองค์กรธุรกิจ (Business Organization) หรือผู้ให้บริการคลาวด์ (Cloud Provider) สามารถดำเนินการบริหารจัดการ และเป็นเจ้าของคลาวด์สาธารณะได้ จากผลการสำรวจขององค์กรการ์เนอร์ (Gartner, 2022) แสดงให้เห็นว่าคลาวด์สาธารณะประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ (IaaS) ยังที่มีอัตราการเติบโตสูงสุด ดังนั้นจึงมีโอกาสประสบปัญหาที่เกี่ยวข้องกับความปลอดภัยมากกว่าคลาวด์ประเภทอื่น ๆ อีกทั้งการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ (IaaS) ยังเป็นโครงสร้างพื้นฐานสำหรับการสร้างคลาวด์ประเภทอื่น เช่น การให้บริการแพลตฟอร์ม (PaaS) และการให้บริการซอฟต์แวร์ (SaaS) หากประสบปัญหาที่เกี่ยวข้องกับความปลอดภัยย่อมส่งผลกระทบต่อเนื่องไปยังคลาวด์ประเภทอื่นอย่างหลีกเลี่ยงไม่ได้

ทั้งนี้ ผู้ให้บริการจะจัดเตรียม IaaS คลาวด์ไว้สำหรับผู้ใช้บริการหรือลูกค้า เช่น หน่วยประมวลผลกลางหรือซีพียู หน่วยเก็บข้อมูล เครือข่าย แอปพลิเคชัน ระบบปฏิบัติการ และทรัพยากรพื้นฐานอื่น ๆ อย่างไรก็ตาม CSA (2019); Ristov, Gusev & Donevski (2013);

Wongthai (2014) ได้อธิบายเกี่ยวความกังวลด้านความปลอดภัยใน IaaS คลาวด์ ในประเด็นเรื่องของความลับ (Confidentiality) ความถูกต้องสมบูรณ์ (Integrity) และความพร้อมใช้งาน (Availability) ตัวอย่างเช่น ไฟล์ที่มีความสำคัญของผู้ใช้บริการ (Critical File) ที่อยู่ใน IaaS คลาวด์ ซึ่งประเด็นปัญหาเหล่านี้เป็นสิ่งที่ขัดขวางการนำคลาวด์ไปใช้งาน จากประเด็นปัญหาดังกล่าวได้รับการพิจารณาโดยนักวิจัยหลายคน โดยเฉพาะอย่างยิ่งองค์กร Cloud Security Alliance หรือ CSA ที่ทำการวิจัย รวบรวม และระบุปัญหาเกี่ยวข้องกับภัยคุกคามต่อคลาวด์มากว่า 14 ปี มีการเผยแพร่รายงานประมาณ 23 ฉบับ และได้รับการแปลเป็นภาษาอื่นอีกห้าภาษา ตัวอย่างเช่น รายงานเรื่อง “Top Threats to Cloud Computing, Version 1.0” ในปี ค.ศ. 2010 ถึงรายงานเรื่อง “Top Threats to Cloud Computing Pandemic Eleven” ในปี ค.ศ. 2022 (CSA, 2010, 2022) เพื่อตรวจสอบวิธีการป้องกันและการบรรเทาความเสี่ยงที่เกี่ยวข้องกับประเด็นปัญหาเหล่านี้ ซึ่งผลจากการตรวจสอบสามารถเพิ่มความมั่นใจให้กับผู้ใช้บริการหรือองค์กรที่ต้องการนำคลาวด์ไปใช้งาน

สำหรับ IaaS คลาวด์หนึ่งในวิธีการบรรเทาประเด็นปัญหาดังกล่าวข้างต้น คือ ระบบบันทึกเหตุการณ์ (Logging System) ซึ่งระบบนี้สามารถช่วยบรรเทาความเสี่ยงที่เกี่ยวข้องกับความกังวลด้านความปลอดภัยของ IaaS คลาวด์ดังที่ได้กล่าวไว้ใน Auxsorn, Wongthai, Porka & Jaiboon (2020); Wiriya, Wongthai & Phoka (2020); W. Wongthai, F. L. Rocha & A. van Moorsel (2013b); Wongthai & Van Moorsel (2016a, 2016b) โดยระบบจะอำนวยความสะดวกในการตรวจสอบเหตุการณ์ที่น่าสงสัย เช่น ใครกำลังเข้าถึงไฟล์ที่มีความสำคัญของผู้ใช้บริการหรือลูกค้าที่อยู่ใน IaaS คลาวด์ และเมื่อระบบสามารถตรวจจับได้ก็จะรายงานว่ามีผู้ใช้งานที่ไม่ได้รับอนุญาตทำอะไรกับไฟล์เป็นต้น และบันทึกข้อมูลที่เกี่ยวข้องกับเหตุการณ์ไว้ในล็อกไฟล์ (Log File) ซึ่งล็อกไฟล์นี้ถือว่าหลักฐานสำคัญในการค้นหาบุคคลมารับผิดชอบต่อการกระทำผิด (Felici & Pearson, 2015) และเป็นกลไกสำคัญสำหรับภาระความรับผิด (Accountability) (Wongthai, 2014) ซึ่งเป็นประโยชน์ทั้งผู้ให้บริการ และผู้ใช้บริการ IaaS คลาวด์ อย่างไรก็ตามงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ได้จัดเตรียมระบบบันทึกเหตุการณ์สำหรับ IaaS คลาวด์ในห้องปฏิบัติการ แทนที่จะเป็น IaaS คลาวด์ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ดังนั้นเนื้อหาในวิทยานิพนธ์เล่มนี้ให้ความสำคัญกับการประยุกต์ใช้ระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สามารถทำงานใน IaaS คลาวด์ที่ใช้งานในการผลิตจริงเพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตามรายงานขององค์กร Cloud Security Alliance

ทั้งนี้ผู้วิจัยได้จำลอง IaaS คลาวด์ที่ใช้งานในการผลิตจริงนี้ ด้วยระบบปฏิบัติการคลาวด์ที่มีชื่อว่า “โอเพนสแต็ก (OpenStack)” ซึ่งระบบปฏิบัติการคลาวด์นี้ถูกนำไปใช้ในการสร้างผลิตภัณฑ์จริงของภาคธุรกิจเพิ่มขึ้นเรื่อย ๆ (Gartner, 2022; OpenStack.org, 2022)

ปัญหาวิจัยหลัก

จากหัวข้อความเป็นมาและความสำคัญของปัญหาและเนื้อหาเอกสารและงานวิจัยที่เกี่ยวข้องในบทที่ 2 สามารถสรุปภาพรวมปัญหาวิจัยหลักของระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ใน IaaS คลาวด์ที่ใช้งานในการผลิตจริงได้ดังนี้

1. ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ได้จัดเตรียมและทดสอบใน IaaS คลาวด์จำลองในห้องปฏิบัติการ โดยใช้เครื่องคอมพิวเตอร์เครื่องเดียว ซึ่งระบบบันทึกเหตุการณ์นี้สามารถทำงานได้ดี อย่างไรก็ตาม IaaS คลาวด์ในสภาพแวดล้อมที่ใช้ในการผลิตจริง เช่น โอเพนสแต็ก (OpenStack) ประกอบด้วยคอมพิวเตอร์จำนวนมาก (OpenStack.org, 2015, 2019) และยังไม่มีการประยุกต์ใช้ระบบบันทึกเหตุการณ์ดังกล่าวให้สามารถทำงานใน IaaS คลาวด์สำหรับสภาพแวดล้อมที่ใช้ในการผลิตจริง เช่น โอเพนสแต็ก ดังนั้นในวิทยานิพนธ์เล่มนี้มีเป้าหมายที่จะประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยก่อนหน้าให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

2. จากปัญหาวิจัยหลักหัวข้อย่อย 1 ยังไม่มีการออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IaaS OpenStack Logging System: IOLS) จากการทบทวนวรรณกรรม ดังนั้นในวิทยานิพนธ์เล่มนี้มีเป้าหมายเพื่อการออกแบบและทำให้เกิดผลของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS)

3. จากปัญหาวิจัยหลักหัวข้อย่อย 1 และหัวข้อย่อย 2 ยังไม่มีผลลัพธ์และการอภิปรายเกี่ยวกับระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ดังนั้นในวิทยานิพนธ์เล่มนี้มีเป้าหมายเพื่อแสดงผลลัพธ์และการอภิปรายเกี่ยวกับระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของ

โอเพนสแต็กใน IaaS คลาวด์ (IOLS) เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

เป้าหมายของงานวิจัย

จากหัวข้อปัญหาวิจัยหลัก ผู้วิจัยมีเป้าหมายเพื่อสร้างต้นแบบระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

วัตถุประสงค์ของการศึกษา

จากหัวข้อปัญหาวิจัยหลัก ได้สรุปภาพรวมปัญหาของระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) เมื่อนำมาประยุกต์ใช้งานใน IaaS คลาวด์ในสภาพแวดล้อมที่ใช้ในการผลิตจริง ซึ่งมีความสำคัญต่อการนำผลงานวิจัยที่มีอยู่ไปสู่การประยุกต์ใช้งานในโลกของความเป็นจริง (Real-world) ดังนั้นผู้วิจัยจึงนำประเด็นปัญหาดังกล่าวมาเป็นเป้าหมายของงานวิจัย ในวิทยานิพนธ์เล่มนี้ ทั้งนี้ระบบบันทึกเหตุการณ์ตามงานวิจัยก่อนหน้าจัดอยู่ในประเภท File-centric log ซึ่งจะเกี่ยวข้องกับการจัดเก็บประวัติของไฟล์ตั้งแต่การสร้าง การอ่าน การแก้ไข และการลบไฟล์ออกจากระบบ ดังนั้น File-centric log สามารถเป็นประวัติของเหตุการณ์ของไฟล์ที่ถูกเข้าถึงจากโปรเซส (Process) ซึ่งโปรเซสนี้จะหมายถึงโปรแกรมที่กำลังถูกเอ็กซิคิวซ์ (Execute) หรือรันบนระบบปฏิบัติการของเครื่องคอมพิวเตอร์ ในทางตรงกันข้ามสิ่งนี้สามารถมองว่าเป็น Process-centric log คือ ประวัติของเหตุการณ์ของโปรเซสที่กำลังเข้าถึงไฟล์ ทั้งนี้ File-centric log และ Process-centric log สามารถสร้างขึ้นได้โดยใช้ Virtual machine introspection (VMI) เจาะเข้าไปในหน่วยความจำหลัก (Main Memory) ของเครื่องเสมือน (Virtual Machine หรือ VM) เพื่อคัดลอกข้อมูลและพฤติกรรมที่เกี่ยวข้องกับโปรเซส

อย่างไรก็ตามในวิทยานิพนธ์เล่มนี้มีเป้าหมายเพื่อสร้างต้นแบบและการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine) ตามเป้าหมายของงานวิจัย โดยให้ความสนใจ Process-centric log มากกว่า File-centric log เนื่องจากการได้มาซึ่งข้อมูลประวัติของเหตุการณ์ของไฟล์ที่ถูกเข้าถึงจากโปรเซส จำเป็นจะต้องทราบข้อมูลเกี่ยวกับโปรเซสเป็นลำดับแรก ดังนั้นหัวข้อนี้จะกล่าวถึงวัตถุประสงค์ของการศึกษาซึ่งจะสอดคล้องกับปัญหาวิจัยหลักและเป้าหมายของงานวิจัย โดยมีรายละเอียดดังต่อไปนี้

1. เพื่อประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

2. เพื่อออภิปรายผลลัพธ์การออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) ว่าทำไมยังคงสามารถทำงานในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ที่สร้างจากโอเพนสแต็ก และระบบบันทึกเหตุการณ์ดังกล่าวสามารถเป็นอีกทางเลือกหนึ่ง สำหรับการบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA

3. เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) สำหรับการบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตามรายงานขององค์กร CSA

4. เพื่อทดสอบและประเมินประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) สำหรับการบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตามรายงานขององค์กร CSA

ขอบเขตของงานวิจัย

เพื่อให้สอดคล้องกับหัวข้อปัญหาวิจัยหลัก หัวข้อเป้าหมายของงานวิจัย และหัวข้อวัตถุประสงค์ของการศึกษา ของงานวิจัยในวิทยานิพนธ์เล่มนี้ ผู้วิจัยได้กำหนดขอบเขตการวิจัยดังต่อไปนี้

1. ขอบเขตด้านเทคโนโลยี อุปกรณ์ และซอฟต์แวร์

1.1 เครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer) จำนวน 2 เครื่องโดยทั้งหมดมีข้อมูลจำเพาะ (Specification) ได้แก่ หน่วยประมวลผลกลางหรือซีพียูแบบ Xeon E5-2689 2.6~3.6 GHz มีจำนวนแกนประมวลผล 8 คอร์ (Core) / 16 เธรด (Thread) หน่วยความจำหลัก (RAM) แบบ ECC 16GB DDR3 1600 MHz หน่วยความจำสำรองขนาด 1 TB และแผ่นวงจรต่อประสานเครือข่าย (Network Interface Card) จำนวน 2 ชุดต่อเครื่อง

1.2 สวิตช์ฮับ (Switch Hub) ซึ่งเป็นอุปกรณ์สำหรับเชื่อมต่อและขยายระบบเครือข่ายแบบ Desktop Ethernet Switch 10/100MB 4 ports จำนวน 2 ชุด

1.3 สาย LAN แบบ CAT5E-Unshielded Twisted Pair (UTP) ซึ่งมีความเร็วสูงสุดในการถ่ายโอนข้อมูลที่ 1 Gbps

1.4 ซอฟต์แวร์ระบบปฏิบัติการ Ubuntu Server 16.04 LTS

1.5 ซอฟต์แวร์ไฮเปอร์ไวเซอร์ (Hypervisor) ซึ่งเป็นซอฟต์แวร์ที่ทำให้เครื่องคอมพิวเตอร์หนึ่งเครื่องสามารถติดตั้งและเรียกใช้งานระบบปฏิบัติการได้มากกว่าหนึ่งระบบในเครื่องเดียวกันพร้อมกันซึ่ง Rocha, Abreu & Correia (2011) ได้กล่าวไว้ โดยในงานวิจัยนี้เลือกใช้ซอฟต์แวร์ เซน (Xen) เวอร์ชัน 4.6.5 สำหรับระบบปฏิบัติการ Ubuntu Server 16.04 LTS

1.6 ซอฟต์แวร์ระบบปฏิบัติการคลาวด์ OpenStack Ocata

1.7 ซอฟต์แวร์ระบบบันทึกเหตุการณ์ (Logging System) ใน IaaS คลาวด์ซึ่งได้รับอนุญาตจาก Auxsorn et al. (2020) และ Wongthai (2014) ให้สามารถแก้ไขและดัดแปลงเพิ่มได้

2. ขอบเขตด้านวิธีการทดลอง

2.1 ศึกษาองค์ประกอบ ขั้นตอนการทำงานของ IaaS คลาวด์และสถาปัตยกรรมตามงานวิจัยของ Wongthai et al. (2013a) หลังจากนั้นทดลองสร้าง IaaS คลาวด์บนเครื่องคอมพิวเตอร์จำนวน 1 เครื่องซึ่งลงระบบปฏิบัติการ Ubuntu Server 16.04 LTS ตามวิธีการขั้นตอน รวมถึงวิธีการแก้ไขปัญหาตามที่ได้ค้นคว้า

2.2 ศึกษาวิธีการ ขั้นตอน รวมถึงวิธีการแก้ไขปัญหาสำหรับการติดตั้งระบบบันทึกเหตุการณ์และการติดตั้งไลบรารี LibVMI ซึ่งเป็นไลบรารีหลักสำหรับการทำงานของระบบบันทึกเหตุการณ์ตามงานวิจัยของ Wongthai et al. (2013a) สำหรับ IaaS คลาวด์ที่ได้จากหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.1

2.3 ทดลองใช้งานระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.2 โดยจำลองเหตุการณ์ให้แฮกเกอร์หรือผู้ไม่ประสงค์ดีเรียกใช้งานแอปพลิเคชัน threat-app ซึ่งเป็นแอปพลิเคชันที่ใช้สำหรับเปิดอ่านไฟล์ ไฟล์ โดยมีรูปแบบคำสั่งสำหรับการเรียกใช้ threat-app ตามภาพ 20 ในขณะเดียวกันระบบบันทึกเหตุการณ์จะทำการตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ตามที่กำหนดไว้

2.4 ศึกษาองค์ประกอบ ขั้นตอนการทำงาน และสถาปัตยกรรมของโอเพนสแต็ก ซึ่งเป็นระบบปฏิบัติการคลาวด์ที่ผู้วิจัยเลือกใช้สำหรับการสร้าง IaaS คลาวด์ โดยพิจารณาจากงานวิจัยของ Sefraoui, Aissaoui & Eleuldj (2012) ซึ่งได้ทำการเปรียบเทียบซอฟต์แวร์ที่ใช้สำหรับการสร้าง IaaS คลาวด์ เช่น Eucalyptus OpenNebula และ OpenStack โดยผลลัพธ์ที่ได้แสดงเห็นว่า OpenStack เป็นตัวเลือกที่เหมาะสมที่สุด เนื่องจากความสามารถในการปรับขนาด (Scalability) ความสามารถที่ใช้แทนกันได้ (Compatible) มีความยืดหยุ่น (Flexible) และเป็นโอเพนซอร์ส (Open Source)

2.5 ศึกษาวิธีการ ขั้นตอน รวมถึงวิธีการแก้ไขปัญหาสำหรับการสร้าง IaaS คลาวด์โดยใช้ซอฟต์แวร์ OpenStack Ocata หลังจากนั้นทดลองสร้าง IaaS คลาวด์ ตามลำดับ

2.6 ทดสอบติดตั้งและปรับปรุงระบบบันทึกเหตุการณ์ตามงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) เพื่อให้สามารถทำงานสำหรับสภาพแวล้อมของโอเพนสแต็กใน IaaS คลาวด์ที่ได้จากหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.5

2.7 ทดลองใช้งานระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.6 โดยจำลองเหตุการณ์ให้แฮกเกอร์หรือผู้ไม่ประสงค์ดีเรียกใช้งานแอปพลิเคชัน threat-app ซึ่งเป็นแอปพลิเคชันที่ใช้สำหรับเปิดอ่านไฟล์ โดยมีรูปแบบคำสั่งสำหรับการเรียกใช้ threat-app ตามภาพ 20 ในขณะเดียวกันระบบบันทึกเหตุการณ์จะทำการตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ตามที่กำหนดไว้ หลังจากนั้นนำผลลัพธ์ที่ได้ไปตีพิมพ์บทความวิชาการ

2.8 ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.6 เพื่อให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของ

โอเพนสแต็กใน IaaS คลาวด์ โดยปรับปรุงระบบบันทึกเหตุการณ์ให้สามารถทำงานแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์ที่อยู่ในเครือข่ายเดียวกันได้ ด้วยการเขียนโปรแกรมซ็อกเก็ต และเพื่อให้รองรับกับสภาพแวดล้อมของโอเพนสแต็กในเรื่องของการปรับขนาด (Scalability) ได้ รวมทั้งเพื่อให้ง่ายต่อการบริหารจัดการสำหรับการเรียกใช้งานระบบบันทึกเหตุการณ์ เมื่อเทียบกับระบบบันทึกเหตุการณ์ตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.6

2.9 ทดลองใช้งานระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.8 โดยจำลองเหตุการณ์ให้แฮกเกอร์หรือผู้ไม่ประสงค์ดีเรียกใช้งานแอปพลิเคชัน threat-app ซึ่งเป็นแอปพลิเคชันที่ใช้สำหรับเปิดอ่านไฟล์ โดยมีรูปแบบคำสั่งสำหรับการเรียกใช้ threat-app ตามภาพ 20 ในขณะเดียวกันระบบบันทึกเหตุการณ์จะทำการตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ตามที่กำหนดไว้ หลังจากนั้นนำผลลัพธ์ที่ได้ไปตีพิมพ์บทความวิชาการ

2.10 ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.8 ให้สามารถทำงานแบบขนาน (Parallel) โดยใช้หลักการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ทั้งนี้เพื่อให้ระบบบันทึกเหตุการณ์สามารถตรวจสอบ (Monitoring) เครื่องเสมือน (Virtual Machine หรือ VM) ได้มากกว่าหนึ่งเครื่องที่อยู่ในคอมพิวเตอร์แม่งาน (Host Computer) หรือโฮสต์ (Host) เดียวกันได้พร้อม ๆ กัน สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

2.11 ทดลองใช้งานระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.10 โดยจำลองเหตุการณ์ให้แฮกเกอร์หรือผู้ไม่ประสงค์ดีเรียกใช้งานแอปพลิเคชัน threat-app ซึ่งเป็นแอปพลิเคชันที่ใช้สำหรับเปิดอ่านไฟล์ โดยมีรูปแบบคำสั่งสำหรับการเรียกใช้ threat-app ตามภาพ 20 ในขณะเดียวกันระบบบันทึกเหตุการณ์จะทำการตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ตามที่กำหนดไว้

2.12 ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้ติดตั้งตามหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.10 โดยปรุงปรับอัลกอริทึมสำหรับการตรวจสอบของระบบบันทึกเหตุการณ์เดิม ทั้งนี้ได้มีการประยุกต์ใช้ตารางแฮช (Hash Table) และการจัดเรียงลำดับการทำงานของชุดคำสั่งใหม่ เพื่อเพิ่มประสิทธิภาพสำหรับการตรวจสอบของระบบบันทึกเหตุการณ์

2.13 ทำการวัดค่าความถูกต้อง (Accuracy) และทดสอบประสิทธิภาพจากค่าความถูกต้อง (Accuracy) และระยะเวลาของอัลกอริทึมที่ใช้สำหรับการตรวจสอบหน่วยความจำหลักของเครื่องเสมือน (VM) ของระบบบันทึกเหตุการณ์จากหัวข้อขอบเขตด้านวิธีการทดลอง

หัวข้อย่อย 2.6 หัวข้อย่อย 2.8 หัวข้อย่อย 2.10 และหัวข้อย่อย 2.12 ตามลำดับ สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ประโยชน์ที่คาดว่าจะได้รับ

จากหัวข้อวัตถุประสงค์ของการศึกษาที่ต้องการประยุกต์ใช้ระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ การอธิปรายผลลัพธ์การออกแบบ (Design) และการทำให้เกิดผล (Implementation) รวมถึงการทดสอบและประเมินประสิทธิภาพของระบบบันทึกเหตุการณ์ดังกล่าว เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine) ซึ่งประโยชน์ของงานวิจัยนี้จึงมีความสำคัญต่อการพิจารณานำระบบบันทึกเหตุการณ์ไปสู่การประยุกต์ใช้งานในโลกของความเป็นจริง (Real-world) สำหรับผู้ให้บริการ (Provider) คลาวด์ เพื่อสร้างความน่าเชื่อถือต่อผู้ใช้บริการ (Consumer) IaaS คลาวด์ โดยมีรายละเอียดดังต่อไปนี้

1. ได้ต้นแบบของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IaaS OpenStack Logging System: IOLS)

2. ได้ต้นแบบสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็ก และนิยามคำศัพท์ของส่วนประกอบในภาพสถาปัตยกรรมเพื่อนำไปใช้เป็นพื้นฐานองค์ความรู้สำหรับใช้อ้างอิงในงานวิจัยที่เกี่ยวข้องกับ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็กหรือคลาวด์ประเภทอื่น ๆ ในสภาพแวดล้อมเดียวกัน

3. ได้ทราบข้อกำหนดเบื้องต้นของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) จากขั้นตอนการทดสอบและประเมินประสิทธิภาพจากค่าความถูกต้อง (Accuracy) ของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หากมีการประยุกต์ใช้งานในโลกของความเป็นจริง (Real-world) ซึ่งอาจจะเป็นประโยชน์ต่อผู้ให้บริการคลาวด์

4. ผู้ให้บริการคลาวด์อาจจะนำข้อมูลของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) จากงานวิจัยในวิทยานิพนธ์เล่นนี้ ประกอบการพิจารณาสร้างระบบอื่น ๆ ที่เกี่ยวข้องกับความปลอดภัยของ IaaS คลาวด์ เพื่อสร้างความน่าเชื่อถือกับบริการต่าง ๆ ที่ผู้ให้บริการคลาวด์เสนอต่อผู้ใช้บริการคลาวด์ขององค์กรได้

5. ผลสำเร็จจากการประยุกต์ใช้ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) สามารถแสดงให้เห็นถึงระบบบันทึกเหตุการณ์ในวิทยานิพนธ์เล่มนี้สามารถประยุกต์ใช้งานได้จริงสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์และยังสามารถขยายไปสู่คลาวด์ประเภทอื่นได้ เช่น การให้บริการแพลตฟอร์ม (Platform as a Service: PaaS) และการให้บริการซอฟต์แวร์ (Software as a Service: SaaS)

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

สำหรับเนื้อหาในบทที่ 2 ผู้วิจัยจะนำเสนอข้อมูลที่เกี่ยวข้องกับเอกสารและงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์เล่มนี้ในเรื่อง การปรับปรุประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ ซึ่งจะแสดงให้เห็นถึงแนวทางการประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) รวมถึงการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ดังกล่าวให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) เช่น โอเพนสแต็ก (OpenStack) สืบเนื่องจากประเด็นเกี่ยวกับความกังวลด้านความปลอดภัยของ IaaS คลาวด์ เช่น เรื่องของความลับ (Confidentiality) ความถูกต้องสมบูรณ์ (Integrity) และความพร้อมใช้งาน (Availability) ตัวอย่างเช่น ไฟล์ที่มีความสำคัญของผู้ใช้บริการ (Critical File) ซึ่งเป็นปัญหาสำคัญที่ขัดขวางการนำ IaaS คลาวด์ไปใช้งาน โดยหนึ่งในวิธีการบรรเทาประเด็นปัญหาดังกล่าวข้างต้น คือ ระบบบันทึกเหตุการณ์ (Logging System) ซึ่งเป็นกลไกสำคัญสำหรับภาระความรับผิด (Accountability) ที่ใช้สำหรับสร้างหลักฐานเพื่อยืนยันว่าบุคคลใดเข้าถึงไฟล์ที่มีความสำคัญของผู้ใช้บริการโดยไม่ได้รับอนุญาต ซึ่งอาจจะเป็นประโยชน์ทั้งผู้ให้บริการ (Provider) และผู้ใช้บริการ (Customer) คลาวด์ที่อาจจะนำไปใช้เป็นหลักฐานในการค้นหาบุคคลมารับผิดชอบต่อการกระทำผิด สำหรับเนื้อหาในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจะมุ่งเน้นสิ่งที่เกี่ยวข้องกับ IaaS คลาวด์สำหรับสภาพแวดล้อมที่ใช้งานในการผลิตจริง โดยแบ่งเป็นหัวข้อย่อยดังต่อไปนี้

1. การประมวลผลแบบกลุ่มเมฆ
2. สถาปัตยกรรมและปัญหาภัยคุกคามของการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์
3. สถาปัตยกรรมของโอเพนสแต็ก
4. สถาปัตยกรรมของระบบบันทึกเหตุการณ์ในการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ
5. การเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์
6. การประมวลผลแบบขนาน
7. การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด
8. การทดสอบการประมวลผลแบบกลุ่มเมฆ

การประมวลผลแบบกลุ่มเมฆ

การประมวลผลแบบกลุ่มเมฆ (Cloud Computing) หรือคลาวด์ (Cloud) ถือว่าเป็นรูปแบบการดำเนินงานและชุดของเทคโนโลยีสำหรับการจัดการการใช้ทรัพยากรคอมพิวเตอร์ที่ใช้ร่วมกัน ทำให้เกิดความคล่องตัว การปรับขนาด ความพร้อมใช้งาน และยังถือเป็นโอกาสสำหรับการลดต้นทุนผ่านการประมวลผลที่เหมาะสมและมีประสิทธิภาพ ปัจจุบันคลาวด์ยังได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ ดังผลการสำรวจขององค์กรการ์เนอร์ (Gartner, 2022) ซึ่งคาดการณ์มูลค่าใช้จ่ายของผู้ใช้บริการบนคลาวด์สาธารณะทั่วโลกสูงถึง 591.8 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2566 คิดเป็นร้อยละ 20.7 เพิ่มขึ้นจาก 490.3 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2565 และในปี พ.ศ. 2565 มีการเติมโตคิดเป็นร้อยละ 18.8 เพิ่มขึ้นจาก 412.6 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2564 การที่ คลาวด์สาธารณะถูกนำมาใช้ในองค์กรเพิ่มขึ้นเนื่องจากความสามารถในการปรับขนาด (Scalability) ได้และมีความยืดหยุ่น (Elasticity) ทั้งนี้ IaaS คลาวด์มีอัตราการเติมโตสูงสุด ดังนั้น IaaS คลาวด์จึงมีโอกาสประสบปัญหาเรื่องความปลอดภัยมากกว่าคลาวด์ประเภทอื่น ๆ อีกทั้ง IaaS คลาวด์ยังเป็นโครงสร้างพื้นฐานสำหรับการสร้างคลาวด์ประเภทอื่น เช่น การให้บริการแพลตฟอร์ม (Platform as a Service: PaaS) และการให้บริการซอฟต์แวร์ (Software as a Service: SaaS) หากประสบปัญหาที่เกี่ยวข้องกับความปลอดภัยย่อมส่งผลกระทบต่อเนื่องไปยังคลาวด์ประเภทอื่นอย่างหลีกเลี่ยงไม่ได้

1. คำจำกัดความของการประมวลผลแบบกลุ่มเมฆ

สถาบันมาตรฐานและเทคโนโลยีแห่งชาติ (Nation Institute of Standard and Technology: NIST) ซึ่งทำหน้าที่กำหนดมาตรฐาน คำนิยาม และให้ข้อกำหนดต่าง ๆ กับหน่วยงานทั้งภาครัฐและเอกชนในประเทศสหรัฐอเมริกา โดยในปี ค.ศ. 2011 มีการเผยแพร่เอกสารชื่อเรื่อง The NIST Definition of Cloud Computing ซึ่งมีเนื้อหาเกี่ยวกับการนิยามศัพท์ต่าง ๆ ที่เกี่ยวข้องกับคลาวด์เพื่อใช้เป็นมาตรฐานสำหรับนักวิจัย องค์กรต่าง ๆ ในการอ้างอิง (Mell & Grance, 2011) สำหรับคำนิยามของคลาวด์ตามที่ได้กล่าวไว้ใน Mell & Grance (2011) มีข้อความดังต่อไปนี้

“A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“

ผู้วิจัยขอแปลความหมายของข้อความดังกล่าวเป็นภาษาไทยพร้อมทั้งเรียบเรียงใหม่ได้ดังข้อความต่อไปนี้

“แบบจำลองสำหรับการใช้ทรัพยากรคอมพิวเตอร์ เช่น เครือข่าย (Network) เครื่องเซิร์ฟเวอร์ (Server) หน่วยเก็บข้อมูล (Storage) โปรแกรมประยุกต์หรือแอปพลิเคชัน (Application) และบริการอื่น ๆ ซึ่งทรัพยากรเหล่านี้สามารถเข้าถึงได้อย่างสะดวกจากอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ โดยเชื่อมต่อผ่านเครือข่ายได้จากทุกที่ การขอใช้หรือยกเลิกการใช้ทรัพยากรเหล่านี้สามารถทำได้อย่างรวดเร็วและปรับเปลี่ยนไปตามความต้องการ โดยใช้ความพยายามในการจัดการหรือติดต่อกับผู้ให้บริการ (Provider) น้อยที่สุด”

2. ประเภทของการประมวลผลแบบกลุ่มเมฆแบ่งตามลักษณะขอบเขตการให้บริการ

Mell & Grance (2011) ได้แบ่งคลาวด์ตามลักษณะขอบเขตของการให้บริการ (Service Model) ไว้ 3 ประเภท ได้แก่ การให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ (Infrastructure as a Service: IaaS) การให้บริการแพลตฟอร์ม (Platform as a Service: PaaS) และการให้บริการซอฟต์แวร์ (Software as a Service: SaaS) โดยที่แบบจำลองการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์หรือ IaaS จะเกี่ยวข้องกับการจัดเตรียมจำนวนหน่วยประมวลผลกลาง หน่วยเก็บข้อมูล เครือข่าย และทรัพยากรคอมพิวเตอร์พื้นฐานอื่น ๆ ซึ่งผู้ใช้บริการสามารถปรับใช้ตามความต้องการ รวมถึงควบคุมระบบปฏิบัติการ การจัดเก็บ การปรับใช้แอปพลิเคชัน และส่วนประกอบของเครือข่ายบางส่วน เช่น ไฟร์วอลล์ (Firewall) ของเครื่องคอมพิวเตอร์แม่งาน (Host Computer) หรือโฮสต์ สำหรับแบบจำลองการให้บริการแพลตฟอร์มหรือ PaaS จะเกี่ยวข้องกับการปรับใช้โครงสร้างพื้นฐานระบบคลาวด์โดยที่ผู้ใช้บริการสามารถนำแอปพลิเคชันที่สร้างขึ้นเองหรือได้รับมาติดตั้งบนโครงสร้างพื้นฐานระบบคลาวด์ได้ ซึ่งแอปพลิเคชันเหล่านี้ถูกสร้างโดยใช้ภาษาโปรแกรม ไลบารี บริการอื่น ๆ ที่รองรับโดยผู้ให้บริการคลาวด์ และสุดท้ายแบบจำลองการให้บริการซอฟต์แวร์หรือ SaaS จะเกี่ยวข้องกับการใช้แอปพลิเคชันของผู้ให้บริการที่ทำงานอยู่บนโครงสร้างพื้นฐานระบบคลาวด์ ซึ่งแอปพลิเคชันสามารถเข้าถึงได้จากอุปกรณ์ต่าง ๆ ผ่านเครือข่าย เช่น เว็บเบราว์เซอร์

3. ประเภทของการประมวลผลแบบกลุ่มเมฆแบ่งตามลักษณะขอบเขตการจัดการ

นอกจากนี้ Mell & Grance (2011) ยังได้แบ่งคลาวด์ตามลักษณะขอบเขตการจัดการ (Deployment Model) ไว้ 4 ประเภท ได้แก่ คลาวด์ภายในองค์กร (Private Cloud) คลาวด์สาธารณะ (Public Cloud) คลาวด์สำหรับผู้ใช้เฉพาะกลุ่ม (Community Cloud) และ

คลาวด์แบบผสมผสาน (Hybrid Cloud) โดยที่แบบจำลองคลาวด์ภายในองค์กรจะเกี่ยวข้องกับการจัดเตรียมโครงสร้างพื้นฐานระบบคลาวด์ไว้สำหรับการใช้งานภายในองค์กรเดียวที่ประกอบด้วยผู้ใช้บริการหลายคนที่อยู่ในองค์กรเดียวกัน เช่น ฝ่ายบัญชี ฝ่ายบุคคล เป็นต้น คลาวด์ประเภทนี้อาจถูกเป็นเจ้าของ ถูกจัดการ และถูกดำเนินการโดยองค์กร บุคคลที่สามหรือบางส่วนรวมกัน ซึ่งอาจจะอยู่ในสถานที่ใดที่หนึ่งภายในองค์กรหรือนอกองค์กร แต่อยู่ภายใต้การควบคุมขององค์กรหนึ่ง สำหรับแบบจำลองคลาวด์สาธารณะจะเกี่ยวข้องกับการจัดเตรียมโครงสร้างพื้นฐานระบบคลาวด์ไว้สำหรับการใช้งานแบบเปิดโดยให้ผู้ใช้บริการทั่วไปใช้งาน คลาวด์ประเภทนี้อาจถูกเป็นเจ้าของ ถูกจัดดการ และถูกดำเนินการโดยองค์กรทางธุรกิจ องค์กรทางการศึกษา องค์กรของรัฐ หรือผสมผสานทั้งสามองค์กรผ่านความร่วมมือกันระหว่างองค์กร สำหรับแบบจำลองคลาวด์สำหรับผู้ใช้เฉพาะกลุ่มจะเกี่ยวข้องกับการจัดเตรียมโครงสร้างพื้นฐานระบบคลาวด์ไว้สำหรับกลุ่มผู้ใช้บริการที่มีความสนใจร่วมกัน โดยผู้ใช้บริการนี้อาจจะมาจากองค์กรที่มีปัญหาหรือความสนใจที่คล้ายกันและสามารถเข้าถึงบริการคลาวด์ประเภทนี้ได้เฉพาะกลุ่มผู้ใช้บริการที่ได้รับอนุญาตเท่านั้น คลาวด์ประเภทนี้อาจถูกเป็นเจ้าของ ถูกจัดการ และถูกดำเนินการโดยองค์กรที่เป็นสมาชิก บุคคลที่สาม หรือผสมผสานองค์กรเข้าร่วมกัน ซึ่งอาจจะสร้างคลาวด์นี้ไว้ภายในหรือภายนอกองค์กรที่เป็นเจ้าของคลาวด์นี้ก็ได้ และสุดท้ายแบบจำลองคลาวด์แบบผสมผสานจะเกี่ยวข้องกับการจัดเตรียมโครงสร้างพื้นฐานระบบคลาวด์ที่แตกต่างกันตั้งแต่สองประเภทขึ้นไป เช่น การผสมผสานระหว่างคลาวด์ภายในองค์และคลาวด์สาธารณะ ทั้งนี้คลาวด์ทั้งสองประเภทยังคงมีเอนทิตีที่ไม่ซ้ำกัน แต่ถูกรวมเข้าด้วยกันด้วยเทคโนโลยีที่เป็นมาตรฐานหรือสร้างขึ้นเฉพาะเพื่อช่วยให้สามารถย้ายข้อมูลหรือแอปพลิเคชันได้

เนื้อหาทั้งหมดในหัวข้อการประมวลผลแบบกลุ่มเมฆ แบ่งเป็นหัวข้อย่อย ได้แก่ หัวข้อคำจำกัดความของการประมวลผลแบบกลุ่มเมฆ หัวข้อประเภทของการประมวลผลแบบกลุ่มเมฆแบ่งตามลักษณะขอบเขตการให้บริการ และหัวข้อประเภทของการประมวลผลแบบกลุ่มเมฆแบ่งตามลักษณะขอบเขตการจัดการ โดยผู้วิจัยขอสรุปเนื้อหาทั้งหมดดังต่อไปนี้

คลาวด์ถือว่าเป็นรูปแบบการดำเนินงานและชุดของเทคโนโลยีสำหรับการจัดการการใช้ทรัพยากรคอมพิวเตอร์ที่ใช้ร่วมกัน ทำให้เกิดความคล่องตัว การปรับขนาด ความพร้อมใช้งาน และยังถือเป็นโอกาสสำหรับการลดต้นทุนผ่านการประมวลผลที่เหมาะสมและมีประสิทธิภาพ ซึ่งปัจจุบันคลาวด์ยังได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ ดังผลการสำรวจขององค์กรการ์เนอร์ (Gartner, 2022) การที่คลาวด์สาธารณะถูกนำมาใช้ในองค์กรเพิ่มขึ้นเนื่องจากความสามารถในการปรับขนาดได้และมีความยืดหยุ่น ทั้งนี้ IaaS คลาวด์มีอัตราการเติมโตสูงสุด ดังนั้น IaaS คลาวด์จึงมี

โอกาสประสบปัญหาเรื่องความปลอดภัยมากกว่าคลาวด์ประเภทอื่น ๆ อีกทั้ง IaaS คลาวด์ยังเป็นโครงสร้างพื้นฐานสำหรับการสร้างคลาวด์ประเภทอื่นอีกด้วย หากประสบปัญหาที่เกี่ยวข้องกับความปลอดภัยย่อมส่งผลกระทบต่อเนื่องไปยังคลาวด์ประเภทอื่นอย่างหลีกเลี่ยงไม่ได้

ทั้งนี้ผู้วิจัยได้แปลเรียบเรียงความหมายคำจำกัดความของคลาวด์ใหม่จากคำจำกัดความของคลาวด์ที่ให้ไว้โดย Mell & Grance (2011) ดังข้อความต่อไปนี้ “แบบจำลองสำหรับการใช้ทรัพยากรคอมพิวเตอร์ เช่น เครือข่าย (Network) เครื่องเซิร์ฟเวอร์ (Server) หน่วยเก็บข้อมูล (Storage) โปรแกรมประยุกต์หรือแอปพลิเคชัน (Application) และบริการอื่น ๆ ซึ่งทรัพยากรเหล่านี้สามารถเข้าถึงได้อย่างสะดวกจากอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ โดยเชื่อมต่อผ่านเครือข่ายได้จากทุกที่ การขอใช้หรือยกเลิกการใช้ทรัพยากรเหล่านี้สามารถทำได้อย่างรวดเร็วและปรับเปลี่ยนไปตามความต้องการ โดยใช้ความพยายามในการจัดการหรือติดต่อกับผู้ให้บริการ (Provider) น้อยที่สุด” และ Mell & Grance (2011) ยังได้แบ่งคลาวด์ตามลักษณะขอบเขตของการให้บริการไว้ 3 ประเภท ได้แก่ การให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ (Infrastructure as a Service: IaaS) การให้บริการแพลตฟอร์ม (Platform as a Service: PaaS) และการให้บริการซอฟต์แวร์ (Software as a Service: SaaS) และแบ่งคลาวด์ตามลักษณะขอบเขตของการจัดการไว้ 4 ประเภท ได้แก่ คลาวด์ภายในองค์กร (Private Cloud) คลาวด์สาธารณะ (Public Cloud) คลาวด์สำหรับผู้ใช้เฉพาะกลุ่ม (Community Cloud) และคลาวด์แบบผสมผสาน (Hybrid Cloud)

ดังนั้นเนื้อหาในวิทยานิพนธ์เล่นนี้จะเน้นที่คลาวด์สาธารณะประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์หรือ IaaS คลาวด์ เนื่องจากคลาวด์สาธารณะถูกนำมาใช้ในองค์กรเพิ่มขึ้นเรื่อย ๆ ด้วยเหตุผลที่ว่าสามารถปรับขนาดได้และมีความยืดหยุ่น และ IaaS คลาวด์ยังมีอัตราการเติบโตสูงสุดซึ่ง Gartner (2022) ได้กล่าวไว้ และในหัวข้อความเป็นมาและความสำคัญของปัญหา ได้กล่าวถึงประเด็นปัญหาเกี่ยวกับความกังวลด้านความปลอดภัยของ IaaS คลาวด์ ซึ่งเป็นปัญหาสำคัญที่ขัดขวางการนำคลาวด์ไปใช้งาน

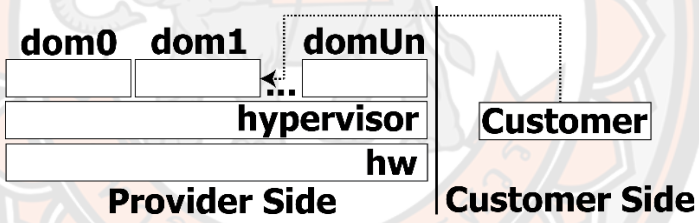
สถาปัตยกรรมและปัญหาภัยคุกคามของการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์

จากหัวข้อการประมวลผลแบบกลุ่มเมฆ ได้กล่าวถึงคลาวด์เพื่อให้ผู้อ่านได้ทราบถึงคำจำกัดความของคลาวด์ การแบ่งประเภทของคลาวด์โดยการแบ่งตามลักษณะขอบเขตของการให้บริการและลักษณะขอบเขตการจัดการ รวมทั้งทราบถึงลักษณะของคลาวด์แต่ละประเภท

สำหรับเนื้อหาในหัวข้อนี้ ผู้วิจัยจะกล่าวถึงสถาปัตยกรรมของ IaaS คลาวด์และปัญหาภัยคุกคามของ IaaS คลาวด์ โดยแบ่งเป็นหัวข้อย่อยดังต่อไปนี้

1. สถาปัตยกรรมของ IaaS คลาวด์
2. ปัญหาภัยคุกคามของ IaaS คลาวด์
3. สถาปัตยกรรมของ IaaS คลาวด์

ภาพ 1 ที่เสนอโดย Wongthai (2014) แสดงสถาปัตยกรรมของ IaaS คลาวด์ สำหรับคำอธิบายจะพิจารณาจากมุมมองของผู้ให้บริการ (Provider) ซึ่งประกอบด้วย 2 ฝ่าย คือ ฝ่ายผู้ให้บริการ (Provider Side) หรือเรียกว่า ผู้ให้บริการ (Provider) หมายถึง องค์กรที่เป็นเจ้าของคลาวด์ ตัวอย่างเช่น บริษัท Rackspace Cloud Service Amazon Elastic Cloud Microsoft Google ที่เสนอบริการต่าง ๆ ให้ผู้ใช้บริการเช่า เช่น เครื่องเสมือน (Virtual Machine หรือ VM) และฝ่ายผู้ใช้บริการ (Customer Side) หรือเรียกว่า ผู้ใช้บริการ (Customer) หมายถึง บุคคลหรือองค์กรที่เช่าบริการของ IaaS คลาวด์จากผู้ให้บริการ ซึ่งผู้ใช้บริการสามารถเข้าถึงบริการของ IaaS คลาวด์ผ่านระบบอินเทอร์เน็ต



ภาพ 1 สถาปัตยกรรมของ IaaS คลาวด์

ที่มา: Wongthai (2014)

ส่วนประกอบหลักของสถาปัตยกรรมของ IaaS คลาวด์ ได้แก่ hw hypervisor dom0 และ domU โดยที่ hw (ย่อมาจาก hardware) คือ เครื่องคอมพิวเตอร์กายภาพสำหรับเป็นโฮสต์ของเครื่องเสมือน หรือ VM ที่ให้เช่าทั้งหมด ดูกล่องด้านล่างฝ่ายผู้ให้บริการ (Provider Side) ของภาพ 1 ซึ่ง hw นี้จะถูกเป็นเจ้าของ ถูกจัดการ และถูกบำรุงรักษาโดยผู้ให้บริการ สำหรับ hypervisor คือ ซอฟต์แวร์ที่ทำให้เครื่องคอมพิวเตอร์กายภาพเครื่องหนึ่งสามารถรันเครื่องเสมือนได้หลายเครื่องพร้อมกัน ดูกล่องตรงกลางฝั่งผู้ให้บริการของภาพ 1(Aalam, Kumar, & Gour, 2021; Rocha et al., 2011) ซึ่งซอฟต์แวร์นี้มีทั้งรูปแบบลิขสิทธิ์และโอเพนซอร์ส ดูกล่องมุมซ้ายบนฝ่ายผู้ให้บริการของภาพ 1 คือ dom0 (อ่านว่าดอมซีโร่ ย่อมาจาก domain zero ซึ่งเป็นคำศัพท์

เฉพาะสำหรับซอฟต์แวร์ Xen โดยซอฟต์แวร์นี้เป็น hypervisor ตัวหนึ่งที่เป็นรูปแบบโอเพนซอร์ส) หรือโดเมน 0 (Domain 0) และเป็นผู้จัดการเครื่องเสมือนหรือ domU ทั้งหมด ซึ่ง dom0 จะเป็นโดเมนที่มีสิทธิพิเศษของซอฟต์แวร์ Xen โดยมีความสามารถเข้าถึง hw ได้โดยตรงและสามารถจัดการ domU ทั้งหมดได้ และซอฟต์แวร์ Xen ดังกล่าวจะเริ่มต้นทำงานเมื่อบูตระบบปฏิบัติการของเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของเครื่องเสมือน และลำดับสุดท้ายคือ domU (อ่านว่าดอมยู ย่อมาจาก domain user ซึ่งเป็นคำศัพท์เฉพาะสำหรับซอฟต์แวร์ Xen) หรือโดเมนผู้ใช้ (Domain User) ซึ่งเป็นเครื่องเสมือน (Virtual Machine หรือ VM) ดูที่กล่องที่มีป้าย dom1 และ domUn ในภาพ 1 เป็นโดเมนทั่วไปสำหรับให้ผู้ใช้บริการเช่าและ domU จะทำงานบน hypervisor และไม่สามารถเข้าถึง hw ได้โดยตรง

ภายใต้ขอบเขตการออกแบบสถาปัตยกรรมของ IaaS คลาวด์ตามงานวิจัยของ Wongthai (2014) ได้ทดลองในสภาพแวดล้อมคลาวด์จำลองบนเครื่องคอมพิวเตอร์กายภาพเครื่องเดียว ซึ่งสถาปัตยกรรมของ IaaS คลาวด์ดังกล่าวถูกนำไปติดตั้งระบบบันทึกเหตุการณ์ เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ซึ่งระบบบันทึกเหตุการณ์นี้สามารถทำงานได้ดี อย่างไรก็ตามยังไม่มีการประยุกต์ใช้ระบบบันทึกเหตุการณ์นี้ในสถาปัตยกรรมของ IaaS คลาวด์ที่ใช้งานในการผลิตจริง เช่น โอเพนสแต็ก โดยผู้วิจัยเชื่อว่าสถาปัตยกรรมของ IaaS คลาวด์ในสภาพแวดล้อมคลาวด์จำลองบนเครื่องคอมพิวเตอร์กายภาพเครื่องเดียวมีความแตกต่างจากสถาปัตยกรรมของ IaaS คลาวด์ที่ใช้งานในการผลิตจริง เช่น โอเพนสแต็ก ดังนั้นระบบบันทึกเหตุการณ์นี้อาจจะมีผลลัพธ์ในการทำงานที่แตกต่างกัน ประกอบกับผู้วิจัยต้องการนำองค์ความรู้จากงานวิจัยไปสู่การใช้งานจริง

2. ปัญหาภัยคุกคามของ IaaS คลาวด์

จากผลการสำรวจขององค์กรการ์เนอร์ (Gartner, 2022) ซึ่งคาดการณ์มูลค่าใช้จ่ายของผู้ใช้บริการบนคลาวด์สาธารณะทั่วโลกสูงถึง 591.8 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2566 คิดเป็นร้อยละ 20.7 เพิ่มขึ้นจาก 490.3 พันล้านดอลลาร์สหรัฐในปี พ.ศ. 2565 เนื่องจากคลาวด์สาธารณะมีความสามารถในการปรับขนาดได้และมีความยืดหยุ่น และ IaaS คลาวด์มีอัตราการเติมโตสูงสุด ดังนั้น IaaS คลาวด์จึงมีโอกาสประสบปัญหาเรื่องความปลอดภัยมากกว่าคลาวด์ประเภทอื่น ๆ อีกทั้ง IaaS คลาวด์ยังเป็นโครงสร้างพื้นฐานสำหรับการสร้างคลาวด์ประเภทอื่น เช่น PaaS และ SaaS คลาวด์ หากประสบปัญหาเรื่องความปลอดภัยย่อมส่งผลกระทบต่อเนื่องไปยังคลาวด์ประเภทอื่นอย่างหลีกเลี่ยงไม่ได้ สำหรับเนื้อหาในหัวข้อนี้ผู้วิจัยจะกล่าวถึงเรื่องความปลอดภัยของระบบคอมพิวเตอร์ และภัยคุกคามของ IaaS คลาวด์ที่ระบุโดยองค์กร CSA

1. ความปลอดภัยของระบบคอมพิวเตอร์

ความปลอดภัย (Security) คือ ปัญหาของคลาวด์ซึ่ง Subashini & Kavitha (2011) ได้กล่าวไว้และจนถึงปัจจุบันปัญหาความปลอดภัยนี้ยังคงอยู่ ซึ่งเห็นได้จากรายงานขององค์กร CSA ที่ทำการวิจัย รวบรวม และระบุปัญหาเกี่ยวข้องกับภัยคุกคามต่อคลาวด์มากว่า 14 ปี มีการเผยแพร่รายงานประมาณ 23 ฉบับ และได้รับการแปลเป็นภาษาอื่นอีกห้าภาษา ตัวอย่างเช่น รายงานเรื่อง “Top Threats to Cloud Computing, Version 1.0” ในปี ค.ศ. 2010 ถึงรายงานเรื่อง “Top Threats to Cloud Computing Pandemic Eleven” ในปี ค.ศ. 2022 (CSA, 2010, 2022) และ Melvin et al. (2022) ยังได้อธิบายอีกว่าการตรวจสอบการบุกรุกบนคลาวด์มีความสำคัญอย่างยิ่งสำหรับทรัพยากรของคลาวด์ เนื่องจากปริมาณการโจมตีและมัลแวร์ที่มีจำนวนเพิ่มขึ้น อย่างไรก็ตามเมื่อพิจารณาลักษณะของคลาวด์แล้วถือว่าเป็นระบบคอมพิวเตอร์รูปแบบหนึ่ง โดย Avizienis, Laprie, Randell & Landwehr (2004) ได้อธิบายเกี่ยวกับความปลอดภัย (Security) หมายถึง ความปลอดภัยของระบบคอมพิวเตอร์และระบบการสื่อสาร ซึ่งในวิทยานิพนธ์เล่มนี้ผู้วิจัยจะพิจารณาเฉพาะความปลอดภัยของระบบคอมพิวเตอร์เท่านั้น โดยไม่นับรวมความปลอดภัยของระบบการสื่อสาร ทั้งนี้ความปลอดภัยของระบบคอมพิวเตอร์ประกอบด้วย 3 องค์ประกอบ ได้แก่ ความลับ (Confidentiality) ความถูกต้องสมบูรณ์ (Integrity) และความพร้อมใช้งาน (Availability) เพื่อให้ง่ายต่อการอธิบายผู้วิจัยขอยกตัวอย่างสถานการณ์ของลูกค้าของธนาคาร ซึ่งลูกค้าแต่ละคนจะมีข้อมูลประวัติธุรกรรมทางการเงินอยู่ในระบบคอมพิวเตอร์ของธนาคารที่ลูกค้าใช้บริการอยู่ โดยแต่ละองค์ประกอบสามารถอธิบายได้ดังต่อนี้

1. **ความลับ** (Confidentiality) หมายถึง ความลับของลูกค้าของธนาคารต้องไม่ถูกเปิดเผย ตัวอย่างเช่น ข้อมูลประวัติการทำธุรกรรมของลูกค้าของธนาคารแต่ละคนต้องไม่ถูกเปิดเผยโดยวิธีการที่ไม่ถูกต้อง

2. **ความถูกต้องสมบูรณ์** (Integrity) หมายถึง ข้อมูลประวัติการทำธุรกรรมของลูกค้าแต่ละคนต้องถูกต้องสมบูรณ์เสมอ ตัวอย่างเช่น ข้อมูลประวัติการฝาก-ถอนเงินของลูกค้าของธนาคารต้องไม่มีการเปลี่ยนแปลงที่เกิดจากการทำงานที่ผิดพลาดของระบบคอมพิวเตอร์ของธนาคาร

3. **ความพร้อมใช้งาน** (Availability) หมายถึง ข้อมูลเกี่ยวกับบัญชีเงินฝากของลูกค้าของธนาคารต้องพร้อมใช้งานตลอดเวลา ตัวอย่างเช่น เมื่อลูกค้าของธนาคารต้องการทำธุรกรรมการฝาก-ถอนเงิน ข้อมูลเกี่ยวกับบัญชีเงินฝากของลูกค้าของธนาคารต้องพร้อมที่จะให้ดำเนินการทันที

2. ภัยคุกคามใน IaaS คลาวด์ที่ระบุโดยองค์กร CSA

จากที่องค์กร Cloud Security Alliance หรือ CSA ที่ทำการวิจัย รวบรวม และระบุปัญหาเกี่ยวข้องกับภัยคุกคามต่อคลาวด์มากว่า 14 ปี มีการเผยแพร่รายงานประมาณ 23 ฉบับ และได้รับการแปลเป็นภาษาอื่นอีกห้าภาษา และมีรายงานเกี่ยวกับคำแนะนำด้านความปลอดภัยบนคลาวด์มาแล้วกว่า 4 ฉบับ ตัวอย่างเช่น รายงานเรื่อง “Security Guidance for Critical Areas of Focus in Cloud Computing v4.0” ในปี ค.ศ. 2017 (CSA, 2017) และรายงานเกี่ยวกับภัยคุกคามต่อคลาวด์ขององค์กร CSA เหล่านี้ได้ถูกนำไปใช้ในการอ้างอิงในบทความวิชาการที่เกี่ยวข้องกับคลาวด์อย่างต่อเนื่อง ดังนั้นในวิทยานิพนธ์เล่นนี้จะใช้รายงานเกี่ยวกับภัยคุกคามต่อคลาวด์ขององค์กร CSA ปี ค.ศ. 2019 ชื่อ “Top Threats to Cloud Computing: Egregious Eleven” ซึ่งได้แบ่งภัยคุกคามต่อคลาวด์ทั้งหมด 11 ประเภท โดยผู้วิจัยขอแปลและเรียบเรียงความหมายของข้อความดังกล่าวเป็นภาษาไทยใหม่ ดังต่อไปนี้

1. **การละเมิดข้อมูล** (Data Breaches) หมายถึง เหตุการณ์ด้านความปลอดภัยในโลกไซเบอร์ที่ข้อมูลละเอียดอ่อนหรือข้อมูลที่มีความสำคัญ ซึ่งข้อมูลเหล่านี้ที่ถูกปกป้องหรือเป็นความลับ ถูกเผยแพร่ ถูกขโมยหรือถูกใช้โดยบุคคลที่ไม่ได้รับอนุญาต

2. **การกำหนดค่าผิดพลาดและการควบคุมการเปลี่ยนแปลงไม่เพียงพอ** (Misconfiguration and Inadequate Change Control) หมายถึง การกำหนดค่าผิดพลาดเกิดขึ้นเมื่อทรัพยากรคอมพิวเตอร์มีการตั้งค่าไม่ถูกต้อง ทำให้ทรัพยากรคอมพิวเตอร์นี้เสี่ยงต่อกิจกรรมที่เป็นอันตราย เช่น การให้สิทธิ์ในการเข้าถึงทรัพยากรคอมพิวเตอร์มากเกินไป หรือการปิดการควบคุมความปลอดภัยตามมาตรฐานการใช้งาน ได้แก่ ปิดการทำงานของไฟร์วอลล์ (Firewall) ของระบบปฏิบัติการ เป็นต้น ทั้งนี้การกำหนดค่าผิดพลาดอาจจะเป็นสาเหตุหลักของการละเมิดข้อมูล และอาจจะทำให้เกิดการลบหรือแก้ไขทรัพยากร รวมถึงทำให้บริการอื่นหยุดการทำงานได้

3. **การขาดสถาปัตยกรรมและกลยุทธ์ด้านการรักษาความปลอดภัยบนคลาวด์** (Lack of Cloud Security Architecture and Strategy) หมายถึง การย้ายโครงสร้างพื้นฐานด้านเทคโนโลยีสารสนเทศบางส่วนไปยังคลาวด์ ต้องคำนึงถึงสถาปัตยกรรมด้านการรักษาความปลอดภัยที่เหมาะสมมาใช้เพื่อรองรับการโจมตีทางไซเบอร์ ซึ่งหลาย ๆ องค์กรยังขาดความเข้าใจเกี่ยวกับโมเดลความรับผิดชอบด้านความปลอดภัยร่วมกัน นอกจากนี้ฟังก์ชันการทำงานและความเร่งรีบในการย้ายข้อมูลไปบนคลาวด์ก็เป็นอีกปัจจัยหนึ่งที่นำไปสู่การขาดสถาปัตยกรรมและกลยุทธ์ด้านการรักษาความปลอดภัยบนคลาวด์

4. **การระบุตัวตน ข้อมูลประจำตัว การเข้าถึงและการจัดการคีย์ไม่เพียงพอ** (Insufficient Identity, Credential, Access and Key Management) หมายถึง คลาวด์ส่งผล

กระทบต่อระบบการจัดการเกี่ยวกับการระบุตัวตน ข้อมูลประจำตัว และการจัดการการเข้าถึง รวมถึงเครื่องมือและนโยบายตามแนวปฏิบัติแบบดั้งเดิม ซึ่งช่วยให้องค์กรสามารถจัดการตรวจสอบ และรักษาความปลอดภัยในการเข้าถึงทรัพยากรคอมพิวเตอร์ได้ แต่เมื่อต้องจัดการสิ่งเหล่านี้บนคลาวด์กลับเป็นปัญหาสำคัญมากที่ผู้ใช้บริการคลาวด์จำเป็นจะต้องจัดการเกี่ยวกับการระบุตัวตน ข้อมูลประจำตัว และการจัดการการเข้าถึงโดยไม่สูญเสียความปลอดภัย หากการจัดการไม่เพียงพออาจจะนำไปสู่การถูกโจมตี เช่น การถูกปลอมแปลงข้อมูลประจำตัวของผู้ใช้บริการคลาวด์

5) **บัญชีผู้ใช้งานถูกขโมย** (Account Hijacking) หมายถึง ภัยคุกคามที่ผู้โจมตีเข้าถึงและนำบัญชีผู้ใช้งานซึ่งอาจจะมีสิทธิพิเศษสูงหรือมีความละเอียดอ่อน เช่น บัญชีผู้ใช้บริการคลาวด์ ซึ่งการได้มาของบัญชีผู้ใช้งานนี้อาจจะมาจากการโจมตีแบบฟิชชิง (Phishing) หรือจากการขโมยข้อมูลประจำตัวเป็นต้น ภัยคุกคามเหล่านี้มีลักษณะเฉพาะตัวและสามารถทำให้เกิดการสูญหายของข้อมูลและทรัพยากรต่าง ๆ ที่บัญชีผู้ใช้บริการคลาวด์ที่ถูกโจมตีนั้นครอบครองอยู่

6. **ภัยคุกคามจากภายใน** (Insider Threat) หมายถึง บุคคลที่มีสิทธิ์หรือได้รับอนุญาตให้เข้าถึงทรัพย์สินขององค์กร ใช้การเข้าถึงที่ได้รับมาไม่ว่าจะประสงค์ร้ายหรือไม่ตั้งใจก็ตาม เพื่อกระทำการในลักษณะที่อาจส่งผลเสียต่อองค์กร ซึ่งบุคคลจากภายในอาจเป็นพนักงานปัจจุบันหรืออดีต ผู้รับเหมา หรือหุ้นส่วนทางธุรกิจที่เชื่อถือได้ ทำให้สามารถเข้าถึงเครือข่าย ระบบคอมพิวเตอร์ และข้อมูลขององค์กรที่มีความละเอียดอ่อนได้โดยตรง

7. **ส่วนต่อประสานและส่วนต่อประสานโปรแกรมประยุกต์ที่ไม่ปลอดภัย** (Insecure Interfaces and APIs) หมายถึง ผู้ให้บริการคลาวด์จะเปิดเผยส่วนต่อประสานผู้ใช้ซอฟต์แวร์และส่วนต่อประสานโปรแกรมประยุกต์ เพื่อให้ผู้ใช้บริการคลาวด์สามารถจัดการและโต้ตอบกับบริการคลาวด์ได้ เช่น ส่วนต่อประสานเกี่ยวกับการระบุตัวตน การควบคุมการเข้าถึง การเข้ารหัส และการตรวจสอบกิจกรรมเป็นต้น ดังนั้นความปลอดภัยและความพร้อมใช้งานของบริการคลาวด์จึงขึ้นอยู่กับความปลอดภัยของส่วนต่อประสานเหล่านี้ หากออกแบบส่วนต่อประสานไม่ดีพออาจจะนำไปสู่การละเมิดข้อมูลสำคัญบางอย่างได้

8. **ขอบเขตการควบคุมที่อ่อนแอ** (Weak Control Plane) หมายถึง บุคคลหรือกลุ่มบุคคลที่รับผิดชอบไม่ได้ควบคุมตรรกะ ควบคุมความปลอดภัย และควบคุมการตรวจสอบของโครงสร้างพื้นฐานข้อมูลได้อย่างเต็มที่ เช่น สถานการณ์ของผู้มีส่วนได้ส่วนเสีย (Stakeholder) ไม่ทราบการกำหนดค่าความปลอดภัย วิธีการไหลของข้อมูล จุดบอด และจุดอ่อนที่มีอยู่ในเชิง

สถาปัตยกรรม ข้อจำกัดเหล่านี้อาจจะส่งผลให้ข้อมูลเสียหาย ข้อมูลไม่พร้อมใช้งาน หรือข้อมูลรั่วไหลได้

9. **ความล้มเหลวของโครงสร้างเมตาและโครงสร้างแอปพลิเคชันของคลาวด์ (Metastructure and Applistructure Failures)** หมายถึง ผู้ให้บริการคลาวด์จะเปิดเผยการดำเนินการและการป้องกันความปลอดภัยเพื่อปรับใช้และปกป้องคลาวด์ของตนเอง ซึ่งสิ่งเหล่านี้ถูกรวมอยู่ในโครงสร้างเมตาของผู้ให้บริการคลาวด์อยู่แล้ว ทั้งนี้โครงสร้างเมตาถูกพิจารณาว่าเป็นเส้นแบ่งเขตระหว่างผู้ให้บริการคลาวด์ (Cloud Provider) และผู้ใช้บริการคลาวด์ (Cloud Consumer) ความล้มเหลวของแบบจำลองโครงสร้างเมตามีหลายระดับ เช่น การเรียกใช้ APIs ที่ออกแบบมาไม่ดีพอของผู้ให้บริการคลาวด์อาจจะทำให้ผู้โจมตีมีโอกาสรบกวนผู้ใช้บริการคลาวด์ เช่น รบกวนการรักษาความลับ รบกวนความถูกต้องสมบูรณ์ และรบกวนความพร้อมใช้งานของบริการคลาวด์ สำหรับโครงสร้างแอปพลิเคชันของคลาวด์ จะเกี่ยวข้องกับการที่ผู้ใช้บริการคลาวด์จำเป็นต้องเข้าใจวิธีการนำแอปพลิเคชันคลาวด์ไปใช้อย่างเหมาะสมในแต่ละสภาพแวดล้อมของคลาวด์ จึงจะใช้แพลตฟอร์มคลาวด์ได้อย่างเต็มประสิทธิภาพ ความล้มเหลวของโครงสร้างแอปพลิเคชันของคลาวาด์ส่วนใหญ่เกิดจากการนำแอปพลิเคชันที่ออกแบบไว้สำหรับแต่ละสภาพแวดล้อมของคลาวด์ไปใช้อย่างไม่เหมาะสมนั่นเอง เช่น นำแอปพลิเคชันของคลาวด์ที่ออกแบบไว้สำหรับ SaaS คลาวด์ไปใช้กับ IaaS คลาวด์

10. **การมองเห็นการใช้งานคลาวด์ที่ถูกจำกัด (Limited Cloud Usage Visibility)** หมายถึง การที่องค์กรไม่มีความสามารถในการจินตนาการและวิเคราะห์เกี่ยวกับการใช้บริการคลาวด์ภายในองค์กรนั้นมีความปลอดภัยหรือเป็นอันตรายอย่างไร ซึ่งแบ่งออกเป็นสองประเด็น คือ ประเด็นแรกเกี่ยวข้องกับการใช้แอปพลิเคชันที่ไม่ได้รับอนุญาต หมายถึง การที่พนักงานใช้แอปพลิเคชันและทรัพยากรของคลาวด์ โดยไม่ได้รับอนุญาตและการสนับสนุนเฉพาะจากฝ่ายเทคโนโลยีสารสนเทศและความปลอดภัยขององค์กร และประเด็นที่สองเกี่ยวข้องกับการใช้แอปพลิเคชันที่ได้รับอนุญาตในทางที่ผิด หมายถึง การที่แอปพลิเคชันที่ได้รับอนุญาตขององค์กรถูกใช้ประโยชน์โดยบัญชีผู้ใช้งานขององค์กรเอง แต่การใช้งานนี้เกิดขึ้นจากผู้โจมตีที่ใช้บัญชีผู้ใช้งานขององค์กรซึ่งได้จากการถูกขโมยข้อมูลประจำตัว

11. **การใช้บริการคลาวด์ในทางที่ผิด (Abuse and Nefarious Use of Cloud Services)** หมายถึง การที่ผู้ประสงค์ร้ายอาจใช้ประโยชน์จากทรัพยากรคลาวด์ เพื่อโจมตีผู้ใช้บริการคลาวด์ องค์กร หรือผู้ให้บริการคลาวด์อื่น เช่น ใช้บริการคลาวด์เพื่อเป็นโฮสต์มัลแวร์ในการแพร่กระจายมัลแวร์ไปยังผู้ใช้บริการคลาวด์รายอื่นหรือใช้เพื่อส่งอีเมลสแปม (Spam Mail) และฟิชชิง (Phishing)

จากรายงานเกี่ยวกับภัยคุกคามต่อคลาวด์ขององค์กร CSA (CSA, 2019) ที่ได้กล่าวมาข้างต้นทำให้ทราบว่า IaaS คลาวด์ได้รับผลกระทบจากภัยคุกคามต่อคลาวด์ทั้ง 11 ประเภท สิ่งนี้ยิ่งย้ำเตือนถึงความสำคัญของการรักษาความปลอดภัยของ IaaS คลาวด์ รวมถึงแนวทางการบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามดังกล่าว จากงานวิจัยของ Auxsorn et al. (2020); Wiriya et al. (2020); Wongthai et al. (2013b) ซึ่งนักวิจัยเหล่านี้ได้กล่าวถึงแนวทางการบรรเทาความเสี่ยงเกี่ยวกับภัยคุกคามตาม CSA ใน IaaS คลาวด์ คือ การใช้ระบบบันทึกเหตุการณ์ (Logging System) ซึ่งระบบนี้ช่วยอำนวยความสะดวกในการตรวจสอบเหตุการณ์ที่น่าสงสัย เช่น โปรเซสไหนกำลังเข้าถึงไฟล์ที่มีความสำคัญของผู้ใช้บริการคลาวด์ในสภาพแวดล้อมของ IaaS คลาวด์ โดยที่ระบบสามารถตรวจจับและรายงานว่ามีผู้ใช้งานที่ไม่ได้รับอนุญาตทำอะไรกับไฟล์เป็นต้น ดังนั้นปัญหาวิจัยหลักในวิทยานิพนธ์เล่มนี้มีเป้าหมายที่จะประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

เนื้อหาทั้งหมดในหัวข้อสถาปัตยกรรมและปัญหาภัยคุกคามของการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ ได้แบ่งเป็นหัวข้อย่อย ได้แก่ หัวข้อสถาปัตยกรรมของ IaaS คลาวด์ และหัวข้อปัญหาภัยคุกคามของ IaaS คลาวด์ โดยผู้วิจัยขอสรุปเนื้อหาทั้งหมดดังต่อไปนี้

จากภาพ 1 ที่เสนอโดย (Wongthai, 2014) แสดงสถาปัตยกรรมของ IaaS คลาวด์ซึ่งพิจารณาจากมุมมองของผู้ให้บริการประกอบด้วย 2 ฝ่าย คือ ฝ่ายผู้ให้บริการและฝ่ายผู้ใช้บริการ โดยส่วนประกอบหลักของสถาปัตยกรรมของ IaaS คลาวด์ตามภาพ 1 ได้แก่ hw hypervisor dom0 และ domU โดยรายละเอียดของแต่ละส่วนประกอบได้ถูกอธิบายไว้ในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์

จากผลการสำรวจขององค์กรการ์เนอร์ (Gartner, 2022) ซึ่งคาดการณ์เกี่ยวกับคลาวด์ สาธารณะถูกนำมาใช้ในองค์กรเพิ่มขึ้นเนื่องจากความสามารถในการปรับขนาดได้และมีความยืดหยุ่น และ IaaS คลาวด์มีอัตราการเติมโตสูงสุด ดังนั้น IaaS คลาวด์จึงมีโอกาสประสบปัญหาเรื่องความปลอดภัยมากกว่าคลาวด์ประเภทอื่น ๆ อีกทั้ง IaaS คลาวด์ยังเป็นโครงสร้าง

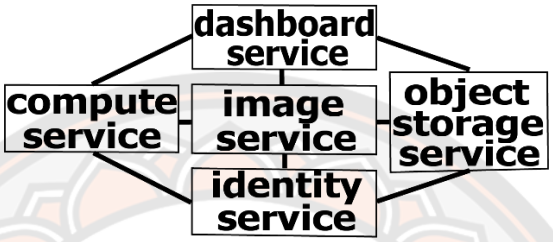
พื้นฐานสำหรับการสร้างคลาวด์ประเภทอื่น เช่น PaaS และ SaaS คลาวด์ หากประสบปัญหาเรื่องความปลอดภัยย่อมส่งผลกระทบต่อเนื่องไปยังคลาวด์ประเภทอื่นอย่างหลีกเลี่ยงไม่ได้ สำหรับภัยคุกคามของ IaaS คลาวด์ที่ระบุโดยองค์กร CSA ตามรายงานเกี่ยวกับภัยคุกคามต่อคลาวด์ปี ค.ศ. 2019 ชื่อ “Top Threats to Cloud Computing: Egregious Eleven” ได้แบ่งภัยคุกคามต่อคลาวด์ทั้งหมด 11 ประเภท ดังต่อไปนี้ คือ 1) การละเมินข้อมูล 2) การกำหนดค่าผิดพลาดและการควบคุมการเปลี่ยนแปลงไม่เพียงพอ 3) การขาดสถาปัตยกรรมและกลยุทธ์ด้านการรักษาความปลอดภัยบนคลาวด์ 4) การระบุตัวตน ข้อมูลประจำตัว การเข้าถึงและการจัดการคีย์ไม่เพียงพอ 5) บัญชีผู้ใช้งานถูกขโมย 6) ภัยคุกคามจากภายใน 7) ส่วนต่อประสานและส่วนต่อประสานโปรแกรมประยุกต์ที่ไม่ปลอดภัย 8) ขอบเขตการควบคุมที่อ่อนแอ 9) ความล้มเหลวของโครงสร้างเมตาและโครงสร้างแอปพลิเคชันของคลาวด์ 10) การมองเห็นการใช้งานคลาวด์ที่ถูกจำกัด และ 11) การใช้บริการคลาวด์ในทางที่ผิด ซึ่งจากรายงานดังกล่าวทำให้ทราบว่า IaaS คลาวด์ได้รับผลกระทบจากภัยคุกคามต่อคลาวด์ทั้ง 11 ประเภท สิ่งนี้ยิ่งย้ำเตือนถึงความสำคัญของการรักษาความปลอดภัยของ IaaS คลาวด์ รวมถึงแนวทางการบรรเทาความเสี่ยงที่เกี่ยวกับภัยคุกคามตาม CSA และหนึ่งในแนวทางการบรรเทาความเสี่ยงเกี่ยวกับภัยคุกคามตาม CSA ใน IaaS คลาวด์ก็คือ ระบบบันทึกเหตุการณ์ ซึ่งระบบนี้ช่วยอำนวยความสะดวกในการตรวจสอบเหตุการณ์ที่น่าสงสัยได้

สถาปัตยกรรมของโอเพนสแต็ก

ปัจจุบันซอฟต์แวร์สำหรับการสร้างคลาวด์มีทั้งรูปแบบลิขสิทธิ์และโอเพนซอร์ส ตัวอย่างซอฟต์แวร์รูปแบบที่มีลิขสิทธิ์ เช่น VMware และสำหรับซอฟต์แวร์รูปแบบโอเพนซอร์ส เช่น Eucalyptus OpenNebula และ OpenStack อย่างไรก็ตามมีแนวโนมการนำซอฟต์แวร์รูปแบบโอเพนซอร์สมาใช้สำหรับสร้างคลาวด์ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ในองค์กรเพิ่มขึ้นเรื่อย ๆ ซึ่ง OpenStack.org (2022); Vailshery (2022) ได้กล่าวไว้ โอเพนสแต็ก (OpenStack) ซึ่งถือว่าเป็นระบบปฏิบัติการคลาวด์ (Sefraoui et al., 2012) รูปแบบโอเพนซอร์สตัวหนึ่งสำหรับนำมาสร้างคลาวด์ ทั้งนี้จากการทำสถิติของ Vailshery (2022) ทำให้ทราบว่าโอเพนสแต็กได้รับความนิยมมากที่สุดในกลุ่มของโอเพนซอร์สและมีการเติบโตอย่างต่อเนื่องในช่วงสามสี่ปีที่ผ่านมา ตัวอย่างกลุ่มองค์กรที่นำโอเพนสแต็กไปใช้งาน เช่น ผู้ให้บริการคลาวด์ หน่วยงานภาครัฐ องค์กรธุรกิจ และสถาบันการศึกษา เป็นต้น

ดังนั้นในวิทยานิพนธ์เล่มนี้ผู้วิจัยให้ความสนใจที่จะใช้โอเพนสแต็กเพื่อสร้าง IaaS คลาวด์ให้ใกล้เคียงกับสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment)

สำหรับใช้ในการวิจัยนี้ และจากงานวิจัยของ Sefraoui et al. (2012) ที่ได้ทำการเปรียบซอฟต์แวร์ที่ใช้สำหรับสร้าง IaaS คลาวด์ เช่น Eucalyptus OpenNebula และ OpenStack ทำให้ทราบว่าโอเพนสแต็ก (OpenStack) เป็นตัวเลือกที่เหมาะสมเนื่องจากมีความสามารถใน การปรับขนาด (Scalability) การใช้แทนกันได้ (Compatible) ความยืดหยุ่น (Flexible / Elasticity) และเป็นโอเพนซอร์ท (Open-source)



ภาพ 2 สถาปัตยกรรมของโอเพนสแต็ก

หมายเหตุ: มีการปรับเปลี่ยนองค์ประกอบ
ที่มา: Sefraoui et al. (2012)

จากภาพ 1 สถาปัตยกรรมของ IaaS คลาวด์ ผู้วิจัยจะจำลองการทำหน้าที่เป็นผู้ให้บริการ IaaS คลาวด์ โดยใช้โอเพนสแต็กสำหรับสร้าง IaaS คลาวด์เพื่อให้ใกล้เคียงกับสภาพแวดล้อมที่ใช้ในการผลิตจริง โดยการติดตั้งเซอร์วิสจากภาพ 2 สถาปัตยกรรมของโอเพนสแต็ก ซึ่งสถาปัตยกรรมนี้มีการปรับเปลี่ยนองค์ประกอบบางส่วนจากสถาปัตยกรรมของโอเพนสแต็กของ Sefraoui et al. (2012) โดยการปรับลดรายละเอียดของแต่ละเซอร์วิสและเพิ่มเครือข่ายการสื่อสารหรือเส้นทึบในภาพ ให้คงเหลือรายละเอียดที่จำเป็นสำหรับการวิจัยนี้ ซึ่งประกอบด้วย 5 เซอร์วิสสำหรับสร้าง IaaS คลาวด์ ได้แก่ คอมพิวเซอร์วิส (Compute Service) อิมเมจเซอร์วิส (Image Service) อ็อบเจกต์สตอเรสท์เซอร์วิส (Object Storage Service) แดชบอร์ดเซอร์วิส (Dashboard Service) และไอเด็นติตี้เซอร์วิส (Identity Service) โดยแต่ละเซอร์วิสเชื่อมต่อกันผ่านเครือข่ายการสื่อสารหรือเส้นทึบในภาพ 2 รายละเอียดแต่ละเซอร์วิสมาจากเว็บไซต์เอกสารอย่างเป็นทางการของโอเพนสแต็ก (OpenStack.org, 2019) ดังต่อไปนี้

1. **คอมพิวเซอร์วิส** (Compute Service) หมายถึง กล่องด้านซ้ายสุดของภาพ 2 ซึ่งช่วยให้ผู้ให้บริการสามารถควบคุม IaaS คลาวด์ได้ เซอร์วิสนี้ยังอนุญาตให้ผู้ให้บริการความคุมเครื่องเสมือน (Virtual Machine: VM) เครือข่าย (Network) และจัดการการเข้าถึงคลาวด์ผ่านบัญชีผู้ใช้งาน นอกจากนี้ยังกำหนดไดรเวอร์ที่โต้ตอบกับกลไกการจำลองเสมือน (Virtualization Mechanism) หรือไฮเปอร์ไวเซอร์ที่ทำงานบนระบบปฏิบัติการของคอมพิวเตอร์แม่งาน (Host Computer) หรือโฮสต์ หรือ dom0 ของผู้ให้บริการ และเซอร์วิสนี้ยังเผยแพร่ฟังก์ชันการทำงานสำหรับเซอร์วิสอื่น ๆ ผ่านทางเว็บ APIs คอมพิวเซอร์วิสมักจะถูกออกแบบมาเพื่อเรียกใช้อิมเมจเซอร์วิส (Image Service)

2. **อิมเมจเซอร์วิส** (Image Service) หมายถึง กล่องตรงกลางของภาพ 2 ทำหน้าที่จัดเก็บอิมเมจของเครื่องเสมือนหรือ VM และจัดการรายการของอิมเมจของ VM ที่สามารถเข้าถึงได้ ซึ่งอิมเมจของ VM คือ ไฟล์ที่สามารถนำมาเรียกใช้งานแล้วจะกลายเป็นเครื่องเสมือนหรือ domU

3) **อ็อบเจกต์สตอเรสท์เซอร์วิส** (Object Storage Service) หมายถึง กล่องด้านขวาสุดของภาพ 2 ที่ใช้สำหรับจัดเก็บข้อมูลขนาดใหญ่ สามารถปรับขนาดได้โดยใช้การทำคลัสเตอร์ของเครื่องเซิร์ฟเวอร์มาตราฐาน เพื่อจัดเก็บข้อมูลขนาดใหญ่ในระดับเพตะไบต์ (Petabyte: PB) และสามารถปรับปรุงหรือค้นคืนข้อมูลที่จัดเก็บได้

4. **แดชบอร์ดเซอร์วิส** (Dashboard Service) หมายถึง กล่องด้านบนสุดของภาพ 2 เป็นส่วนต่อประสานบนเว็บที่ช่วยให้ผู้ให้บริการจัดการทรัพยากรและบริการของโอเพนสแต็ก และโต้ตอบกับตัวควบคุมคอมพิวเซอร์วิสผ่าน APIs ของโอเพนสแต็ก

5. **ไอเด็นติตี้เซอร์วิส** (Identity Service) หมายถึง กล่องด้านล่างสุดของภาพ 2 ซึ่งเป็นระบบจัดการข้อมูลประจำตัวเริ่มต้นสำหรับโอเพนสแต็ก เมื่อผู้ให้บริการ IaaS คลาวด์ติดตั้งเซอร์วิสนี้แล้ว ก็จะสามารถกำหนดค่าต่าง ๆ ของข้อมูลสำหรับเริ่มต้นการทำงานของเซอร์วิส เช่น ข้อมูลเกี่ยวกับบัญชีผู้ใช้งานระดับผู้บริหารระบบ เป็นต้น

เนื้อหาทั้งหมดในหัวข้อสถาปัตยกรรมของโอเพนสแต็ก แสดงให้เห็นถึงแต่ละส่วนประกอบของโอเพนสแต็กที่ใช้สำหรับสร้าง IaaS คลาวด์ ที่ประกอบด้วย 5 เซอร์วิส ได้แก่ คอมพิวเซอร์วิส (Compute Service) อิมเมจเซอร์วิส (Image Service) อ็อบเจกต์สตอเรสท์เซอร์วิส (Object Storage Service) แดชบอร์ดเซอร์วิส (Dashboard Service) และไอเด็นติตี้เซอร์วิส (Identity Service) ซึ่งในวิทยานิพนธ์เล่มนี้ให้ความสนใจที่จะใช้โอเพนสแต็กเพื่อสร้าง IaaS คลาวด์ให้ใกล้เคียงกับสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production

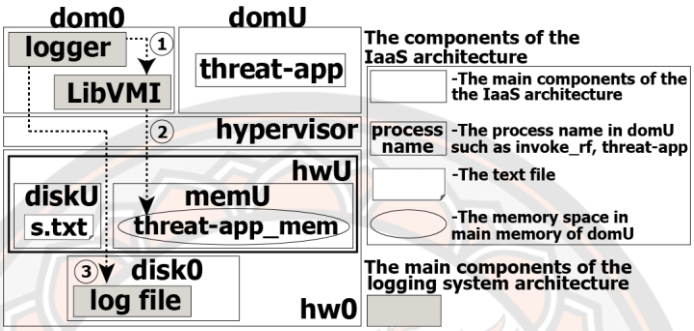
Environment) เนื่องจากเป็นตัวเลือกที่เหมาะสมจากการเปรียบเทียบซอฟต์แวร์ที่ใช้สำหรับสร้าง IaaS คลาวด์จากงานวิจัยของ Sefraoui et al. (2012) ประกอบกับจากการทำสถิติของ Vailshery (2022) ที่แสดงให้เห็นว่าโอเพนสแต็กได้รับความนิยมมากที่สุดในกลุ่มของโอเพนซอร์ส และมีการเติบโตอย่างต่อเนื่อง ตัวอย่างองค์ที่นำโอเพนสแต็กไปสร้าง IaaS คลาวด์ เช่น Rackspace Technology DreamHost Cisco Developer AT&T และ Walmart เป็นต้น สำหรับในประเทศไทยองค์กรที่นำโอเพนสแต็กมาประยุกต์ใช้ เช่น True IDC AIS CAT Telecom และ Synnex (Thailand) Public Company Limited เป็นต้น ประกอบกับโอเพนสแต็กยังมีความสามารถในการปรับขนาด (Scalability) การใช้แทนกันได้ (Compatible) ความยืดหยุ่น (Flexible / Elasticity) และเป็นโอเพนซอร์ท (Open-source) อย่างไรก็ตามอ็อบเจกต์สตอเรสท์เซอร์วิส (Object Storage Service) ในภาพ 2 ไม่เกี่ยวข้องกับการทดลองในวิทยานิพนธ์เล่มนี้

สถาปัตยกรรมของระบบบันทึกเหตุการณ์ในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

ในหัวข้อนี้ผู้วิจัยจะอธิบายเกี่ยวกับสถาปัตยกรรมของระบบบันทึกเหตุการณ์สำหรับ IaaS คลาวด์ในสภาพแวดล้อมของห้องปฏิบัติการ เพื่อให้ผู้อ่านได้เข้าใจภาพรวมของระบบบันทึกเหตุการณ์และการทำงานของระบบบันทึกเหตุการณ์ในแต่ละองค์ประกอบ เพื่อเป็นข้อมูลพื้นฐานในการทำความเข้าใจเกี่ยวกับระบบบันทึกเหตุการณ์ในวิทยานิพนธ์เล่มนี้

จากงานวิจัยของ Auxsorn et al. (2020); Wiriya et al. (2020); Wongthai (2014); Wongthai et al. (2013b) ได้ทดลองระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ที่ไม่ใช่สภาพแวดล้อมที่ใช้ในการผลิตจริง (Non-real-world Production Environment) ซึ่งสภาพแวดล้อมของห้องปฏิบัติการนี้สร้างจากคอมพิวเตอร์กายภาพเพียงเครื่องเดียวเพื่อจำลอง IaaS คลาวด์ตามภาพ 3 สถาปัตยกรรมของ IaaS คลาวด์และสถาปัตยกรรมของระบบบันทึกเหตุการณ์ ซึ่งมีการปรับเปลี่ยนองค์ประกอบบางส่วนจากงานวิจัยของ Wiriya et al. (2020) เพื่อให้สื่อความหมายตรงตามงานวิจัยในวิทยานิพนธ์เล่มนี้ คือ โปรเซส threat-app จากเดิมเป็นโปรเซส read ซึ่งทั้งสองโปรเซสจะทำหน้าที่เหมือนกัน จากหัวข้อสถาปัตยกรรมของ IaaS คลาวด์ ซึ่งได้อธิบายเกี่ยวกับสถาปัตยกรรมของ IaaS คลาวด์ไปแล้วบางส่วนว่า ประกอบด้วย 2 ฝ่าย คือ ฝ่ายผู้ให้บริการ (Provide Side) หรือเรียกว่า ผู้ให้บริการ (Provider) และฝ่ายผู้ใช้บริการ (Customer Side) หรือเรียกว่า ผู้ใช้บริการ (Customer) ซึ่งภาพ 3 นี้จะอธิบายสถาปัตยกรรมของ IaaS คลาวด์ที่มีรายละเอียดเพิ่มเติมในมุมมองของฝ่ายผู้ให้บริการ ทั้งนี้จะแยกอธิบายเป็น 2 ส่วน

หลัก คือ ส่วนของสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ หมายถึง กล่องสีขาวทั้งหมด วงรี และรูปร่างของไฟล์ข้อความในภาพ 3 คือ ส่วนประกอบของสถาปัตยกรรมของ IaaS คลาวด์ และส่วนของสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ หมายถึง กล่องสีเทาทั้งหมดในภาพ 3 คือ ส่วนประกอบหลักของระบบบันทึกเหตุการณ์ ซึ่งผู้วิจัยจะได้อธิบายรายละเอียดด้านล่าง



ภาพ 3 สถาปัตยกรรมของ IaaS คลาวด์และสถาปัตยกรรมของระบบบันทึกเหตุการณ์

หมายเหตุ: มีการปรับเปลี่ยนองค์ประกอบ

ที่มา: Wiriya et al. (2020)

1. สถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ หมายถึง กล่องสีขาวทั้งหมด วงรี และรูปร่างของไฟล์ข้อความในภาพ 3 ซึ่งกล่องสีขาวประกอบด้วยไฮเปอร์ไวเซอร์ dom0 hw0 disk0 domU hwU diskU และ memU ชื่อของส่วนประกอบเหล่านี้ที่ลงท้ายด้วย “0” แสดงถึงส่วนประกอบทางกายภาพนี้ถูกเป็นเจ้าของและถูกจัดการโดยผู้ให้บริการ IaaS คลาวด์ สำหรับส่วนประกอบที่ลงท้ายด้วย “U” แสดงถึงส่วนประกอบเสมือนซึ่งจะถูกเป็นเจ้าของและถูกจัดการโดยผู้ใช้บริการ IaaS คลาวด์ โดยมีรายละเอียดแต่ละส่วนดังต่อไปนี้

1. ไฮเปอร์ไวเซอร์ (Hypervisor) หมายถึง กล่องสีขาวที่มีหมายเลข 2 ในภาพ 3 คือ ซอฟต์แวร์ที่ให้สิทธิ์คอมพิวเตอร์กายภาพเครื่องหนึ่งสามารถเป็นโฮสต์ของเครื่องเสมือนหรือ VM หนึ่งเครื่องหรือมากกว่า

2. กล่องสีขาวที่อยู่ใน domU วงรีที่อยู่ใน memU และรูปร่างของไฟล์ข้อความในภาพ 3 หมายถึง ส่วนประกอบสำหรับการจำลองเหตุการณ์บางอย่างในเป้าประสงค์การทดลองใน

วิทยานิพนธ์เล่มนี้ เช่น โปรเซสชื่อ threat-app เป็นโปรเซสของแอปพลิเคชันตัวหนึ่งที่ใช้สำหรับจำลองสถานการณ์ว่าผู้โจมตีเข้ามาควบคุมและใช้แอปพลิเคชันดังกล่าวเปิดอ่านไฟล์ข้อความชื่อ s.txt ซึ่งไฟล์นี้เก็บข้อมูลที่มีความสำคัญของผู้ใช้บริการ

3. dom0 หรือโดเมน 0 หมายถึง โดเมนที่มีสิทธิพิเศษของซอฟต์แวร์ Xen ซึ่งเป็นไฮเปอรไวเซอร์ตัวหนึ่งที่จะเริ่มต้นทำงานเมื่อบูตระบบปฏิบัติการของเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของ VM หรือ domU และยังถือว่าเป็น VM หนึ่งอีกด้วย ทั้งนี้ dom0 เป็นตัวเดียวกับ dom0 ที่ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์ โดย dom0 นี้มีความสามารถเข้าถึง hw0 ได้โดยตรงและเป็นผู้จัดการ domU ทั้งหมดที่สร้างขึ้นโดยผู้ใช้บริการ ดูกล่องสีขาวด้านซ้ายบนสุดในภาพ 3

4. hw0 หมายถึง ฮาร์แวร์ทางกายภาพและถูกจัดการโดย dom0 ดูกล่องสีขาวด้านล่างในภาพ 3 ทั้งนี้ hw0 เป็นตัวเดียวกับ hw ที่ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์

5. domU หรือโดเมนผู้ใช้ หมายถึง เครื่องเสมือน (VM) ของผู้ใช้บริการที่สร้างโดย dom0 ดูกล่องสีขาวด้านขวาบนสุดในภาพ 3 ซึ่ง domU จะทำงานอยู่บนไฮเปอร์ไวเซอร์ และถือว่าเป็นผลิตภัณฑ์ของ IaaS คลาวด์ที่ผู้ให้บริการเปิดให้ผู้ใช้บริการเช่าใช้งานและไม่สามารถเข้าถึง hw0 ได้โดยตรง ทั้งนี้ domU เป็นตัวเดียวกับ domU ที่ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์

6. hwU หมายถึง ฮาร์แวร์เสมือนของ domU ที่ตั้งอยู่ใน hw0 ซึ่งในทางกายภาพผู้ให้บริการจะเป็นเจ้าของและถูกจัดการผ่าน dom0 อย่างไรก็ตามในทางฮาร์แวร์เสมือนแล้วจะถูกเป็นเจ้าของและถูกจัดการโดย domU หรือผู้ใช้บริการ

7. disk0 หมายถึง ดิสก์ทางกายภาพของ dom0 และถือว่าเป็นฮาร์แวร์ทางกายภาพตัวหนึ่ง

8. diskU หมายถึง ดิสก์เสมือนของ domU และถือว่าเป็นฮาร์แวร์เสมือนของ domU ตัวหนึ่ง

9. memU หมายถึง หน่วยความจำหลักเสมือนของ domU

2. สถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ หมายถึง กล่องสีเทาทั้งหมดในภาพ 3 คือ ส่วนประกอบหลักของระบบบันทึกเหตุการณ์ ได้แก่ ล็อกกิงโปรเซส (Logging Process) และล็อกไฟล์ (Log File) (Auxsorn et al., 2020; Wongthai, 2014) สถาปัตยกรรมของระบบบันทึกเหตุการณ์ที่มีอยู่ในภาพ 3 สามารถบันทึกเหตุการณ์ที่เกิดขึ้นใน domU หรือ VM ของผู้ใช้บริการ เช่น ใครมีสิทธิ์เข้าถึงหรือเกิดอะไรขึ้นกับไฟล์ของผู้ใช้บริการในดิกส์ของ VM หรือ diskU (Ko, Jagadpramana, & Lee, 2011; Wongthai, 2014) ในวิทยานิพนธ์เล่มนี้การอ้างอิงถึง ล็อกกิงโปรเซส จะใช้คำย่อว่า “ล็อกเกอร์” หรือ logger และสถาปัตยกรรมของระบบบันทึกเหตุการณ์นี้จะถูกนำไปใช้กับการทดลองในวิทยานิพนธ์เล่มนี้ กล่องสีขาวที่อยู่ใน domU ในภาพ 3 คือ โปรเซส threat-app เป็นโปรเซสของแอปพลิเคชันตัวหนึ่งที่ใช้สำหรับจำลองสถานการณ์ว่าผู้โจมตีเข้ามาควบคุมและใช้แอปพลิเคชันดังกล่าวเปิดอ่านไฟล์ข้อความชื่อ s.txt ใน diskU ซึ่งไฟล์นี้เก็บข้อมูลที่มีความสำคัญของผู้ใช้บริการ สำหรับ threat-app_mem ที่อยู่ใน memU คือ พื้นที่หน่วยความจำที่ถูกสงวนไว้สำหรับโปรเซส threat-app โดยระบบปฏิบัติการ (Operating System: OS) ที่โปรเซส threat-app กำลังรันอยู่ จากภาพ 3 ระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้องของห้องปฏิบัติการมีขั้นตอนการทำงาน 3 ขั้นตอนโดยแสดงด้วยวงกลมที่มีตัวเลข 1 ถึง 3 ซึ่งมีรายละเอียดแต่ละขั้นตอนดังต่อไปนี้

1. ขั้นตอนที่ 1 คือ ล็อกเกอร์ (Logger) ที่อยู่ใน dom0 เรียกใช้ไลบารี LibVMI (LibVMI เป็นไลบารีเขียนด้วยภาษาซีและถูกติดตั้งใน dom0 มีความสามารถเข้าถึง threat-app_mem ใน memU เพื่อตรวจสอบกิจกรรมที่เป็นอันตรายของโปรเซส threat-app ที่กำลังเปิดอ่านไฟล์ข้อความชื่อ s.txt)

2. ขั้นตอนที่ 2 คือ ไลบารี LibVMI เข้าไปใน memU เพื่อดึงข้อมูลที่ต้องการที่อยู่ใน threat-app_mem เช่น ชื่อไฟล์ s.txt หรือสตริง ‘s.txt’ และโปรเซส ID เป็นต้น จากนั้นจะส่งข้อมูลที่ได้มาไปยังล็อกเกอร์

3. ขั้นตอนที่ 3 คือ ล็อกเกอร์จะจัดการกับข้อมูลที่ได้รับมาก่อนที่จะเขียนลงในล็อกไฟล์ (Log File)

เนื้อหาทั้งหมดในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ได้อธิบายเกี่ยวกับสถาปัตยกรรมของระบบบันทึกเหตุการ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ เพื่อให้ผู้อ่านได้เข้าใจภาพรวมของระบบ

บันทึกเหตุการณ์และการทำงานของระบบบันทึกเหตุการณ์ในแต่ละองค์ประกอบ โดยได้แยกอธิบายเป็น 2 ส่วนหลัก คือ ส่วนของสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ซึ่งหัวข้อสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการเป็นการอธิบายรายละเอียดเพิ่มเติมจากหัวข้อสถาปัตยกรรมของ IaaS คลาวด์ ภายใต้มุมมองของฝ่ายผู้ให้บริการ และส่วนของสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้องของห้องปฏิบัติการ

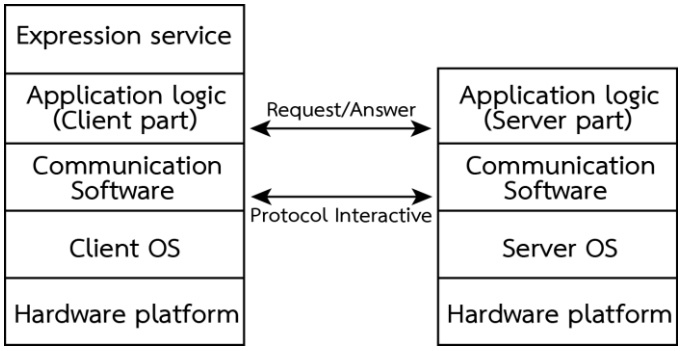
สถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ หมายถึง กล่องสีขาวทั้งหมด วงรี และรูปร่างของไฟล์ข้อความในภาพ 3 ซึ่งกล่องสีขาวประกอบด้วย ไฮเปอร์ไวเซอร์ dom0 hw0 disk0 domU hwU diskU และ memU ชื่อของส่วนประกอบเหล่านี้ที่ลงท้ายด้วย “0” แสดงถึงส่วนประกอบทางกายภาพนี้ถูกเป็นเจ้าของและจัดการโดยผู้ให้บริการ IaaS คลาวด์ สำหรับส่วนประกอบที่ลงท้ายด้วย “U” แสดงถึงส่วนประกอบเสมือนนี้ถูกเป็นเจ้าของและจัดการโดยผู้ใช้บริการ IaaS คลาวด์

สถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้องของห้องปฏิบัติการ มีส่วนประกอบหลัก ได้แก่ ล็อกกิงโปรเซส (Logging Process) และล็อกไฟล์ สถาปัตยกรรมของระบบบันทึกเหตุการณ์นี้ สามารถบันทึกเหตุการณ์ที่เกิดขึ้นใน domU หรือ VM ของผู้ใช้บริการ เช่น ใครมีสิทธิ์เข้าถึงหรือเกิดอะไรขึ้นกับไฟล์ของผู้ใช้บริการในดิกส์ของ VM หรือ diskU

การเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์

จากหัวข้อสถาปัตยกรรมของโอเพนสแต็ก ได้อธิบายเกี่ยวกับสถาปัตยกรรมของโอเพนสแต็ก ซึ่งแต่ละเซอร์วิสในสถาปัตยกรรมนี้มีการสื่อสารและแลกเปลี่ยนข้อมูลผ่านเครือข่าย ดังนั้นถ้าหากจะปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้เหมาะสมกับสภาพแวดล้อมของโอเพนสแต็ก ก็สามารถนำหลักการเขียนโปรแกรมเครือข่าย (Network Programming) เป็นแนวทางการพัฒนา

Xue & Zhu (2009) ได้กล่าวว่าการพัฒนาแอปพลิเคชันบนเครือข่ายคอมพิวเตอร์สามารถใช้หลักการเขียนโปรแกรมเครือข่าย ตัวอย่างเช่น โหมดไคลเอ็นท์/เซิร์ฟเวอร์ (Client/Server) ซึ่งโหมดไคลเอ็นท์/เซิร์ฟเวอร์ หมายถึง การแบ่งปันข้อมูลซึ่งกันและกันระหว่างไคลเอ็นท์ (Client) และเซิร์ฟเวอร์ (Server) ที่ใช้กันอย่างแพร่หลาย ไคลเอ็นท์ หมายถึง คอมพิวเตอร์ส่วนบุคคลหรือเวิร์กสเตชัน และเซิร์ฟเวอร์ หมายถึง เครื่องคอมพิวเตอร์เซิร์ฟเวอร์ที่จัดเตรียมไว้สำหรับให้บริการกับกลุ่มของผู้ใช้งานฝั่งไคลเอ็นท์ที่ใช้โปรแกรมบริการร่วมกัน



ภาพ 4 โครงสร้างระบบไคลเอ็นท์/เซิร์ฟเวอร์

ที่มา: Xue & Zhu (2009)

ภาพ 4 เป็นโครงสร้างระบบไคลเอ็นท์/เซิร์ฟเวอร์ ซึ่งแกนหลักของโครงสร้างนี้ คือ การกระจายงาน (Task) ของแอปพลิเคชันระหว่างไคลเอ็นท์และเซิร์ฟเวอร์ โดยยอมให้มีการแบ่งปันข้อมูลและทรัพยากรระหว่างระบบ เช่น ไฟล์ พื้นที่ดิกส์ หน่วยประมวลผลกลางหรือซีพียู และอุปกรณ์ต่อพวงให้สามารถทำงานร่วมกันและส่งข้อความระหว่างซีพียูจำนวนมากได้ ซึ่งการดำเนินการเหล่านี้ใช้ซ็อกเก็ตเพื่อเตรียมการสื่อสารระหว่างโปรเซส (Inter-process Communication: IPC) ในหลายเครื่อง

Maata, Cordova, Sudramurthy & Halibas (2017) ได้กล่าวว่าการเขียนโปรแกรมซ็อกเก็ตเป็นหนึ่งในวิธีที่ดีที่สุดในการปรับปรุงประสิทธิภาพของระบบการคำนวณแบบกระจาย โดยมีโพรโทคอล (Protocol) สำหรับการสื่อสาร 2 ประเภทที่ใช้กันทั่วไปในการเขียนโปรแกรมซ็อกเก็ต คือ User Datagram Protocol (UDP) เป็นโพรโทคอลแบบไร้การเชื่อมต่อซึ่งจำเป็นต้องทราบที่อยู่ภายในเครือข่ายของซ็อกเก็ตที่รับข้อมูล และ transfer control protocol (TCP) เป็นโพรโทคอลแบบที่ต้องมีการสร้างการเชื่อมต่อเสมือนก่อนระหว่างซ็อกเก็ตตัวส่งและซ็อกเก็ตตัวรับข้อมูล

เนื้อหาทั้งหมดในหัวข้อการเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์ ซึ่งอธิบายเกี่ยวกับการเขียนโปรแกรมเพื่อการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์ โดยใช้หลักการเขียนโปรแกรมเครือข่ายเป็นแนวทางการพัฒนา ซึ่ง Xue & Zhu (2009) ได้กล่าวว่าการพัฒนาแอปพลิเคชันบนเครือข่ายคอมพิวเตอร์สามารถใช้หลักการเขียนโปรแกรมเครือข่าย ตัวอย่างเช่น โหมดไคลเอ็นท์/เซิร์ฟเวอร์ (Client/Server) เพื่อการแบ่งปันข้อมูลซึ่งกันและกันระหว่างไคลเอ็นท์/เซิร์ฟเวอร์ และ Maata et al. (2017) ยังกล่าวอีกว่าการเขียนโปรแกรม

ซ็อกเก็ตเป็นหนึ่งในวิธีที่ดีที่สุดในการปรับปรุงประสิทธิภาพของระบบการคำนวณแบบกระจาย จากหัวข้อสถาปัตยกรรมของโอเพนสแต็ก ได้อธิบายให้เห็นว่าแต่ละเซอร์วิสในสถาปัตยกรรมนี้มีการสื่อสารและแลกเปลี่ยนข้อมูลผ่านเครือข่าย ดังนั้นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้องของโอเพนสแต็กในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจะดำเนินการออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์ใหม่ ด้วยประยุกต์ใช้การเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์ โดยใช้โพรโตคอล TCP เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็ก

การประมวลผลแบบขนาน

การประมวลผลแบบขนาน (Parallel Computing หรือ Parallel Processing) คือ การใช้ทรัพยากรการประมวลผลหลายรายการพร้อมกันเพื่อแก้ปัญหาการคำนวณ (Barney, Livermore Computing (Retired), & Frederick, 2019) นักวิจัยหลายคนใช้การประมวลผลแบบขนานเพื่อปรับปรุงความเร็วของปัญหาการคำนวณ (Fernando, Murad, & Wijanarko, 2018; Sujatha et al., 2015; Thakur, Kumar, & Patle, 2014) โดย Fernando et al. (2018) ได้การจำแนกประเภทของความขนาน (Parallelism) ซึ่งส่วนใหญ่ใช้ดำเนินการในการประมวลผลแบบขนาน เช่น การขนานแบบกระจาย (Distributed Parallel) โปรเซสเซอร์แบบมัลติ-คอร์ (Multicore Processor) การประมวลผลแบบขนานขนาดใหญ่ (Massively Parallel Computing) และหน่วยประมวลผลกราฟิก (Graphic Processing Unit: GPU) Thakur et al. (2014) ได้อธิบายถึงระดับของความขนาน (Level of Parallelism) ประกอบด้วยดังต่อไปนี้

1. Instruction Level Parallelism (ILP) หมายถึง การใช้ประโยชน์จากลำดับของคำสั่งของไมโครโพรเซสเซอร์ที่ต้องใช้ฟังก์ชันแตกต่างกันในการทำงาน เช่น ฟังก์ชันการโหลด (Load) ฟังก์ชันการคำนวณและตรรกะ (Arithmetic and Logical Unit: ALU) และตัวคูณทศนิยม (Floating Point Multiplier)

2. Data Level Parallelism (DLP) หมายถึง การดำเนินการแบบเดียวกันแต่กระทำกับข้อมูลหลาย ๆ ข้อมูลที่แตกต่างกันพร้อม ๆ กัน เช่น การหาผลรวมของค่าตัวเลขที่อยู่ในอาร์เรย์ 1 มิติจำนวน 10 อีลิเมนต์ ได้แก่ [0]…[10-1] โดยจะทำการรันเธรด ‘A’ บนโปรเซสเซอร์คอร์หมายเลข 0 ให้หาผลรวมของค่าตัวเลขที่อยู่ในอาร์เรย์ 1 มิติจำนวน 5 อีลิเมนต์ ([0]…[4]) และทำการรันเธรด ‘B’ บนโปรเซสเซอร์คอร์หมายเลข 1 ให้หาผลรวมของค่าตัวเลขที่อยู่ในอาร์เรย์ 1 มิติจำนวน 5 อีลิเมนต์ ([5]…[9]) ซึ่งเธรดทั้งสองจะทำงานแบบขนานกันบนโปรเซสเซอร์คอร์ที่ต่างกัน

3. Thread Level Parallelism (TLP) หมายถึง ความขนานที่มีอยู่ในแอปพลิเคชันที่ทำงานหลายเธรดพร้อมกัน ซึ่งลักษณะนี้พบมาในแอปพลิเคชันที่พัฒนาขึ้นสำหรับเซิร์ฟเวอร์เชิงพาณิชย์ เช่น โปรแกรมบริหารจัดการฐานข้อมูล ซึ่งรันหลายเธรดพร้อมกันและเพื่อรองรับปริมาณงานจำนวนมาก

จากการทดลองเปรียบเทียบระยะเวลาที่แอปพลิเคชันใช้ในการทำงานสำหรับโปรเซสเซอร์แบบคอร์เดียว และแบบหลายคอร์ เกี่ยวกับการเรียงลำดับแบบฟอง (Bubble Sort) และการค้นหาแบบเชิงเส้น (Linear Search) ของ Sujatha et al. (2015) พบว่าการประมวลผลแบบขนานโดยใช้โปรเซสเซอร์แบบหลายคอร์สามารถเพิ่มความเร็วของการจัดเรียงข้อมูลแบบฟองและการค้นหาแบบเชิงเส้นได้ โดยที่การเรียงลำดับแบบฟอง เป็นอัลกอริทึมการเรียงลำดับแบบง่าย โดยทำการเปรียบเทียบแต่ละคู่ของรายการข้อมูลที่อยู่ติดกันและสลับค่าข้อมูลกันหากพบว่าอยู่ในตำแหน่งไม่ถูกต้อง อัลกอริทึมนี้จะทำซ้ำไปเรื่อย ๆ จนกว่าจะไม่มีการสลับค่าข้อมูลกันสุดท้ายก็จะได้รายการข้อมูลที่ถูกเรียงลำดับในที่สุด (Hammad, 2015) สำหรับการค้นหาแบบเชิงเส้น เป็นอัลกอริทึมการค้นหาโดยจะเริ่มต้นการเปรียบเทียบข้อมูลที่ต้องการค้นหากับข้อมูลแต่ละลำดับในอาร์เรย์ เช่น A[0]…A[N-1] โดยจะเริ่มต้นที่ข้อมูลตัวแรกในอาร์เรย์ A และทำการเปรียบเทียบตามลำดับต่อไปเรื่อย ๆ จนกว่าจะพบข้อมูลที่ต้องการค้นหาหรือค้นหาครบทุกตัวในอาร์เรย์ A ถ้าไม่พบข้อมูลที่ต้องการค้นหาก็จะแสดงว่าข้อมูลนั้นไม่มีอยู่ในอาร์เรย์ A เป็นต้น นอกจากนี้ Marszatek, Woźniak & Połap (2018) ยังได้ปรับปรุงการเรียงลำดับแบบผสาน (Merge Sort) เพื่อให้สามารถนำการจัดเรียงนี้ไปใช้กับโปรเซสเซอร์คอร์จำนวนมากได้โดยที่มีการแจกจ่ายงานอย่างยืดหยุ่นระหว่างโปรเซสเซอร์คอร์ อันจะเป็นการเพิ่มประสิทธิภาพการเรียงลำดับได้ ซึ่งการเรียงลำดับแบบผสาน เป็นอัลกอริทึมการเรียงลำดับที่ทำงานโดยการแบ่งอาร์เรย์ออกเป็นอาร์เรย์ย่อยที่มีขนาดเล็กลง แล้วจัดเรียงแต่ละอาร์เรย์ย่อย หลังจากนั้นก็ผสานอาร์เรย์ย่อยที่จัดเรียงลำดับแล้วกลับเข้าด้วยกันก็จะได้อาร์เรย์ที่เรียงลำดับในขั้นตอนสุดท้าย

เนื้อหาทั้งหมดในหัวข้อการประมวลผลแบบขนาน ซึ่งอธิบายเกี่ยวกับการประมวลผลแบบขนาน ซึ่งเป็นการใช้ทรัพยากรการประมวลผลหลายรายการพร้อมกันเพื่อปรับปรุงความเร็วในการแก้ปัญหาการคำนวณ และมีการแบ่งระดับของความขนาน (Level of Parallelism) เป็น 3 ระดับโดย Thakur et al. (2014) ได้แก่ 1) Instruction Level Parallelism (ILP) 2) Data Level Parallelism (DLP) และ 3) Thread Level Parallelism (TLP) จากการทดลองของ Sujatha et al. (2015) พบว่าการประมวลผลแบบขนานโดยใช้โปรเซสเซอร์แบบหลายคอร์สามารถเพิ่มความเร็วของการจัดเรียงข้อมูลแบบฟอง และการค้นหาแบบเชิงเส้นได้ นอกจากนี้ Marszatek et al.

(2018) ยังได้ปรับปรุงการเรียงลำดับแบบผสาน (Merge Sort) เพื่อให้สามารถนำไปใช้กับโปรเซสเซอร์คอร์จำนวนมากได้โดยที่มีการแจกจ่ายงานอย่างยืดหยุ่นระหว่างโปรเซสเซอร์คอร์ อันจะเป็นการเพิ่มประสิทธิภาพการเรียงลำดับได้ ดังนั้นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้องของโอเพนสแต็กในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจะดำเนินการออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์ใหม่ ด้วยประยุกต์ใช้การประมวลผลแบบขนาน โดยใช้โปรเซสเซอร์แบบมัลติคอร์ และใช้ระดับของความขนานแบบ TLP

การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

Thakur et al. (2014) ได้กล่าวว่าเพื่อให้บรรลุการประมวลผลแบบขนาน ตามหัวข้อการประมวลผลแบบขนาน แอปพลิเคชันหรือโปรแกรมต้องถูกแยกออกเป็นส่วน ๆ ที่มีความเป็นอิสระ เพื่อให้แต่ละโปรเซสเซอร์สามารถเรียกใช้ส่วนหนึ่งของโปรแกรมพร้อมกันได้จากโปรเซสเซอร์อื่น ดังนั้นการประมวลผลแบบขนานสามารถทำได้บนเครื่องคอมพิวเตอร์เครื่องเดียวที่มีโปรเซสเซอร์หลายตัว หรือคอมพิวเตอร์ที่เชื่อมต่อกันด้วยระบบเครือข่าย หรือทั้งสองแบบรวมกัน

ปัจจุบันนี้คอมพิวเตอร์มีซีพียูตั้งแต่หนึ่งตัวหรือหลายตัวได้โดยแต่ละตัวของซีพียูสามารถมีแกนประมวลผลหรือคอร์ได้หลายคอร์ ซึ่งช่วยทำงานหลายอย่างพร้อมกัน เช่น การรันโปรแกรมหลายโปรแกรม รวมถึงระบบปฏิบัติการพร้อมกัน ทำให้นักพัฒนาสามารถแอปพลิเคชันแบบขนานเพื่อแบ่งภาระงาน (Workload) ออกเป็นส่วนงาน (Task) และผสานงานเข้ากับงานหลัก โมเดลการเขียนโปรแกรมแบบขนาน (Parallel Programming Model) หมายถึง การกำหนดสาระสำคัญของสถาปัตยกรรมระบบคอมพิวเตอร์ที่ใช้หน่วยความจำแบบใช้ร่วมกัน (Shared Memory) หรือหน่วยความจำแบบกระจาย (Distributed Memory) (Diaz, Munoz-Caro, & Nino, 2012) ดังนั้นการเขียนโปรแกรมแบบขนานแบ่งได้ 2 ประเภท ดังต่อไปนี้

1. โมเดลหน่วยความจำแบบใช้ร่วมกัน (Shared Memory Model) หมายถึง การที่หน่วยประมวลผลหลายหน่วยสามารถเข้าถึงพื้นที่หน่วยความจำเดียวกันที่ใช้งานร่วมกันได้

2. โมเดลหน่วยความจำแบบกระจาย (Distributed Memory Model) หมายถึง การที่หน่วยประมวลผลหลายหน่วย ซึ่งแต่ละหน่วยจะมีพื้นที่หน่วยความจำของตนเอง และข้อมูลสามารถส่งผ่านหากันได้ระหว่างหน่วยประมวลผลเหล่านี้

โมเดลเธรด (Thread Model) เป็นหนึ่งในวิธีสำหรับเขียนโปรแกรมแบบขนาน ที่โปรเซสตัวหนึ่งสามารถสร้างโปรเซสอีกตัวหนึ่งขึ้นมาได้หลายตัว แต่ละตัวเรียกว่า “เธรด” และแต่ละเธรดทำงานแยกกันโดยไม่ขึ้นกับเธรดอื่นแม้ว่าเธรดทั้งหมดจะสามารถเข้าถึงพื้นที่หน่วยความจำที่ใช้

ร่วมกันได้ และเธรดสามารถถูกสร้าง รวมทั้งถูกยกเลิกได้ตามต้องการโดยโปรแกรมหลักหรือโปรเซสหลัก ตัวอย่างงานวิจัยที่ใช้โมเดลเธรด เช่น งานวิจัยของ Spiliotis, Bekakos & Boutalis (2020) ที่นำเสนอวิธีการแบบขนานสำหรับการแสดงภาพเป็นชุดของบล็อกสี่เหลี่ยมที่ไม่ทับซ้อนกันโดยใช้ OpenMP ซึ่งเป็น API ที่ได้รับความนิยมตัวหนึ่งสำหรับการเขียนโปรแกรมแบบขนานที่ใช้หน่วยความจำร่วมกัน โดยการแบ่งภาพออกเป็นบล็อกและกำหนดแต่ละบล็อกเป็นเธรดแยกกันประมวลผลพร้อมกัน และงานวิจัยของ (Sonuç & Özcan, 2023) ที่นำเสนออัลกอริทึมวิวัฒนาการคู่ขนานสำหรับแก้ปัญหาการเลือกทำเลสถานที่ตั้งที่ไม่มีข้อจำกัดด้านกำลังการผลิต (Uncapacitated Facility Location Problem: UFLP) และใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด เพื่อให้แต่ละเธรดแก้ปัญหาเริ่มต้นที่แตกต่างกันเพื่อให้ได้การกระจายที่ดีขึ้นเมื่อเริ่มต้นการค้นหา เป็นต้น

ตัวอย่างการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด โดยใช้ GCC ซึ่งเป็นคอมไพเลอร์ที่สนับสนุนภาษาโปรแกรมหลาย ๆ ภาษา เช่น ภาษา C ภาษา C++

```
#include <stdio.h>
#include <pthread.h>

// Define a function that will be executed in a separate thread
void* print_numbers(void* arg) {
int start = *((int*) arg);
for (int i = start; i <= 10; i++) {
printf("%d\n", i);
}
pthread_exit(NULL);
}
int main() {
// Create a thread object
pthread_t thread;
int arg = 1;

// Create the thread and pass in the argument
```

```
pthread_create(&thread, NULL, print_numbers, &arg);

// Wait for the thread to finish
pthread_join(thread, NULL);

printf("Done\n");

return 0;
}
```

จากตัวอย่างชุดคำสั่งสำหรับการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรดข้างต้น ผู้วิจัยจะขออธิบายการทำงานพอสังเขปดังนี้ จะเรียกใช้ไลบรารี ‘pthread’ ของภาษา C เพื่อสร้างเธรดใหม่ ซึ่งจะเป็นการเรียกใช้ฟังก์ชัน ‘print_numbers’ โดยที่ฟังก์ชันนี้รับอาร์กิวเมนต์หนึ่งตัวที่เป็นตัวชี้ไปยัง int ซึ่งแสดงหมายเลขเริ่มต้นสำหรับลำดับในโปรแกรม และสร้างอ็อปเจกต์เธรดโดยการประกาศตัวแปร ‘pthread_t’ และเรียกใช้ฟังก์ชัน ‘pthread_create’ ส่งผ่านตัวแปร ‘thread’ ‘NULL’ ‘printer_numbers’ และตัวชี้ไปยังตัวแปร ‘arg’ หลังจากเรียกใช้ฟังก์ชันนี้ก็จะมีการสร้างเธรดขึ้นมาทำงานตามฟังก์ชัน ‘print_numbers’ ตามลำดับ

เนื้อหาทั้งหมดในหัวข้อการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ซึ่งอธิบายเกี่ยวกับการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด เพื่อให้แอปพลิเคชันหรือโปรแกรมสามารถประมวลผลแบบขนานได้ โดยการแยกแอปพลิเคชันหรือโปรแกรมออกเป็นส่วน ๆ แต่ละส่วนมีความเป็นอิสระต่อกันเพื่อให้แต่ละโปรเซสเซอร์สามารถเรียกใช้ส่วนหนึ่งของโปรแกรมพร้อมกันได้จากโปรเซสเซอร์อื่น ปัจจุบันคอมพิวเตอร์สามารถมีซีพียูมากกว่าหนึ่งตัวและแต่ละตัวของซีพียูสามารถมีแกนประมวลผลหรือคอร์ได้หลายคอร์ช่วยกันทำงานหลายอย่างพร้อมกัน ทำให้นักพัฒนาสามารถพัฒนาแอปพลิเคชันหรือโปรแกรมแบบขนานเพื่อแบ่งภาระงานออกเป็นส่วนงานและผสานส่วนงานที่แบ่งออกกลับเข้าไปยังงานหลักได้

ดังนั้นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้องของโอเพนสแต็กในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจะดำเนินการออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์ใหม่ ด้วยประยุกต์ใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด โดยใช้โมเดลหน่วยความจำแบบกระจาย

การทดสอบการประมวลผลแบบกลุ่มเมฆ

Yao, Maleki Shoja & Tabrizi (2019) ได้กล่าวว่า การทดสอบ (Testing) การประมวลผลแบบกลุ่มเมฆหรือการทดสอบคลาวด์สามารถตีความได้ว่า 1) การทดสอบแอปพลิเคชันคลาวด์ ซึ่งเกี่ยวกับข้องกับการตรวจสอบสถานะของแอปพลิเคชันคลาวด์อย่างต่อเนื่องเพื่อให้เป็นไปตามข้อตกลงระดับการให้บริการ (Service Level Agreement: SLA) โดยข้อตกลงนี้เป็นข้อตกลงระหว่างผู้ให้บริการและผู้ใช้บริการว่าจะทำการรักษาระดับคุณภาพการให้บริการแก่ผู้ใช้บริการตามที่ได้ตกลงกันไว้ และ 2) การทดสอบ หมายถึง บริการคลาวด์ประเภทหนึ่งที่เกี่ยวข้องกับการใช้คลาวด์เป็นมิดเดิลแวร์ (Middleware) เพื่อดำเนินการจำลองการโต้ตอบของผู้ใช้งานตามเวลาจริง (Real-time) ของซอฟต์แวร์ที่พัฒนาขึ้น ในปริมาณมาก ๆ เช่น การทดสอบเว็บไซต์ โดยใช้คลาวด์เป็นมิดเดิลแวร์เพื่อจำลองการโต้ตอบของผู้ใช้งานตามเวลาจริง โดยการสร้างสถานการณ์ที่เหมือนจริงผ่านการใช้คลาวด์เพื่อจำลองการส่งคำร้องขอ (Request) และการตอบกลับ (Response) จากเว็บไซต์

นอกจากนี้ Yao et al. (2019) ยังอธิบายวิธีการในการทดสอบระบบคลาวด์ โดยแยกเป็นหมวดหมู่ ได้แก่ ประเภทของการทดสอบ (Type of Testing) การทดสอบบริการคลาวด์ (Testing Cloud Service) และวิธีการประเมินผล (Evaluation Method) โดยมีรายละเอียดดังต่อไปนี้

1. ประเภทของการทดสอบ

1.1 การทดสอบฟังก์ชัน (Functionality Testing) คือ การแสดงค่าความเบี่ยงเบนระหว่างฟังก์ชันที่ได้ออกแบบไว้และฟังก์ชันการทำงานจริงของซอฟต์แวร์

1.2 การทดสอบประสิทธิภาพ (Performance Testing) คือ การกำหนดการตอบสนองและประสิทธิผลของซอฟต์แวร์ด้วยการวัดเป็นตัวเลข

1.3 การทดสอบความยืดหยุ่นและความสามารถในการปรับขนาด (Elasticity and Scalability Testing) คือ การกำหนดความสามารถในการปรับตัวของคลาวด์เมื่อความต้องการทรัพยากรมีการเปลี่ยนแปลง

1.4 การทดสอบความปลอดภัย (Security Testing) คือ การแสดงการช่องโหว่ด้านความปลอดภัยที่ทำให้ซอฟต์แวร์เสี่ยงต่อการถูกโจมตี

1.5 การทดสอบอัตโนมัติ (Automatic Testing) คือ ระบบอัตโนมัติสำหรับการสร้างกรณีทดสอบ (Test Case) ตัวอย่างเช่น เฟรมเวิร์กที่เสนอโดย Wu & Lee (2012) โดยที่ข้อมูลเกี่ยวกับความหมายต่าง ๆ จะอยู่ในรูปแบบของ Web Service Description Language หรือ WSDL ซึ่งเป็นเอกสารที่เขียนในรูปแบบภาษา XML ที่ใช้ในการอธิบายการทำงานของเว็บเซอร์วิส (Web Service) ถูกสร้างจากคำสั่งในโปรแกรมที่เขียนด้วยภาษาคอมพิวเตอร์ เช่น C Java และคำอธิบายที่เกี่ยวกับคำสั่งในโปรแกรม หลังจากนั้นนำข้อมูลจาก WSDL มาสร้างกราฟลำดับเหตุการณ์ และสุดท้ายนำกราฟลำดับเหตุการณ์มาสร้างเป็นกรณีทดสอบ

2. การทดสอบบริการคลาวด์

2.1 การทดสอบสำหรับการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์หรือ IaaS คือ การทดสอบที่เกิดขึ้นในสภาพแวดล้อมของ IaaS หลาย ๆ แบบ (Cunha, Mendonca, & Sampaio, 2013) ตัวอย่างเช่น Cico & Dika (2014) แสดงการปรับใช้แอปพลิเคชันเดียวกันบนโครงสร้างพื้นฐานสองแบบ คือ 1) บนเครื่องคอมพิวเตอร์เซิร์ฟเวอร์กายภาพ (Physical Server) และ 2) บนเครื่องคอมพิวเตอร์เสมือน (Virtual Machine) และเปรียบเทียบประสิทธิภาพของแอปพลิเคชันภายใต้สภาวะที่มีภาระงาน (Workload) สูง ๆ หรืองานวิจัยของ Saghir & Masood (2019) ซึ่งวิเคราะห์และประเมินประสิทธิภาพของ OpenStack Neutron Network ซึ่งเป็นส่วนหนึ่งของโครงสร้างเครือข่ายของ IaaS คลาวด์ในโอเพนสแต็ก โดยใช้การสร้างสถานการณ์จำลองที่มีการใช้ทรัพยากรคอมพิวเตอร์เป็นพื้นฐาน ผลการทดสอบแสดงให้เห็นว่าความเร็วของเครือข่ายและประสิทธิภาพของ OpenStack Neutron Network สูงและเหมาะสมสำหรับการใช้งานที่มีการติดต่อกับเครือข่าย เป็นต้น

2.2 การทดสอบสำหรับการให้บริการแพลตฟอร์มหรือ PaaS คือ การทดสอบที่เกิดขึ้นในสภาพแวดล้อมของ PaaS ตัวอย่างเช่น วิธีการตามงานวิจัยของ Bucur, Kinder & Candea (2013); Zhou, Zhou & Li (2014) ถูกใช้สำหรับการทดสอบ PaaS คลาวด์ขององค์กรด้วยการสร้างสคริปต์ทดสอบโดยอัตโนมัติซึ่งทำให้กระบวนการทดสอบมีประสิทธิภาพมากขึ้น

2.3 การทดสอบสำหรับการให้บริการซอฟต์แวร์หรือ SaaS คือ การทดสอบที่เกิดขึ้นในสภาพแวดล้อมของ SaaS ตัวอย่างเช่น เฟรมเวิร์กที่ Cotroneo, Paudice & Pecchia (2016); Tsai, Huang & Shao (2011); Wu & Lee (2012) ได้เสนอไว้เพื่อจัดประเภทการแจ้งเตือนด้านความปลอดภัยใน SaaS คลาวด์ ซึ่งมีการสร้างกรณีทดสอบโดยอัตโนมัติอิงตามสถาปัตยกรรมเชิงบริการ เพื่อทดสอบความสามารถในการปรับขนาด (Scalability) ได้ของแอปพลิเคชันบน SaaS (Tsai et al., 2011)

3. วิธีการประเมินผล

Yao et al. (2019) ได้กล่าวว่ามีวิธีการประเมินหลายวิธี เช่น กรณีศึกษา (Case Study) การพัฒนาต้นแบบ (Prototype Development) การพิสูจน์อย่างเป็นทางการ (Formal Proof) การสำรวจ (Survey) และการศึกษาแบบอื่น ๆ สำหรับหัวข้อนี้จะกล่าวถึงการใช้กรณีศึกษาซึ่งมีนักวิจัยจำนวนมากนำมาใช้ประเมินประสิทธิภาพของวิธีการปฏิบัติงาน (Methodology) ของตนโดยการปรับใช้วิธีการปฏิบัติงานของตนในระบบคลาวด์ที่มีการใช้งานจริง

ดังนั้นการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจะใช้การทดสอบประสิทธิภาพ (Performance Testing) โดยการวัดการตอบสนองของระบบบันทึกเหตุการณ์ออกมาเป็นตัวเลขจำนวนครั้งของการตรวจสอบพบโปรเซสเป้าหมายหรือ threat-app ที่อยู่ในหน่วยความจำของเครื่องเสมือนหรือ domU สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยใช้โปรเซส threat-app เป็นกรณีศึกษาสำหรับการจำลองสถานการณ์ว่าผู้โจมตีเข้ามาควบคุมและใช้โปรเซสนี้เปิดอ่านไฟล์ข้อความที่มีความละเอียดอ่อนหรือมีความสำคัญของผู้ใช้บริการ

บทที่ 3

วิธีการดำเนินการวิจัย

จากเนื้อหาบทที่ 2 เอกสารและงานวิจัยที่เกี่ยวข้อง ซึ่งผู้วิจัยได้ทำการศึกษาแนวคิด ทฤษฎีที่เกี่ยวข้องเพื่อนำมาเป็นองค์ความรู้พื้นฐานของการวิจัยในวิทยานิพนธ์เล่นนี้ ในการปรับปรุงประสิทธิภาพของ IaaS OpenStack Logging System (IOLS) ซึ่งหัวข้อทั้งหมดสอดคล้องกับหัวข้อปัญหาวิจัยหลัก หัวข้อเป้าหมายของงานวิจัย และหัวข้อวัตถุประสงค์ของการศึกษา ที่จะสร้างต้นแบบระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

ในบทที่ 3 วิธีการดำเนินการวิจัย ถือว่าเป็นส่วนสำคัญอย่างยิ่งที่จะแสดงให้ผู้อ่านได้เข้าใจถึงสถาปัตยกรรม โครงสร้างภายใน และปัญหาของระบบบันทึกเหตุการณ์เมื่อนำมาประยุกต์ใช้งานใน IaaS คลาวด์ ที่สร้างจากโอเพนสแต็ก การทดลองสร้างระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยกรอบวิธีการดำเนินการวิจัยจะดำเนินการตามหัวข้อขอบเขตด้านวิธีการทดลอง โดยแบ่งเป็นหัวข้อย่อยดังต่อไปนี้

1. การสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมจำลองของห้องปฏิบัติการ
2. สภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
3. ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
4. ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
5. การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

6. การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

7. การวัดค่าความถูกต้องและทดสอบประสิทธิภาพจากค่าความถูกต้องและระยะเวลาของอัลกอริทึมที่ใช้สำหรับการตรวจสอบหน่วยความจำหลักของเครื่องเสมือนของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

การสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ โดย IaaS คลาวด์นี้จะจำลองบนเครื่องคอมพิวเตอร์จำนวน 1 เครื่องบนระบบปฏิบัติการ Ubuntu Server 16.04 LTS ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.1 และหัวข้อย่อย 2.2 เพื่อศึกษาองค์ประกอบ สถาปัตยกรรมและขั้นตอนการทำงานของ IaaS คลาวด์ รวมถึงการติดตั้งระบบบันทึกเหตุการณ์และส่วนประกอบต่าง ๆ ของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์ โดยมีรายละเอียดดังต่อไปนี้

1. การสร้าง IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สำหรับการสร้าง IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ โดยในการทดลองในวิทยานิพนธ์เล่มนี้จะจำลองสร้าง IaaS คลาวด์บนเครื่องคอมพิวเตอร์กายภาพจำนวนหนึ่งเครื่อง และใช้ระบบปฏิบัติการ Ubuntu Server 16.04 LTS โดยมีขั้นตอนการติดตั้งดังต่อไปนี้

1.1 ขั้นตอนที่ 1 ติดตั้งระบบปฏิบัติการ Ubuntu Server 16.04 LTS ซึ่งเป็นระบบปฏิบัติการตระกูลลินุกซ์บนเครื่องคอมพิวเตอร์กายภาพจำนวนหนึ่งเครื่องและกำหนดค่าเริ่มต้นสำหรับเชื่อมต่อเครื่องคอมพิวเตอร์กายภาพดังกล่าวกับระบบเครือข่าย

1.2 ขั้นตอนที่ 2 ติดตั้งซอฟต์แวร์ Xen เวอร์ชัน 4.6.5 สำหรับระบบปฏิบัติการ Ubuntu Server 16.04 LTS

1.3 ขั้นตอนที่ 3 ติดตั้งไลบารี LibVMI เวอร์ชัน 0.12-rc3 บน dom0 ของเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของเครื่องสเมือน (Virtual Machine: VM) หรือ domU

เมื่อดำเนินการทุกขั้นตอนเสร็จสิ้น ผู้วิจัยก็จะได้ IaaS คลาวด์สำหรับสภาพแวดล้องของห้องปฏิบัติการตามหัวข้อสถาปัตยกรรมของการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ที่เสนอโดย Wongthai (2014) ตามภาพ 1 คือ ฝ่ายผู้ให้บริการ (Provider Side) ที่อยู่ด้านซ้ายของภาพนี้

1.4 ขั้นตอนที่ 4 ติดตั้งซอฟต์แวร์ Virtual Machine Manager ซึ่งเป็นเครื่องมือสำหรับการบริการจัดการเครื่องเสมือนใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

2. การติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สำหรับการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการจากหัวข้อข้างบน ผู้วิจัยได้นำระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) ซึ่งได้รับอนุญาตให้สามารถการแก้ไขระบบบันทึกเหตุการณ์จากเจ้าของงานวิจัย มาปรับแต่งชุดคำสั่งบางส่วนเพื่อให้สอดคล้องกับการทำงานสำหรับระบบปฏิบัติการ Ubuntu Server 16.04 LTS โดยยังคงมีการกำหนดตำแหน่งของส่วนประกอบและฟังก์ชันการทำงานต่าง ๆ ของระบบบันทึกเหตุการณ์เหมือนต้นฉบับ โดยขั้นตอนการติดตั้งเพียงการสำเนา โปรแกรมระบบบันทึกเหตุการณ์ไปจัดเก็บไว้ในไดเรกทอรีที่กำหนดไว้ สำหรับการทดลองในวิทยานิพนธ์เล่มนี้กำหนดไดเรกทอรีสำหรับการจัดเก็บโปรแกรมดังกล่าวไว้ที่ root@compute1:/home/compute1# หลังจากนั้นผู้วิจัยก็จะได้ระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการตามหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ คือ กล่องสีเทาทั้งหมดในภาพ 3

การสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง วิธีการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อนำสภาพแวดล้อมนี้ไปใช้ในการทดลองเกี่ยวกับระบบบันทึกเหตุการณ์ในวิทยานิพนธ์เล่มนี้ ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.4 และหัวข้อย่อย 2.5 เพื่อศึกษาองค์ประกอบ ขั้นตอนการทำงาน และของสถาปัตยกรรมของโอเพนสแต็ก ซึ่งเป็นระบบปฏิบัติการคลาวด์ รวมถึงวิธีการแก้ไขปัญหาสำหรับการสร้างสำหรับการสร้าง IaaS คลาวด์โดยใช้ซอฟต์แวร์ OpenStack Ocata จากหัวข้อปัญหาวิจัยหลัก ซึ่งระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ได้จัดเตรียมและทดสอบใน IaaS คลาวด์จำลองของห้องปฏิบัติการ โดยใช้เครื่องคอมพิวเตอร์เพียงเครื่องเดียว แต่ยังไม่ปรากฏว่ามีการนำไปใช้ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) อย่างไรก็ตามเพื่อเป็นการนำงานวิจัยไปสู่การใช้ประโยชน์ได้จริง ประกอบกับสภาพแวดล้อมใน IaaS คลาวด์จำลองของห้องปฏิบัติการ และสภาพแวดล้อมใน IaaS คลาวด์ที่ใช้ในการผลิตจริงมีความแตกต่างกัน เช่น จำนวนของเครื่องคอมพิวเตอร์แม่งาน (Host Computer) หรือโฮสต์ ซึ่งเป็นที่อยู่ของเครื่องเสมือนหรือ VM ที่ในสภาพแวดล้อมที่ใช้ในการผลิตจริงสามารถมี

ได้เป็นจำนวนมากในระบบ เป็นต้น ซึ่งผู้วิจัยมีความเชื่อว่าสภาพแวดล้อมเหล่านี้มีผลกระทบต่อการทำงานของระบบบันทึกเหตุการณ์ดังกล่าวข้างต้น

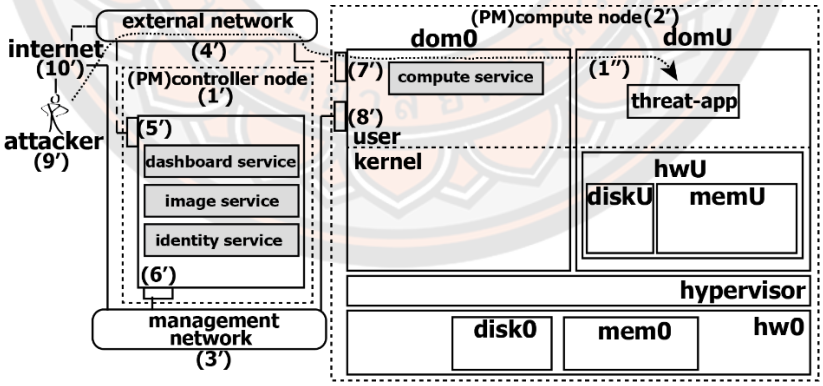
ดังนั้นผู้วิจัยจะจำลอง IaaS คลาวด์ที่ใช้ในการผลิตจริง โดยใช้ระบบปฏิบัติการคลาวด์ที่ชื่อว่า “โอเพนสแต็ก (OpenStack)” ซึ่งระบบปฏิบัติการคลาวด์นี้ถูกนำไปใช้ในการสร้างผลิตภัณฑ์จริงของภาคธุรกิจเพิ่มขึ้นเรื่อย ๆ (Gartner, 2022; OpenStack.org, 2022) โดยมีขั้นตอนการติดตั้ง IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็กดังต่อไปนี้

1. ขั้นตอนที่ 1 ติดตั้งระบบปฏิบัติการ Ubuntu Server 16.04 Long Term Support (LTS) ซึ่งเป็นระบบปฏิบัติการตระกูลลินุกซ์ที่ได้รับความนิยมตัวหนึ่งในการนำมาใช้งานบนเครื่องคอมพิวเตอร์กายภาพ (Physical Machine: PM) จำนวนสองเครื่อง และเชื่อมต่อเครื่องคอมพิวเตอร์กายภาพทั้งสองผ่านระบบเครือข่าย

2. ขั้นตอนที่ 2 ติดตั้งระบบปฏิบัติการ OpenStack Ocata บนเครื่องคอมพิวเตอร์กายภาพทั้งสองเครื่องตามคู่มือการติดตั้ง (OpenStack.org, 2017)

3. ขั้นตอนที่ 3 ติดตั้งซอฟต์แวร์ Xen เวอร์ชัน 4.6.5 สำหรับระบบปฏิบัติการ Ubuntu Server 16.04 LTS

4. ขั้นตอนที่ 4 ติดตั้งไลบารี LibVMI เวอร์ชัน 0.12-rc3 บน dom0 ของเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของ VM



ภาพ 5 สถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็ก

หมายเหตุ: มีการปรับเปลี่ยนองค์ประกอบ

ที่มา: Jaiboon, Wongthai, Phoka & Auxsorn (2020)

ในภาพ 5 ผู้วิจัยแสดงมุมมองของฝ่ายผู้ให้บริการของสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็ก ผู้วิจัยได้แมปแต่ละเซอร์วิสของสถาปัตยกรรมของโอเพนสแต็กในภาพ 2 เข้ากับคอมพิวเตอร์กายภาพสองเครื่องตามฟังก์ชันในคู่มือการติดตั้ง OpenStack.org (2017) โดยภาพ 5 มีการปรับเปลี่ยนองค์ประกอบบางส่วนจากงานวิจัยของ Jaiboon et al. (2020) เพื่อสื่อความหมายให้ตรงตามหัวข้อของงานวิจัยในวิทยานิพนธ์เล่มนี้ คือ เปลี่ยนชื่อโปรเซส เดิมคือ โปรเซส read เป็นใหม่ คือ โปรเซส thread-app ซึ่งทั้งสองโปรเซสจะทำหน้าที่เหมือนกัน และนำส่วนประกอบของระบบบันทึกเหตุการณ์ ได้แก่ logger LibVMI และ log file ออกเพื่อความชัดเจนในการแสดงสถาปัตยกรรมของ IaaS คลาวด์ ทั้งสถาปัตยกรรมในภาพ 5 ประกอบด้วยเครื่องคอมพิวเตอร์กายภาพ คือ โหนดคอมพิว (Compute Node) และโหนดคอนโทรเลอร์ (Controller Node) ดูที่กล่องเส้นประที่มีหมายเลข 1’ ทางด้านซ้ายของภาพ 5 เป็นโหนดคอนโทรเลอร์สำหรับดำเนินการของเซอร์วิสทั้งสามของภาพ 2 ได้แก่ ไอเด็นติตี้เซอร์วิส (Identity Service) อิมเมจเซอร์วิส (Image Service) และแดชบอร์ดเซอร์วิส (Dashboard Service) ซึ่งเซอร์วิสเหล่านี้ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของโอเพนสแต็ก ดูที่กล่องเส้นประที่มีหมายเลข 2’ ทางด้านขวาของภาพ 5 เป็นโหนดคอมพิวสำหรับดำเนินการของคอมพิวเซอร์วิสของภาพ 2 ซึ่งเซอร์วิสนี้ใช้เพื่อควบคุมไฮเปอร์ไวเซอร์หรือ Xen เพื่อจัดการ dom0 และ domU โหนดคอมพิวในสภาพแวดล้อมที่ใช้ในการผลิตจริงสามารถมีได้หลายโหนด สำหรับโหนดอ็อบเจกต์สตอเรสท์เซอร์วิสในภาพ 2 จะไม่เกี่ยวข้องกับการทดลองในวิทยานิพนธ์เล่มนี้

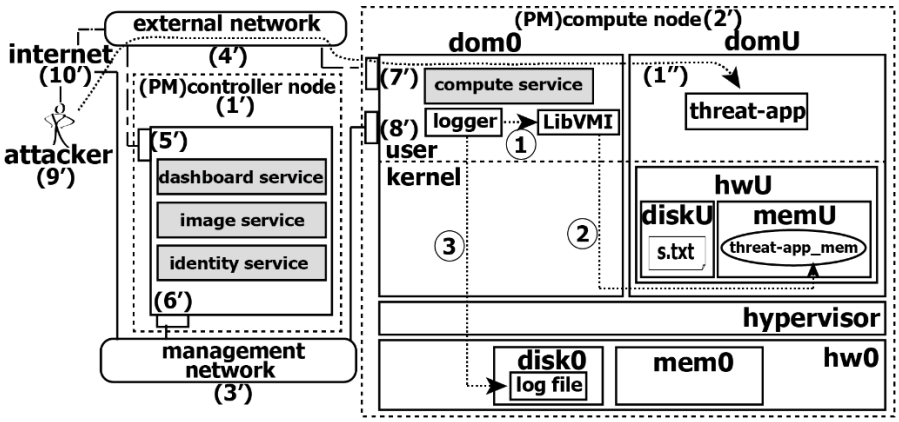
อย่างไรก็ตามเพื่อลดความซับซ้อนในการทดลอง จากสถาปัตยกรรมในภาพ 5 ผู้วิจัยจะใช้โหนดคอนโทรเลอร์และโหนดคอมพิวอย่างละหนึ่งโหนดเท่านั้น ซึ่งแต่ละโหนดจะใช้เน็ตเวิร์คอะแดปเตอร์ (Network Adapter) จำนวนสองตัวต่อโหนด (5’ ถึง 8’) เพื่อเชื่อมต่อเครือข่ายภายนอก (4’) และเครือข่ายการจัดการ (3’) ตามลำดับ ดูเส้นประที่มีหมายเลย 1’’ เครือข่ายภายนอกจะใช้สำหรับให้ผู้ใช้บริการเชื่อมต่อกับ domU ผ่านอินเทอร์เน็ตหรือ 10’ สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์นี้ ในการทดลองนี้เครือข่ายภายนอก (External Network) อาจจะเป็นช่องทางที่ผู้โจมตีหรือ 9’ สามารถใช้ช่องทางนี้เพื่อเชื่อมต่อกับ domU ด้วยข้อมูลประจำตัวและรหัสผ่านที่ผู้โจมตีอาจจะขโมยผ่านการทำฟิชชิงและการฉ้อโกงซึ่ง Wongthai (2014) ได้กล่าวไว้ สำหรับเครือข่ายการจัดการ (Management Network) จะใช้สำหรับการติดตั้งแอปพลิเคชันและอัปเดตแพตช์ด้านความปลอดภัยของระบบปฏิบัติการของโหนดคอนโทรเลอร์และโหนดคอมพิว สำหรับส่วนประกอบอื่นนอกเหนือจากที่ผู้วิจัยได้กล่าวถึง ถูกอธิบายแล้วในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ เมื่อผู้วิจัยได้

ติดตั้งเสร็จเรียบร้อยแล้วก็จะได้ระบบการประมวลผลแบบกลุ่มเมฆสาธารณะประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์สำหรับสภาพแวดล้อมของโอเพนสแต็ก หรือใช้คำย่อว่า “โอเพนสแต็ก IaaS คลาวด์” สำหรับการทดลองในวิทยานิพนธ์เล่นนี้

การสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.6 เพื่อติดตั้งและปรับปรุงระบบบันทึกเหตุการณ์ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ได้ ซึ่ง Wongthai (2014) ได้เสนอวิธีการบรรเทาความเสี่ยวเกี่ยวกับภัยคุกคามตาม CSA ใน IaaS คลาวด์โดยใช้ระบบบันทึกเหตุการณ์ และได้อธิบายว่าระบบบันทึกเหตุการณ์ใน IaaS คลาวด์นี้เป็นระบบในโครงสร้างพื้นฐานคลาวด์ของผู้ให้บริการที่จะรวบรวมและจัดเก็บข้อมูลประวัติของส่วนประกอบของโครงสร้างพื้นฐาน เช่น VM หรือ domU ระบบบันทึกเหตุการณ์นี้ประกอบด้วย ล็อกกิงโปรเซส (Logging Process) และล็อกไฟล์ (Log File) โดยรายละเอียดแต่ละส่วนประกอบและลำดับขั้นตอนการทำงานของระบบบันทึกเหตุการณ์นี้ได้ถูกอธิบายแล้วในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

ในภาพ 6 ให้พิจารณาเฉพาะส่วนของโหนดคอมพิวหรือกล่องสี่เหลี่ยมเส้นประที่อยู่ด้านขวาของภาพนี้ซึ่งเป็นเครื่องคอมพิวเตอร์กายภาพและติดตั้งไฮเปอร์ไวเซอร์ไว้แล้ว ระบบบันทึกเหตุการณ์อยู่ในโหนดคอมพิวสำหรับสภาพแวดล้อมของโอเพนสแต็ก ซึ่งในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ได้อธิบายเกี่ยวกับสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมห้องปฏิบัติการ และหัวข้อการสร้างสถาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ได้อธิบายเกี่ยวสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็กไปแล้ว ในหัวข้อนี้จะอธิบายส่วนประกอบเพิ่มเติมที่เกี่ยวข้องกับระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแตกใน IaaS คลาวด์ (IaaS OpenStack Logging System: IOLS) ซึ่งรายละเอียดแต่ละส่วนประกอบและขั้นตอนการทำงานของระบบบันทึกเหตุการณ์ในภาพ 6 จะเหมือนกับที่ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ



ภาพ 6 สถาปัตยกรรมของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

หมายเหตุ: มีการปรับเปลี่ยนองค์ประกอบ

ที่มา: Jaiboon et al. (2020)

ในภาพ 6 เป็นผลลัพธ์ที่ผู้วิจัยได้แมปสถาปัตยกรรมของระบบบันทึกเหตุการณ์ในหัวข้อสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ กับสถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็กในหัวข้อการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยส่วนที่มีการปรับเพิ่มเติมส่วนประกอบใหม่เข้าไปในภาพนี้ คือ ส่วนประกอบของระบบบันทึกเหตุการณ์ ได้แก่ logger LibVMI และ log file โดยผู้วิจัยขอสรุปดังนี้ ดูกล่องล็อกเกอร์ในระดับผู้ใช้ใน dom0 ซึ่งเป็น ล็อกเกอร์หรือล็อกกิงโปรเซสของระบบบันทึกเหตุการณ์ที่รันใน dom0 เพื่อเก็บข้อมูลที่ต้องการจะบันทึก ในภาพนี้ดูกล่องในระดับผู้ใช้ใน domU คือ โปรเซส thread-app ซึ่งในการทดลองผู้วิจัยสันนิษฐานว่าเป็นโปรเซสที่สามารถอ่านไฟล์ที่มีความสำคัญของผู้ใช้บริการ IaaS คลาวด์หรือ s.txt (กล่องที่อยู่ใน diskU) โดยปกติไฟล์นี้จะถูกจัดเก็บไว้ใน diskU ที่เป็นของผู้ใช้บริการหรือเจ้าของ domU ไฟล์นี้เป็นทรัพย์สินและมีค่าสำหรับองค์กรธุรกิจ (Wongthai, 2014) สำหรับคอมพิวเซอร์วิสซึ่งเป็นแอปพลิเคชันในระดับผู้ใช้ของ dom0 ที่ใช้ควบคุมไฮเปอร์ไวเซอร์ซึ่งไม่เกี่ยวข้องกับระบบบันทึกเหตุการณ์

การสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งสอดคล้องกับหัวข้อ

ขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.8 เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IaaS OpenStack Logging System: IOLS) จากหัวข้อการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ได้อธิบายว่า สถาปัตยกรรมของ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็ก โดยส่วนประกอบหลักอันหนึ่ง คือ โหนดคอมพิว (Compute Node) ซึ่งเป็นเครื่องคอมพิวเตอร์กายภาพที่มีซอฟต์แวร์ไฮเปอร์ไวเซอร์ Xen โดยซอฟต์แวร์นี้สามารถสร้างเครื่องเสมือน (VM) ได้มากกว่าหนึ่งเครื่องในเครื่องคอมพิวเตอร์กายภาพเครื่องเดียว ซึ่งในสถาปัตยกรรมนี้สามารถมีโหนดคอมพิวได้เป็นจำนวนมาก และระบบบันทึกเหตุการณ์ก็สามารถกระจายไปอยู่ในแต่ละโหนดคอมพิวเหล่านั้น ดังนั้นสามารถพิจารณาได้ว่าระบบบันทึก เหตุการณ์นี้เป็นสถาปัตยกรรมแบบกระจาย เช่นเดียวกับสถาปัตยกรรมของ Mishra, Verma & Gupta (2020) ซึ่งการกระจายนี้เองทำให้เกิดความยุ่งยากในการจัดการระบบบันทึกเหตุการณ์ที่อยู่ในแต่ละโหนดคอมพิว ดังนั้นผู้วิจัยจะปรับปรุงสถาปัตยกรรมของ IOLS ในภาพ 6 ให้สามารถดำเนินการแบบรวมศูนย์ได้ โดยใช้วิธีการเขียนโปรแกรมซ็อกเก็ต (Socket Programming Method) ซึ่งวิธีการนี้สามารถสร้างสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ได้และทำให้สามารถจัดการระบบบันทึกเหตุการณ์ในสถาปัตยกรรมนี้ได้ง่ายขึ้น

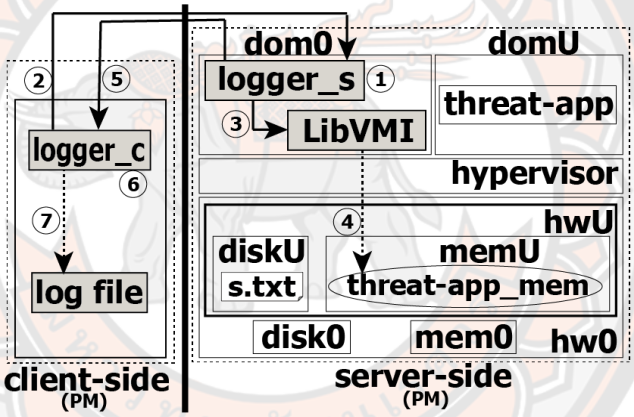
1. สถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ต

สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สามารถทำได้โดยใช้วิธีการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์ ซึ่งการทำสถาปัตยกรรมแบบรวมศูนย์นี้จำเป็นต้องแก้ไขระบบบันทึกเหตุการณ์ในภาพ 3 โดยรายละเอียดการแก้ไขจะกล่าวถึงในหัวข้อต่อไป

1.1 ส่วนประกอบและพารามิเตอร์ของสถาปัตยกรรมของระบบบันทึกเหตุการบนพื้นฐานการเขียนโปรแกรมซ็อกเก็ต

ผู้วิจัยจะออกแบบระบบบันทึกเหตุการณ์ในภาพ 3 ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ เพื่อให้รองรับสถาปัตยกรรมแบบรวมศูนย์ด้วยวิธีการเขียนโปรแกรมแบบซ็อกเก็ตและบนพื้นฐานของสถาปัตยกรรมในภาพ 2 และภาพ 6 ซึ่ง Maata et al. (2017) ได้กล่าวว่า วิธีการเขียนโปรแกรมซ็อกเก็ตหรือวิธีซ็อกเก็ตเป็นหนึ่งในวิธีที่ดีที่สุดในการทำการประมวลผลแบบกระจาย ภาพ 7 เป็นสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์ (ไม่ใช่ในโอเพนสแต็ก IaaS คลาวด์) ภาพนี้เป็นการพิสูจน์ว่าส่วนประกอบทั้งหมดของระบบบันทึก

เหตุการณ์ที่มีอยู่ (logger LibVMI และ log file) ในภาพ 3 สามารถแยกออกเป็นส่วนประกอบใหม่ได้ โดยที่ยังคงทำหน้าที่หลักของระบบบันทึกเหตุการณ์ได้เหมือนเดิม ซึ่งส่วนประกอบที่ได้ออกแบบใหม่ คือ 1) logger เดิมจากภาพ 3 ได้ออกแบบใหม่เป็น logger_c และ logger_s ในภาพ 7 และ 2) ทำการย้ายตำแหน่งของ log file เดิมจากภาพ 3 ซึ่งถูกกำหนดให้อยู่ใน disk0 ของ dom0 ได้ออกแบบใหม่โดยกำหนดตำแหน่งของ log file ให้อยู่ใน disk ของเครื่องคอมพิวเตอร์กายภาพเครื่องอื่นที่ติดตั้ง logger_c และปรับปรุงให้ logger_c และ logger_s สามารถส่งข้อมูลผ่านระบบเครือข่ายได้ ทำให้สถาปัตยกรรมของระบบบันทึกเหตุการณ์ในภาพ 7 สามารถใช้ logger_c ร้องขอการเชื่อมต่อไปยัง logger_s ที่อยู่ในเครื่องคอมพิวเตอร์กายภาพเครื่องอื่นผ่านระบบเครือข่ายได้ โดยรายละเอียดของแต่ละส่วนประกอบและขั้นตอนการทำงานของระบบบันทึกเหตุการณ์ที่ได้ออกแบบใหม่จะกล่าวถึงในย่อหน้าต่อไป



ภาพ 7 สถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์ (ไม่ใช่ในโอเพนสแต็ก IaaS คลาวด์)

สถาปัตยกรรมในภาพ 7 แบ่งออกเป็นสองฝั่งซึ่งแต่ละฝั่งหมายถึงเครื่องคอมพิวเตอร์กายภาพ (Physical Machine: PM) หนึ่งเครื่อง โดยฝั่งแรกจะเรียกว่า “server-side” และฝั่งที่สองเรียกว่า “client-side” ส่วนประกอบและฟังก์ชันการทำงานทั้งหมดบนฝั่ง server-side จะเหมือนกับส่วนประกอบในภาพ 3 ยกเว้น logger_s โดย logger_s ในภาพ 7 จะแตกต่างจาก logger ในภาพ 3 เพราะว่ามีการเพิ่มวิธีซ็อกเก็ตเข้าไปใน logger_s เพื่อช่วยให้ logge_s สามารถแลกเปลี่ยนข้อมูลระหว่างตัวมันเอง (ฝั่ง server-side) และ logger_c (ฝั่ง client-side) ผ่านระบบเครือข่าย และสำหรับฝั่ง client-side มีส่วนประกอบของ logger_c และ log file โดยที่ฝั่ง client-side และฝั่ง server-side สามารถสื่อสารผ่านเครือข่ายท้องถิ่น (Local Area Network: LAN) และ

ส่วนประกอบหลักของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์นี้ คือ กล่องสีเท่าทั้งหมดในภาพ 7 ได้แก่ logger_s log file logger_c และ LibVMI ซึ่งมีรายละเอียดดังต่อไปนี้

1. logger_s ถูกออกแบบมาเพื่อตรวจสอบโปรเซส threat-app ใน domU ซึ่งโปรเซสนี้เป็นการจำลองสถานการณ์ว่าเป็นภัยคุกคามใน domU และการทำงานของ logger_s จะเหมือนกับ logger ในภาพ 3 อย่างไรก็ตาม logger_s ยังถูกออกแบบให้รับการเชื่อมต่อจาก logger_c จากฝั่ง client-side เพื่อแลกเปลี่ยนข้อมูล เช่น ชื่อของ domU และชื่อของโปรเซสเป้าหมาย เช่น threat-app

2. logger_c ดูกล่องสีเทาด้านบนของฝั่ง client-side ในภาพ 7 ซึ่งถูกออกแบบมาเพื่อร้องขอการเชื่อมต่อกับ logger_s ในฝั่ง server-side

3. LibVMI และ log file ดูกล่องสีเท่าที่มีป้ายกำกับว่า “LibVMI” และ “log file” ตามลำดับ ส่วนประกอบทั้งสองมีฟังก์ชันตามภาพ 3 และถูกอธิบายไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

ตามที่กล่าวมาข้างต้นภาพ 7 เป็นสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์ (ไม่ใช่ในโอเพนสแต็ก IaaS คลาวด์) หรือสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์ใน IaaS คลาวด์

1.2 ขั้นตอนการทำงานของสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตใน IaaS คลาวด์ (ไม่ใช่ในโอเพนสแต็ก IaaS คลาวด์)

ขั้นตอนการทำงานของสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์ใน IaaS คลาวด์นี้ มีทั้งหมด 7 ขั้นตอน ดังต่อไปนี้

1. ขั้นตอนที่ 1 ทำการรัน logger_s ที่ dom0 แล้ว และ logger_s จะรอการร้องขอการเชื่อมต่อจาก logger_c

2. ขั้นตอนที่ 2 เมื่อ logger_c เชื่อมต่อกับ logger_s สำเร็จ จากนั้น logger_c จะส่งข้อความไปยัง logger_s ซึ่งข้อความจะประกอบด้วย 1) ชื่อของ domU ที่อยู่ในฝั่ง server-side เช่น du1 และ 2) ชื่อของโปรเซสเป้าหมายที่อยู่ใน domU เช่น threat-app

3. ขั้นตอนที่ 3 เมื่อ logger_s ได้รับข้อความจาก logger_c แล้วจะใช้ข้อมูลที่อยู่ในข้อความเพื่อตั้งค่าพารามิเตอร์เริ่มต้นสำหรับการเรียกใช้ LibVMI

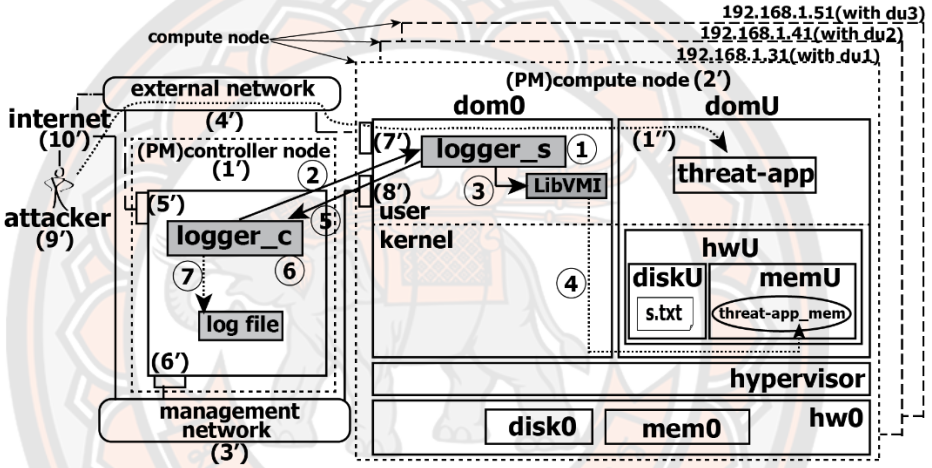
4. ขั้นตอนที่ 4 เมื่อ LibVMI ดึงข้อมูลที่ต้องการจะบันทึกจาก memU ของ domU ตามค่าพารามิเตอร์ที่กำหนด

5. ขั้นตอนที่ 5 หลังจากที่ logger_s ได้รับข้อมูลที่ต้องการจะบันทึกจาก LibVMI แล้วจากนั้น logger_s จะส่งข้อมูลเหล่านี้กลับไปยัง logger_c ในรูปแบบของข้อความ

6. ขั้นตอนที่ 6 logger_c ได้รับข้อความจาก logger_s ซึ่งเป็นผลลัพธ์จากขั้นตอนที่ 5

7. ขั้นตอนที่ 7 logger_c จะทำการประมวลผลขอมูลก่อนที่จะเขียนข้อมูลเหล่านี้ไปยังล็อกไฟล์

2. สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์



ภาพ 8 สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ภาพ 8 ที่ไม่มีกล่องสีเทาทั้งหมด คือ โอเพนสแต็ก IaaS คลาวด์ ที่มีการขยายโหนดคอมพิว ซึ่งสถาปัตยกรรมนี้มีถูกนำไปใช้ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) และ OpenStack.org (2015) ได้แสดงให้เห็นการขยายโหนดคอมพิวจำนวน 1,000 โหนดในสถาปัตยกรรมของโอเพนสแต็ก IaaS คลาวด์ สำหรับการขยายโหนดคอมพิวนี้ สามารถทำได้โดยการเพิ่มโหนดคอมพิวใหม่เข้าไปในสถาปัตยกรรมของโอเพนสแต็ก IaaS คลาวด์หรือภาพ 6 ซึ่งโหนดคอมพิวใหม่ที่เพิ่มเข้าไปทั้งหมดจะมีรายละเอียดเหมือนกับโหนดคอมพิวเดิมที่มีอยู่ในสถาปัตยกรรมของโอเพนสแต็ก IaaS คลาวด์ โดยโหนดคอมพิวใหม่สามารถตั้งค่าตามคู่มือการติดตั้งโอเพนสแต็กใน OpenStack.org (2017) ดังนั้นอาจมีล็อกเกอร์ (Logger) มากกว่าหนึ่งตัวในสถาปัตยกรรมของโอเพนสแต็ก IaaS คลาวด์

และการเรียกใช้งานล็อกเกอร์ที่อยู่ในแต่ละโหนดคอมพิวก็จะจัดการได้ยาก เพราะฉะนั้นผู้วิจัยจะเรียกระบบบันทึกเหตุการณ์ในภาพ 6 เป็นระบบบันทึกเหตุการณ์แบบกระจายศูนย์ ด้วยเหตุนี้ผู้วิจัยจึงได้ออกแบบสถาปัตยกรรมแบบรวมศูนย์ของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็ก IaaS คลาวด์ หรือเรียกว่า สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็ก IaaS คลาวด์

สำหรับการทดลองนี้ผู้วิจัยจะเพิ่มโหนดคอมพิวจำนวนสองโหนดเพิ่มเติมในสถาปัตยกรรมโอเพนสแต็ก IaaS คลาวด์หรือภาพ 6 ดังนั้นทางด้านขวาของภาพ 8 โหนดคอมพิว (Compute Node) ที่ขยาย คือ กล่องเส้นประที่มีไอพีแอดเดรส (IP Address) ลงท้ายด้วย 41 และ 51 ดังนั้นระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้องของโอเพนสแต็กใน IaaS คลาวด์ (IOLS) ที่มีการขยายโหนดคอมพิว คือ กล่องสีเทาทั้งหมดในภาพ 8 ระบบบันทึกเหตุการณ์นี้มีส่วนประกอบหลักและตำแหน่งของส่วนประกอบเหล่านี้เหมือนกับภาพ 7 ดูกล่องเส้นประที่มีหมายเลข 1’ ทางด้านซ้ายของภาพ 8 คือ โหนดคอนโทรเลอร์ (Controller Node) ซึ่งเป็นเครื่องคอมพิวเตอร์กายภาพในโอเพนสแต็ก IaaS คลาวด์ ผู้วิจัยแมปส่วนประกอบทั้งสองของสถาปัตยกรรมของระบบบันทึกเหตุการณ์ในภาพ 7 ได้แก่ logger_c และ log file ลงในโหนดคอนโทรเลอร์ในภาพ 8 จากนั้นแมปส่วนประกอบและตำแหน่งของ logger_s และ LibVMI ในภาพ 7 ลงในโหนดคอมพิว ดูกล่องเส้นประที่มีหมายเลข 2’ ทางด้านขวาของภาพ 8 นอกจากนี้โหนดคอมพิวที่ขยายแต่ละโหนดในภาพ 8 มีส่วนประกอบของ logger_s และ LibVMI รวมถึงตำแหน่งของส่วนประกอบเหล่านี้อยู่ในแต่ละโหนดคอมพิวที่ขยายเหมือนกับ logger_s และ LibVMI ในกล่องเส้นประที่มีหมายเลข 2’ ทางด้านขวาของภาพ 8 รายละเอียดขององค์ประกอบหลักของสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน Iaas คลาวด์ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ต และการขยายโหนดคอมพิวได้อธิบายไว้ในย่อหน้าข้างบน ตอนนี้ผู้วิจัยได้สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้องของโอเพนสแต็กใน IaaS คลาวด์ที่มีการขยายโหนดคอมพิว

การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการวิจัย หัวข้อย่อย 2.10 เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งได้ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์โดยใช้วิธีการเขียนโปรแกรมซ็อกเก็ต เพื่อให้สอดคล้องกับการทำงานในสภาพแวดล้อมของโอเพนสแต็ก โดยระบบบันทึกเหตุการณ์จากสถาปัตยกรรมนี้ยังคงสามารถทำงานได้ตามวัตถุประสงค์

อย่างไรก็ตามเนื่องจากสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์หรือภาพ 8 ที่ประยุกต์ใช้การเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์โดยใช้โพรโตคอล TCP ซึ่งผู้วิจัยเลือกใช้นั้นจำเป็นต้องทราบหมายเลขซ็อกเก็ตของ logger_s ใน dom0 ของโหนดคอมพิว (Compute Node) ก่อนถึงจะสามารถเรียกใช้ logger_c ที่อยู่ในโหนดคอนโทรเลอร์ (Controller Node) ให้ร้องขอการเชื่อมต่อไปยัง logger_s ใน dom0 ของโหนดคอมพิวได้ ดังนั้นการทำงานของระบบบันทึกเหตุการณ์ในภาพ 8 logger_s จำเป็นต้องระบุหมายเลขซ็อกเก็ตเมื่อเริ่มต้นทำงานเสมอ ตัวอย่างคำสั่งสำหรับรัน logger_s เช่น ./logger_s 1456 โดยที่ ./logger_s หมายถึง การอ้างถึงตำแหน่งของไฟล์และชื่อของโปรแกรม logger_s และ 1456 หมายถึง หมายเลขซ็อกเก็ตบนระบบปฏิบัติการ Ubuntu Server 16.04 LTS บนเครื่องคอมพิวเตอร์กายภาพเครื่องเดียวกัน จากภาพ 8 หากต้องการรัน logger_s จำนวนสองตัวบนโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย 31 จำเป็นต้องระบุหมายเลขซ็อกเก็ตแตกต่างกัน ตัวอย่างคำสั่งสำหรับรัน logger_s จำนวนสองตัว เช่น ./logger_s 1456 และ ./logger_s 5678 ดูภาพ 9 และ ภาพ 10 ตามลำดับ ภาพทั้งสองจะมีความแตกต่างกันตรงส่วนของกล่องที่มีป้าย “b” (1456 และ 5678) จะเห็นได้ว่าหากต้องการรัน logger_s จำนวนมากในโหนดคอมพิวเดียวกันเป็นเรื่องที่ยุ่งยากสำหรับการจัดการ

```
                    a       b
⟨1⟩root@compute1:/home# ./logger_s 1456
     .....Waiting for connection.....
```

ภาพ 9 รูปแบบคำสั่งของ logger_s ใน dom0 ของโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย 31 ครั้งที่ 1

```
                    a       b
⟨1⟩root@compute1:/home# ./logger_s 5678
     .....Waiting for connection.....
```

ภาพ 10 รูปแบบคำสั่งของ logger_s ใน dom0 ของโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย 31 ครั้งที่ 2

ดังนั้นการประยุกต์ใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด สำหรับระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์หรือภาพ 8 สามารถแก้ไขปัญหาที่กล่าวข้างต้นได้ และยังช่วยให้ logger_s ทำงานแบบขนานกันได้ นั่นคือระบบบันทึกเหตุการณ์นี้สามารถตรวจสอบ domU ได้มากกว่าหนึ่งตัวพร้อมกันในเวลาเดียวกันได้ ง่ายกว่าระบบบันทึกเหตุการณ์ในหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

การประยุกต์ใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด สำหรับระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์หรือภาพ 8 ทำให้การรัน logger_s จำนวนสองตัวหรือมากกว่าบนโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย 31 สามารถทำได้ด้วยการรัน logger_s และระบุหมายเลขซ็อกเก็ตเพียงครั้งเดียว ตัวอย่างคำสั่งสำหรับรัน logger_s ในสถาปัตยกรรมนี้ คือ ./logger_s 1456 หรือภาพ 9 หลังจากนั้นก็สามารถใช้ logger_c ที่อยู่ในโหนดคอนโทรเลอร์ให้ร้องขอการเชื่อมต่อไปยัง logger_s ใน dom0 ของโหนดคอมพิวได้ที่มีไอพีแอดเดรสลงท้ายด้วย 31 โดยเมื่อ logger_s และ logger_c เชื่อมต่อกันสำเร็จโมเดลเธรดจะทำการแตกเป็นโปรเซสเธรดย่อยและกำหนดหมายเลขซ็อกเก็ตใหม่ เพื่อสร้างการเชื่อมต่อระหว่าง logger_c กับโปรเซสเธรดย่อยที่แตกออกมา ลักษณะแบบนี้จะทำให้หมายเลขซ็อกเก็ต 1456 สามารถนำกลับมาใช้รอรับการเชื่อมต่อจาก logger_c ตัวต่อไปได้ ดูภาพ 11 และภาพ 12 ตามลำดับ โดยภาพทั้งสองนี้จะมีความแตกต่างกันตรงส่วนของกล่องที่มีป้าย “d” (du1 และ du2)

```
                    a          b          c    d       e
①root@client:/home# ./logger_c 192.168.1.31 1456 du1 threat-app
     .....Waiting for answer.....
```

ภาพ 11 รูปแบบคำสั่งของ logger_c **ในโหนดคอนโทรเลอร์ร้องขอการเชื่อมต่อไปยัง** logger_s **ในโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย** 31 **ครั้งที่** 1

```
                    a          b          c    d       e
①root@client:/home# ./logger_c 192.168.1.31 1456 du2 threat-app
     .....Waiting for answer.....
```

ภาพ 12 รูปแบบคำสั่งของ logger_c **ในโหนดคอนโทรเลอร์ร้องขอการเชื่อมต่อไปยัง** logger_s **ในโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย** 31 **ครั้งที่** 2

การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS **คลาวด์บนพื้นฐานตารางแฮช**

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.12 เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ซึ่งเป็นการปรับปรุงอัลกอริทึมสำหรับการตรวจสอบหน่วยความจำของเครื่องเสมือนหรือ VM โดยการประยุกต์ใช้ตารางแฮช (Hash Table) และการจัดเรียงชุดคำสั่งใหม่ จากการวิเคราะห์ชุดคำสั่งของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ผู้วิจัยพบว่าการทำงานของระบบบันทึกเหตุการณ์นี้ส่วนใหญ่เกิดขึ้นที่ส่วนประกอบของ logger_s เป็นหลัก ซึ่งเป็นการดำเนินการเกี่ยวกับการตรวจสอบข้อมูลในหน่วยความจำเสมือนของ domU โดยการเรียกใช้ไลบารี LibVMI เพื่อดึงข้อมูลที่ต้องการมาตรวจสอบ ดังนั้นผู้วิจัยเชื่อว่าหากสามารถลดระยะเวลาในการตรวจสอบนี้ได้ ก็อาจจะสามารถเพิ่มประสิทธิภาพของระบบบันทึกเหตุการณ์โดยรวมได้

ผลลัพธ์จากการวิเคราะห์ชุดคำสั่งของในส่วนประกอบของ logger_s ผู้วิจัยพบว่าในแต่ละรอบของการวนตรวจสอบข้อมูลในหน่วยความจำเสมือนของ domU โดย logger_s จะมีการเปรียบเทียบว่าข้อมูลที่ได้รับมาจากการเรียกใช้ไลบารี LibVMI เช่น ชื่อของโปรเซส ว่าเป็นโปรเซสเป้าหมายในการตรวจสอบหรือไม่ ผ่านคำสั่งของภาษา C ซึ่งเป็นภาษาสำหรับการพัฒนาระบบ

บันทึกเหตุการณ์นี้ คือ ฟังก์ชัน strcmp ซึ่งฟังก์ชันนี้เป็นการเปรียบเทียบสตริง (String) จำนวนสองชุดแบบคำนึงถึงขนาดตัวพิมพ์ของสตริง (Case-sensitive) และคืนค่าผลลัพธ์ของการเปรียบเทียบดังต่อไปนี้

1. น้อยกว่า 0 หมายถึง ค่าของสตริงชุดที่ 1 น้อยกว่าค่าของสตริงชุดที่ 2
2. เท่ากับ 0 หมายถึง ค่าของสตริงชุดที่ 1 เท่ากับค่าของสตริงชุดที่ 2
3. มากกว่า 0 หมายถึง ค่าของสตริงชุดที่ 1 มากกว่าค่าของสตริงชุดที่ 2

โดยการทำงานของฟังก์ชัน strcmp นั้นจะเป็นการเปรียบเทียบของสตริงจำนวนสองชุดทีละตัวอักษรตามลำดับ เช่น สตริงชุดที่ 1 มีค่า “Nareasuan” และ สตริงชุดที่ 2 มีค่า “Nareasuan”

สำหรับการเปรียบเทียบประสิทธิภาพของอัลกอริทึมที่แตกต่างกันและเลือกสิ่งที่ดีที่สุดในการแก้ปัญหาเฉพาะ จะพิจารณาจากเวลา (Time) และพื้นที่ (Space) ที่เป็นปัจจัยสำคัญส่วนใหญ่ ความซับซ้อนเชิงพื้นที่ (Space Complexity) ของอัลกอริทึมจะจากปริมาณของพื้นที่หน่วยความจำที่อัลกอริทึมใช้ในการทำงาน ในทำนองเดียวกันความซับซ้อนเชิงเวลา (Time Complexity) ของอัลกอริทึมจะวัดจากระยะเวลาที่อัลกอริทึมใช้ในการทำงาน (Phalke, Vaidya, & Metkar, 2022) ผลลัพธ์จากการวิเคราะห์เกี่ยวกับความซับซ้อนของอัลกอริทึมจะออกมาในรูปแบบการประมาณการเชิงสัญลักษณ์หรือเรียกว่า สัญกรณ์ (Notation) ได้แก่ โอใหญ่ (Big-O) โอเมก้าใหญ่ (Big-Ω) และ ทีต้าใหญ่ (Big-Θ) และสามารทำการเปรียบเทียบประสิทธิภาพได้ 2 กรณีที่ได้รับความนิยม คือ 1) ประสิทธิภาพกรณีแย่ที่สุด และ 2) ประสิทธิภาพกรณีถัวเฉลี่ย โดยการทดลองในวิทยานิพนธ์เล่มนี้จะให้ความสนใจในเรื่องของเวลาหรือความซับซ้นเชิงเวลา และประสิทธิภาพกรณีแย่ที่สุด โดยขอกล่าวถึงสัญกรณ์เพียงตัวเดียว คือ โอใหญ่ ซึ่งจะบ่งบอกถึงระยะเวลาในการประมวลผลที่มากที่สุดที่อัลกอริทึมใช้ในการทำงาน (Alsuwaiyel, 2021) ในการวิเคราะห์ความซับซ้อนของอัลกอริทึมที่เสนอ เมื่อวิเคราะห์ความซับซ้อนเชิงเวลาของฟังก์ชัน strcmp จะมีความซับซ้อนเชิงเวลาเท่ากับ O(n) โดยที่ n คือ ความยาวของสตริงที่สั้นที่สุดระหว่างสตริงชุดที่ 1 และ สตริงชุดที่ 2 เมื่อเทียบกับความซับซ้อนเชิงเวลาการใช้ตารางแฮช (Hash Table) จะมีความซับซ้อนเชิงเวลาเท่ากับ O(1) กรณีการค้นหาข้อมูลที่ต้องการว่ามีอยู่หรือไม่

ดังนั้นผู้วิจัยจะเลือกใช้ตารางแฮช ซึ่งเป็นโครงสร้างข้อมูลประเภทหนึ่งที่มีความสามารถในการแมปคีย์กับค่าข้อมูลต่าง ๆ โดยที่ตารางแฮชจะประกอบด้วยสองส่วน คือ 1) ส่วนที่ใช้สำหรับจัดเก็บข้อมูลอาจจะใช้อาร์เรย์ และ 2) ส่วนฟังก์ชันแฮช ซึ่งเป็นสิ่งที่จะช่วยให้ตัดสินใจว่าข้อมูลที่ป้อนเข้ามาจะถูกจัดเก็บไว้ที่ใดในหน่วยความจำของเครื่องคอมพิวเตอร์ เมื่อวิเคราะห์ความซับซ้อนเชิงเวลาการทำงานของตารางแฮท จะใช้ความซับซ้อนเชิงเวลาโดยเฉลี่ยเท่ากับ O(1)

(Shah & Shaikh, 2016) ซึ่งเมื่อเทียบกับการทำงานของฟังก์ชัน strcmp แล้วตารางแฮชอาจจะสามารถทำงานได้เร็วกว่า ดังนั้นการนำตารางแฮชมาประยุกต์ใช้ในการทำงานของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ผู้วิจัยเชื่อว่าอาจจะทำให้ประสิทธิภาพโดยรวมของระบบบันทึกเหตุการณ์นี้เพิ่มขึ้น

Shah & Shaikh (2016) ยังกล่าวอีกว่าการเลือกใช้ฟังก์ชันแฮชให้เหมาะสมสำหรับการแมปคีย์กับค่าสตริงเป็นสิ่งที่จำเป็น ทั้งนี้ฟังก์ชันแฮชที่ดีควรทำงานได้รวดเร็ว และควรมีการชนกัน (Collision) น้อยที่สุด ซึ่งการชนกันนี้ หมายถึง เหตุการณ์ที่สตริงสองสตริง เมื่อผ่านฟังก์ชันแฮชแล้วได้ค่าตำแหน่งเดียวกันในตารางแฮช ดังนั้นผู้วิจัยจะขอสรุปฟังก์ชันแฮชยอดนิยมสองฟังก์ชันที่ได้กล่าวไว้ใน Ram, Ranjan, Chakrabarti & Samanta (2015) และหนึ่งในสองฟังก์ชันจะถูกเลือกใช้สำหรับการวิจัยในวิทยานิพนธ์เล่มนี้ ได้แก่ 1) ฟังก์ชันแฮช SDBM สามารถแสดงโดยสมการทางคณิตศาสตร์ตามภาพ 13 มีค่า hash[0] เท่ากับ 0 และ *s* คือ สตริงของอักขระที่ใช้แทนคำทั่วไป ฟังก์ชันแฮช SDBM โดยรวมแล้วสามารถให้การกระจายของข้อมูลที่ดีและมีการแบ่งที่น้อยลง อย่างไรก็ตามฟังกชันแฮชนี้มีข้อด้อยเรื่องความเร็วที่ต่ำในการทำงาน และ 2) ฟังก์ชันแฮช DJB2 ที่พัฒนาโดย Dan Berstein สามารถ

$$hash[i] = hash[i-1] * 65599 + s[i] ; \qquad (when\ i > 0)$$

ภาพ 13 สมการทางคณิตศาสตร์ของฟังก์ชันแฮช SDBM

ที่มา: Shah & Shaikh (2016)

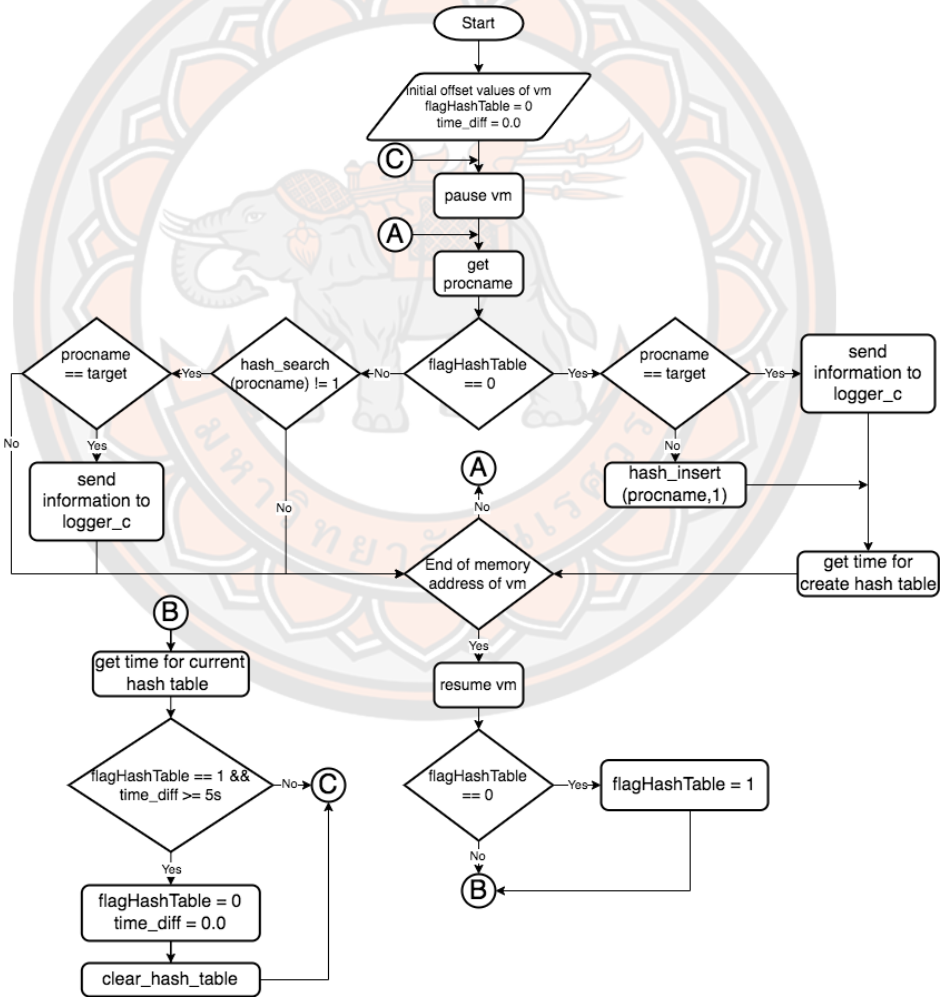
$$hash[i] = hash[i-1] * 33 + s[i] ; \qquad (when\ i > 0)$$

ภาพ 14 สมการทางคณิตศาสตร์ของฟังก์ชันแฮช DJB2

ที่มา: Shah & Shaikh (2016)

แสดงโดยสมการทางคณิตศาสตร์ตามภาพ 14 มีค่า hash[0] เท่ากับ 5381 และ *s* คือ สตริงของอักขระที่ใช้แทนคำทั่วไป ฟังก์ชันแฮช DJB2 ไม่เพียงแต่ให้การกระจายของข้อมูลที่ยอดเยี่ยมแล้วยังทำงานได้เร็ว สำหรับกลุ่มของคีย์ที่แตกต่างกัน รวมถึงขนาดของตารางที่แตกต่างกันด้วย

ใน Ram et al. (2015) ได้สรุปว่า DJB2 เป็นฟังก์ชันแฮชที่ดี เนื่องจากให้ความสมดุลที่ดีระหว่างความเร็วในการประมวลผลและการกระจายของข้อมูลโดยรวม ดังนั้นผู้วิจัยจึงเลือกใช้ฟังก์ชันแฮช DJB2 สำหรับปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ภาพ 15 เป็นบางส่วนของผังงานของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ซึ่งผู้วิจัยเสนอเฉพาะส่วนของการตรวจสอบหน่วยความจำของเครื่องเสมือนหรือ VM หลังจากประยุกต์ใช้ตารางแฮชเพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์นี้



ภาพ 15 กระบวนการทำงานของระบบบันทึกเหตุการณ์แบบรวมศูนย์ เฉพาะส่วนของการตรวจสอบหน่วยความจำหลักของเครื่องเสมือนเมื่อประยุกต์ใช้ตารางแฮช

การวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับรายละเอียดของเนื้อหาในหัวข้อนี้จะกล่าวถึง การวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.12 เพื่อแสดงให้เห็นถึงประสิทธิภาพของระบบบันทึกเหตุการณ์ในด้านการตรวจจับและบันทึกข้อมูลของโปรเซสเป้าหมายที่ต้องการตรวจสอบและความเร็วของอัลกอริทึมที่ใช้สำหรับการตรวจสอบหน่วยความจำของเครื่องเสมือน ทั้งนี้ Molyneaux (2014) ได้กล่าวถึง วงจรการพัฒนาซอฟต์แวร์ในหนังสือเรื่อง “The Art of application Performance Testing” ที่แอปพลิเคชันที่สร้างใหม่หรือถูกออกแบบใหม่จำเป็นต้องได้รับการทดสอบประสิทธิภาพก่อนที่จะนำไปใช้งานจริงเสมอ โดยในวิทยานิพนธ์เล่มนี้จะทำการวัดค่าความถูกต้อง (Accuracy) และทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ตามแนวทางการวิจัยของ Auxsorn et al. (2020); Wiriya et al. (2020); Wongthai (2014) ด้วยการวัดค่าความถูกต้องในการทำงานของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งการวัดค่าความถูกต้องจะมุ่งเน้นไปที่องค์ประกอบหลักของระบบบันทึกเหตุการณ์ คือ logger และ logger_s ของแต่ละสถาปัตยกรรมของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด และ หัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช โดยในหัวข้อนี้จะใช้คำว่า ล็อกเกอร์ แทนซึ่งจะหมายถึง logger หรือ logger_s ของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อดังกล่าวข้างต้น โดยมีรายละเอียดการวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์เหล่านี้ ดังนี้

1. การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์

ผู้วิจัยจะจำลองแอปพลิเคชันชื่อ threat-app ซึ่งแอปพลิเคชันหรือโปรเซสนี้อาจจะถูกควบคุมหรือใช้งานโดยผู้โจมตีเพื่อเข้าถึงไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ใน diskU ของ domU ดูกล่องใน domU ในภาพ 3 ขั้นตอนการดำเนินงานของโปรเซส threat-app เป็นดังต่อไปนี้

ขั้นตอนที่ 1 เปิดไฟล์ s.txt ดูรูปร่างของไฟล์ข้อความใน diskU ในรูปภาพ 3

ขั้นตอนที่ 2 อ่านและแสดงเนื้อหาของไฟล์

ขั้นตอนที่ 3 ปิดไฟล์

ขั้นตอนที่ 4 โปรเซสสิ้นสุดการทำงาน

การวัดค่าความถูกต้องของล็อกเกอร์นั้นจะสัมพันธ์กับ Sleeping Time ของโปรเซส threat-app โดยที่ Sleeping Time หรือ SPT หมายถึงค่าของเวลาที่เพิ่มให้กับการทำงานของโปรเซส threat-app หลังจากขั้นตอนที่ 2 ก่อนขั้นตอนที่ 3 ในขั้นตอนการดำเนินงานของโปรเซส threat-app และ SPT นี้มีหน่วยเป็นมิลลิวินาที (Millisecond: ms) ตัวอย่างเช่น SPT คือ 65ms หมายความว่าหลังจากขั้นตอนที่ 2 ของโปรเซส threat-app จะไม่ได้ทำงานหรืออยู่ในโหมด sleep เป็นระยะเวลา 65ms หลังจากนั้นโปรเซสนี้จะปิดไฟล์และสิ้นสุดการดำเนินงาน ทั้งนี้ค่าของ SPT สามารถปรับเปลี่ยนได้ซึ่งการปรับเปลี่ยนจะส่งผลต่อค่าความถูกต้องของล็อกเกอร์ซึ่งได้กล่าวไว้ในงานวิจัยของ Auxsorn et al. (2020); Wiriya et al. (2020); Wongthai (2014) โดยในการวิจัยนี้จะมีการปรับค่า SPT ของโปรเซส threat-app เริ่มตั้งแต่ 0ms ถึง 10ms และเพิ่มขึ้นครั้งละหนึ่งหนึ่งหน่วยตามลำดับ

ในการวัดค่าความถูกต้องของล็อกเกอร์ในวิทยนิพนธ์เล่มนี้ ผู้วิจัยจะกำหนดค่าเริ่มต้น SPT ของโปรเซส threat-app เช่น SPT มีกำหนดไว้ที่ 65ms และจะรันล็อกเกอร์เพียงครั้งเดียวก่อนการรันโปรเซส threat-app เสมอ หลังจากนั้นล็อกเกอร์จะตรวจสอบหาชื่อโปรเซสของโปรเซส threat-app ในหน่วยความจำเสมือนของ domU ในแต่ละครั้งจากจำนวนทั้งหมด 10,000 ครั้ง ดังนั้นค่าความถูกต้องของล็อกเกอร์จะถูกวัดโดยจำนวนของความถูกต้องเมื่อล็อกเกอร์ตรวจพบชื่อโปรเซสหรือสตริง “threat-app” ในแต่ละครั้งจากจำนวนทั้งหมด 10,000 ครั้ง ซึ่งสตริง “threat-app” จะถูกดักจับจาก threat-app_mem ดูกล่องใน memU ในภาพ 3 ถ้าถูกต้องจะเรียกว่า 1 “hit” หรือไม่ถูกต้องจะเรียกว่า 1 “miss” ดังนั้นหากโปรเซส threat-app เข้าถึงไฟล์ s.txt จำนวน 10,000 ครั้งและล็อกเกอร์สามารถตรวจสอบพบสตริง “threat-app” ได้จำนวน 10,000 ครั้งหรือ 10,000 hit ผู้วิจัยจะแปลความหมายว่าล็อกเกอร์มีค่าความถูกต้อง 100% เมื่อโปรเซส threat-app มี SPT อย่างน้อย 65 มิลลิวินาทีหรือ 65ms

จากที่กล่าวในย่อหน้าข้างบน ผู้วิจัยถือว่าเป็นการวัดค่าความถูกต้องเพียงรอบเดียว สำหรับการวัดค่าความถูกต้องของล็อกเกอร์แต่ละตัวในวิทยานิพนธ์เล่มนี้จะมีการดำเนินการทั้งหมด 10 รอบ หลังจากนั้นจะนำจำนวน “hit” ทั้งหมดมาหาค่าเฉลี่ยและคำนวณเป็นเปอร์เซ็นต์ตามลำดับ สำหรับการทดลองนี้ได้แบ่งกลุ่มการวัดค่าความถูกต้องออกเป็น 2 กลุ่ม คือ 1) กลุ่มที่มีการรัน domU เพียงหนึ่งตัวในหนึ่งโหนดคอมพิว และ 2) กลุ่มที่มีการรัน domU มากกว่าหนึ่งตัวพร้อมกันในหนึ่งโหนดคอมพิว โดยแต่ละกลุ่มจะมีการกำหนดข้อมูลจำเพาะสำหรับสร้างเครื่อง

เสมือนหรือ domU ตามตาราง 1 และ ตาราง 2 ตามลำดับ ซึ่ง Auxsorn et al. (2020); Wiriya et al. (2020) ได้กล่าวว่า การปรับเปลี่ยนข้อมูลจำเพาะของเครื่องเสมือนหรือ domU จะมีผลต่อค่าความถูกต้องของระบบบันทึกเหตุการณ์

ตาราง 1 ข้อมูลจำเพาะของ domU สำหรับการวัดค่าความถูกต้องกรณีรัน domU เพียงตัวเดียวในหนึ่งโหนดคอมพิว

ลำดับที่	จำนวนคอร์ของซีพียูเสมือน	จำนวนหน่วยความหลัก (GB)	จำนวนของ domU ที่ทำงาน / โหนดคอมพิว
1	1	1	1
2	2	2	1
3	3	4	1

ตาราง 2 ข้อมูลจำเพาะของ domU สำหรับการวัดค่าความถูกต้องกรณีรัน domU มากกว่าหนึ่งตัวพร้อมกันในหนึ่งโหนดคอมพิว

ลำดับที่	จำนวนคอร์ของซีพียูเสมือน	จำนวนหน่วยความหลัก (GB)	จำนวนของ domU ที่ทำงาน / โหนดคอมพิว
1	1	1	3
2	2	2	3
3	3	4	3

2 การทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์

สำหรับการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด และหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ตามลำดับ ทั้งนี้ผู้วิจัยจะมุ่งเน้นไปที่ส่วนประกอบหลักของระบบบันทึกเหตุการณ์ คือ ล็อกเกอร์ของแต่ละสถาปัตยกรรมที่เสนอ ซึ่งการ

ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ในวิทยานิพนธ์เล่นนี้จะมองที่ความเร็วในการตรวจสอบหน่วยความจำของเครื่องเสมือนหรือ domU โดยให้ล็อกเกอร์ของแต่ละสถาปัตยกรรมที่เสนอตรวจสอบหน่วยความจำของเครื่องเสมือนหรือ domU จำนวน 100 รอบต่อครั้ง และจับเวลาที่ล็อกเกอร์ใช้ในการทำงานแต่ละครั้ง โดยจะทดลองเป็นจำนวนทั้งหมด 10 ครั้ง หลังจากนั้นนำระยะเวลาที่ล็อกเกอร์ใช้ในการทำงานแต่ละครั้งมาหาค่าเฉลี่ย และนำระยะเวลาที่ล็อกเกอร์แต่ละสถาปัตยกรรมที่เสนอใช้ในการทำงานเปรียบเทียบกับระยะเวลาที่ล็อกเกอร์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ใช้ในการทำงานซึ่งล็อกเกอร์นี้ถือว่าเป็นล็อกเกอร์หลักสำหรับเป็นพื้นฐานในการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับการทดลองนี้ผู้วิจัยจะใช้ข้อมูลจำเพาะของ domU คือ จำนวนคอร์หรือซีพียูของ domU เท่ากับ 1 คอร์ จำนวนของหน่วยความจำของ domU เท่ากับ 1 GB และอยู่ภายใต้สภาพแวดล้อมเดียวกันทุกครั้งสำหรับการทดลอง ทั้งนี้เพื่อลดปัจจัยที่อาจจะส่งผลกระทบต่อการทำงานของระบบบันทึกเหตุการณ์ และทุกครั้งที่รัน domU ครั้งแรกจะปล่อยให้ domU ทำงานอย่างน้อย 3 นาทีโดยที่ไม่มีการดำเนินการอย่างอื่น เพื่อให้ระบบปฏิบัติการของ domU มีความเสถียรก่อนการทดลอง

บทที่ 4

ผลการวิจัย

จากเนื้อหาทั้งหมดของบทที่ 3 วิธีการดำเนินการวิจัย ซึ่งได้กล่าวถึงสถาปัตยกรรมโครงสร้างภายใน และปัญหาของระบบบันทึกเหตุการณ์เมื่อนำมาประยุกต์ใช้งานใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็ก สถาปัตยกรรมของโอเพนสแต็ก การออกแบบและทดลองสร้างระบบบันทึกเหตุการณ์ และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine) โดยกรอบวิธีการดำเนินการวิจัยจะดำเนินการตามหัวข้อขอบเขตด้านวิธีการทดลอง ในหัวข้อนี้จะนำเสนอผลลัพธ์จากการออกแบบ การทำให้เกิดผล การทดลองและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ของแต่ละสถาปัตยกรรมที่ได้ออกแบบไว้ในบทที่ 3 วิธีการดำเนินการวิจัย โดยแบ่งออกเป็นหัวข้อย่อยดังต่อไปนี้

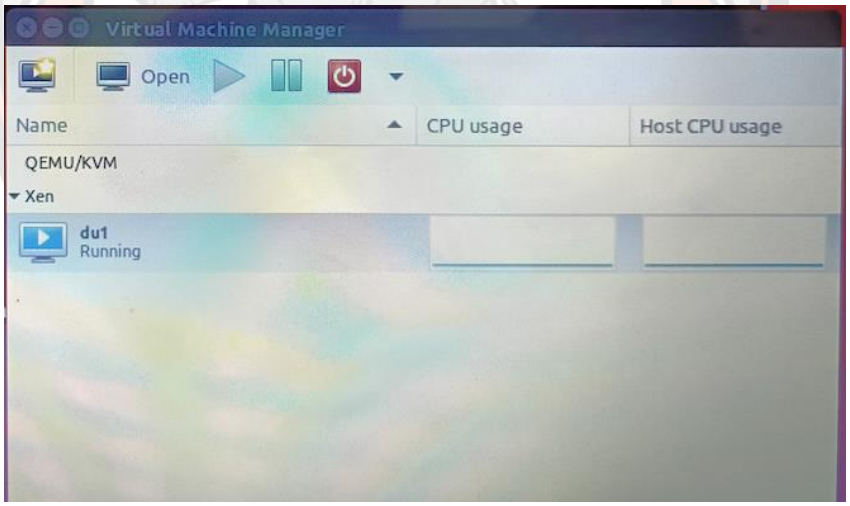
1. ผลลัพธ์ของการทดลองสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ
2. ผลลัพธ์ของการทดลองสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
3. ผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
4. ผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์
5. ผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด
6. ผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช
7. ผลลัพธ์ของการทดลองการวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ผลลัพธ์ของการทดลองสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

สำหรับหัวข้อนี้จะนำเสนอหน้าจอผลลัพธ์ของการทดลองสร้าง IaaS คลาวด์และติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการตามหัวข้อการสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ซึ่งผลลัพธ์ของการทดลองที่ได้จะสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.3 โดยมีผลลัพธ์ดังนี้

1. หน้าจอผลลัพธ์ของการทดลองสร้าง IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

จากหัวข้อการสร้าง IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ซึ่งได้อธิบายเกี่ยวกับขั้นตอนการสร้าง IaaS คลาวด์ เมื่อดำเนินการตามขั้นตอนดังกล่าวแล้วนั้น ก็จะได้ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการหรือภาพ 16 ซึ่งสร้างบนเครื่องคอมพิวเตอร์กายภาพเพียงเครื่องเดียว

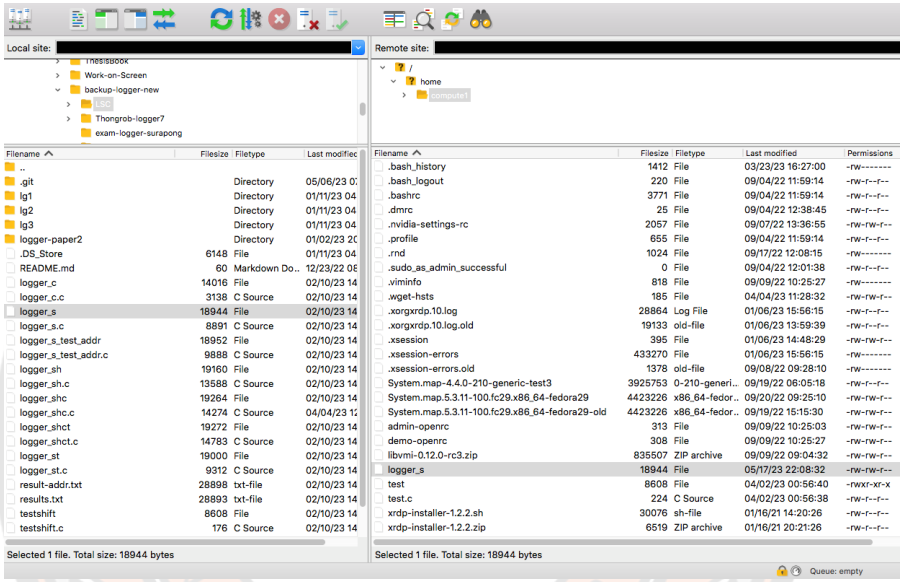


ภาพ 16 หน้าจอของ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

จากภาพ 16 ผู้วิจัยเรียกใช้ซอฟต์แวร์ Virtual Machine Manager ซึ่งเป็นเครื่องมือสำหรับการบริการจัดการ IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ เช่น การสร้าง การลบ การรัน การหยุดการทำงานชั่วคราว และการรีบูตเครื่องเสมือน (VM) เป็นต้น จากภาพนี้ผู้วิจัยได้สร้าง VM ขึ้นมาหนึ่งตัวชื่อ “du1” ซึ่งทำงานอยู่บนไฮเปอร์ไวเซอร์ Xen

2. หน้าจอผลลัพธ์ของการทดลองการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการใน IaaS คลาวด์

จากหัวข้อการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ ซึ่งได้อธิบายเกี่ยวกับขั้นตอนการติดตั้งระบบบันทึกเหตุการณ์ เมื่อดำเนินการตามขั้นตอนดังกล่าวแล้ว ก็จะได้ระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมจำลองของห้องปฏิบัติการ



ภาพ 17 หน้าจอการติดตั้งระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการผ่านซอฟต์แวร์ FileZilla

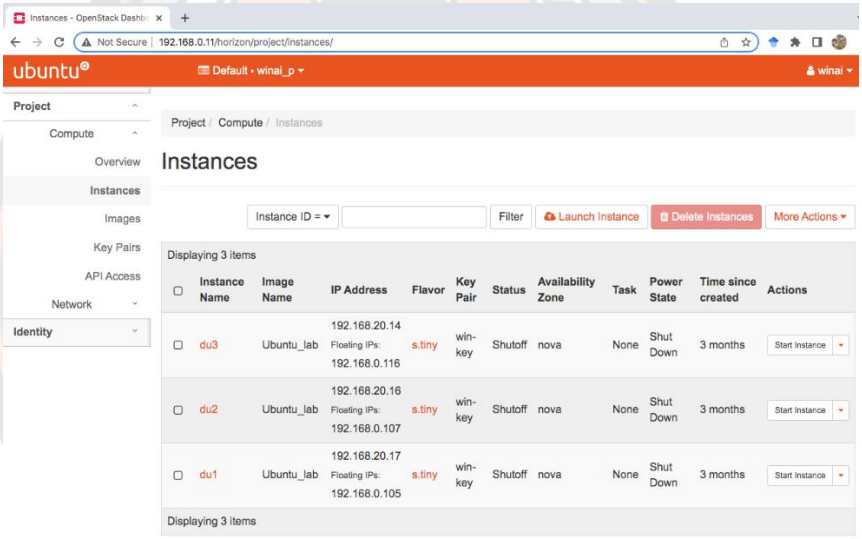
จากภาพ 17 ผู้วิจัยเรียกใช้ซอฟต์แวร์ FileZilla ซึ่งเป็นเครื่องมือสำหรับการรับส่งข้อมูลไปยังเซิร์ฟเวอร์ โดยในที่นี้ผู้วิจัยจะใช้สำหรับการสำเนาโปรแกรมระบบบันทึกเหตุการณ์ไปติดตั้งบนเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของเครื่องเสมือน ใน IaaS คลาวด์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ สำหรับการทดลองในวิทยานิพนธ์เล่มนี้กำหนดไดเรกทอรีสำหรับการจัดเก็บโปรแกรมระบบบันทึกเหตุการณ์นี้ไว้ที่ root@compute1:/home/compute1#

สำหรับการเรียกใช้งานระบบบันทึกเหตุการณ์ที่ติดตั้งบนเครื่องคอมพิวเตอร์กายภาพที่ทำหน้าที่เป็นโฮสต์ของเครื่องเสมือนใน IaaS คลาวด์ และการเรียกใช้แอปพลิเคชัน threat-app ที่ใช้จำลองสถานการณ์ว่าผู้โจมตีเข้ามาควบคุมแอปพลิเคชันนี้เพื่อใช้สำหรับอ่านไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ใน domU สำหรับการทดลองในวิทยานิพนธ์เล่มนี้จะใช้แอปพลิชัน

SSH Client ซึ่งเป็นการเข้าถึงระบบคอมพิวเตอร์ระยะไกลสำหรับการล็อกอินและการรันคำสั่งที่เครื่องคอมพิวเตอร์ปลายทางและมีความปลอดภัยเนื่องจากมีการเข้ารหัสข้อมูลระหว่างการสื่อสาร

ผลลัพธ์ของการทดลองสร้างสภาพแวดล้องของโอเพนสแต็กใน IaaS คลาวด์

สำหรับหัวข้อนี้จะนำเสนอหน้าจอผลลัพธ์ของการทดลองการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ตามหัวข้อการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งได้อธิบายเกี่ยวกับขั้นตอนการติดตั้ง IaaS คลาวด์โดยใช้โอเพนสแต็ก เมื่อดำเนินการตามขั้นตอนดังกล่าวแล้ว ก็จะได้สภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อใช้สำหรับการทดลองในวิทยานิพนธ์เล่มนี้



ภาพ 18 หน้าจอของสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ภาพ 18 เป็นหน้าจอของเว็บแดชบอร์ดซึ่งเป็นเครื่องมือสำหรับการบริการจัดการ IaaS คลาวด์สำหรับสภาพแวดล้อมของโอเพนสแต็ก ซึ่งสิทธิ์ในการบริการจัดการจะขึ้นอยู่กับระดับสิทธิ์ของบัญชีผู้ใช้งาน ตัวอย่างเช่น การสร้าง การลบ การรัน การหยุดการทำงานชั่วคราว และการรีบูตเครื่องเสมือน (VM) เป็นต้น จากภาพนี้ผู้วิจัยได้สร้างเครื่องเสมือนหรือ domU จำนวนสามตัว คือ du1 du2 และ du3

ผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้องของโอเพนสแต็กใน IaaS คลาวด์

สำหรับหัวข้อนี้จะนำเสนอหน้าจอผลลัพธ์การทดลองการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (IaaS OpenStack Logging System: IOLS) ตามหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งเป็นคลาวด์ที่ใช้ในสภาพแวดล้อมที่ใช้ในการผลิตจริง โดยผลลัพธ์การทดลองที่ได้จะสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.7 โดยมีหน้าจอผลลัพธ์ดังนี้

```
                                    a      b      c
①root@compute1:/home# ./logger du1 threat-app
     .....Waiting for answer.....
        a     b
②[1265][threat-app]
```

ภาพ 19 ล็อกเกอร์ใน dom0

```
①root@du1:/home# ./threat-app
```

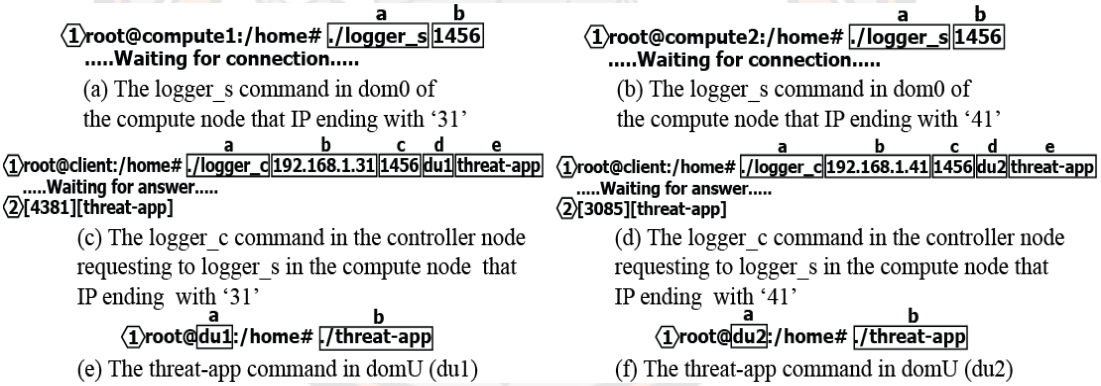
ภาพ 20 คำสั่ง threat-app ใน domU

ในภาพ 19 บรรทัดที่มีหมายเลข 1 หมายถึง เมื่อ logger ต้องการตรวจจับชื่อและโปรเซส ID ของโปรเซสที่ต้องการตรวจสอบหรือ threat-app ใน domU ดังนั้นจะใช้คำสั่ง logger ในช่องที่มีป้าย “a” จะถูกรันใน dom0 โดยผู้ให้บริการหรือผู้ใช้งานที่มีสิทธิ์ นอกจากนี้ยังมีพารามิเตอร์สองตัวของคำสั่งนี้ คือ ชื่อของ domU หรือ “du1” ดูในช่องที่มีป้าย “b” และชื่อของโปรเซสที่ต้องการตรวจสอบที่อยู่ใน domU หรือ “threat-app” ดูในช่องที่มีป้าย “c” คำสั่ง logger จะตรวจสอบ threat-app_mem จนกว่าจะพบโปรเซส threat-app ปรากฏขึ้นใน threat-app_mem หลังจากที่โปรเซส threat-app ถูกรันใน domU ดูบรรทัดที่มีหมายเลข 1 ในภาพ 20 นี้คือช่วงที่ชื่อและโปรเซส ID ของโปรเซส threat-app ปรากฏขึ้นใน threat-app_mem จากนั้น logger จะตรวจจับโปรเซส ID ของโปรเซส theat-app คือ “1265” และชื่อของโปรเซส theat-app คือ “threat-app” ดูในช่องที่มีป้าย “a” และ “b” ในบรรทัดที่มีหมายเลข 2 ในภาพ 19 ตามลำดับ จากนั้นคำสั่ง logger จะถูกยกเลิก

ผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับหัวข้อนี้จะนำเสนอหน้าจอผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ตามหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อปรับปรุงระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยใช้หลักการเขียนโปรแกรมซ็อกเก็ต โดยผลลัพธ์การทดลองที่ได้จะสอดคล้องกับหัวข้อ 1.5.2 ขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 9 โดยมีหน้าจอผลลัพธ์ดังนี้

ภาพ 21 คือ หน้าจอผลลัพท์ของการทดสอบระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ที่มีการขยายโหนดคอมพิวหรือภาพ 8 ซึ่งมีการประยุกต์ใช้การเขียนโปรแกรมซ็อกเก็ตกับระบบบันทึกเหตุการณ์ในภาพ 6 ทำให้เกิดเป็นระบบบันทึกเหตุการณ์แบบรวมศูนย์ และทำให้เกิดความสะดวกในการจัดการระบบบันทึกเหตุการณ์ในสภาพแวดล้อมของโอเพนสแต็ก



ภาพ 21 รูปแบบคำสั่งของ logger_s logger_c threat-app และผลลัพธ์ของคำสั่ง

หน้าจอผลลัพธ์การดำเนินงานของสถาปัตยกรรมระบบบันทึกเหตุการณ์แบบรวมศูนย์ที่เสนอในภาพ 8 คือ ภาพ 21(a) ถึง ภาพ 21(f) ส่วนแรกของผลลัพธ์ คือ ภาพ 21(a) ถึง ภาพ 21(d) ซึ่งสร้างมาจากขั้นตอนการทำงานหกขั้นตอนของสถาปัตยกรรมระบบบันทึกเหตุการณ์แบบรวมศูนย์ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ต ส่วนที่สองของผลลัพธ์ คือ ภาพ 21(e) ถึง ภาพ 21(f) สร้างมาจากขั้นตอนการทำงานของแอปพลิเคชัน threat-app ซึ่งกล่าวไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ใน IaaS คลาวด์สำหรับสภาพแวดล้องของห้องปฏิบัติการ

ในภาพ 21(a) บรรทัดที่มีหมายเลข 1 เป็นรูปแบบคำสั่งของ logger_s เมื่อต้องการตรวจจับชื่อและโปรเซส ID ของโปรเซสเป้าหมายที่ต้องการตรวจสอบหรือ threat-app ใน domU ของผู้ใช้บริการ logger_s ในกล่องที่มีป้าย “a” มีพารามิเตอร์หนึ่งตัว คือ หมายเลขซ็อกเก็ตหรือหมายเลขพอร์ต (Port Number) ในกล่องที่มีป้าย “b” หรือ 1456 คำสั่งนี้จะดำเนินการใน dom0 โดยผู้ใช้ที่ได้รับอนุญาต หลังจากรัน logger_s แล้วก็จะรอการร้องขอการเชื่อมต่อจาก logger_c โดย logger_c จะอยู่ในเครื่องคอมพิวเตอร์กายภาพ (Physical Machine: PM) ที่เชื่อมต่อกับเครือข่ายการจัดการเดียวกัน ดู 3’ ในภาพ 8 ของโอเพนสแต็ก IaaS คลาวด์

ในภาพ 21(c) บรรทัดที่มีหมายเลข 1 เป็นรูปแบบคำสั่งของ logger_c ในกล่องที่มีป้าย “a” ซึ่งทำงานร่วมกับ logger _s เมื่อต้องการตรวจสอบโปรเซสเป้าหมายหรือ threat-app ใน domU ของผู้ใช้บริการ ซึ่ง logger_c มีสี่พารามิเตอร์ ได้แก่ ไอพีแอดเดรส (IP Address) หมายเลขพอร์ต ชื่อของ domU และชื่อของโปรเซสเป้าหมาย โดยไอพีแอดเดรสในกล่องที่มีป้าย “b” หรือ “192.168.1.31” คือ ที่อยู่เฉพาะของอุปกรณ์บนเครือข่ายท้องถิ่นของโหนดคอมพิวในโอเพนสแต็ก IaaS คลาวด์ หมายเลขพอร์ตในกล่องที่มีป้าย “c” หรือ “1456” ซึ่งระบุถึงพอร์ตของ logger_s บนโหนดคอมพิวในโอเพนสแต็ก IaaS คลาวด์และหมายเลขพอร์ตนี้จะต้องตรงกับหมายเลขพอร์ตในกล่องที่มีป้าย “b” ของภาพ 21(a) ชื่อของ domU ในกล่องที่มีป้าย “d” หรือ “du1” เป็นชื่อของ domU เป้าหมายในโหนดคอมพิวในโอเพนสแต็ก IaaS คลาวด์และสุดท้ายชื่อของโปรเซสเป้าหมายในกล่องที่มีป้าย “e” หรือ “threat-app” คือ ชื่อของโปรเซสเป้าหมายใน domU ที่ logger_s ต้องการจะตรวจสอบ

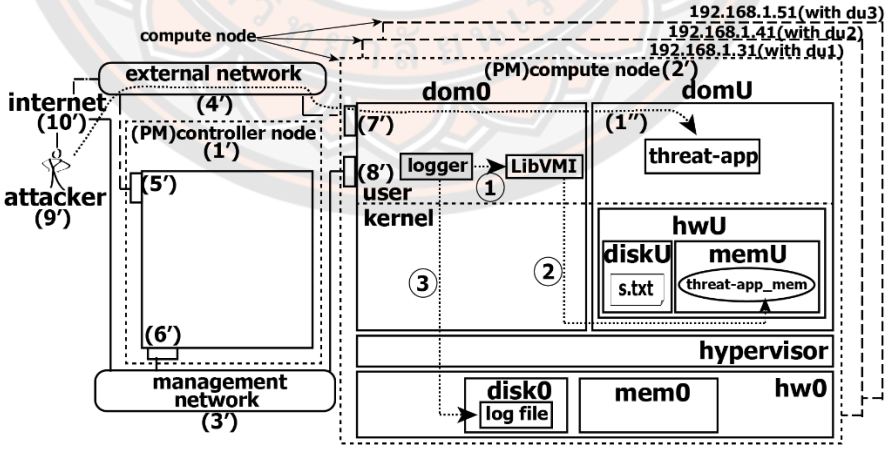
ขั้นตอนการทำงานร่วมกันระหว่าง logger_s และ logger_c ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตแล้ว ภาพ 21(d) จะมีรูปแบบคำสั่งเหมือนกันภาพ 21(c) แต่จะมีพารามิเตอร์ที่แตกต่างกันในกล่องที่มีป้าย “b” “c” “d” และ “e” ตัวอย่างเช่น ภาพ 21(d) มีพารามิเตอร์แตกต่างจากภาพ 21(c) ดูที่กล่องที่มีป้าย “b” และ “d” ของภาพ คือ ไอพีแอดเดรสที่ลงท้ายด้วย 41 แทนที่ 31 และ du2 แทนที่ du1 ภาพ 21(e) คือ คำสั่งของ threat-app ที่รันใน domU หรือดูบรรทัดที่มีหมายเลข 1 ในภาพจากนั้นชื่อและโปรเซส ID ของ threat-app จะปรากฏขึ้นในพื้นที่หน่วยความจำเสมือนภายใน memU ของ du1 ซึ่งพื้นที่นี้เรียกว่า mem ของ threat-app ดูรูปวงรีในภาพ 3 ซึ่งกล่าวไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์ในระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์สำหรับสภาพแวดล้อมของห้องปฏิบัติการแล้ว จากนั้น logger_s ในภาพ 21(a) สามารถตรวจจับโปรเซส ID หรือ “4381” และชื่อของโปรเซส หรือ “threat-app” ดู

บรรทัดที่มีหมายเลข 2 ของภาพ 21(c) สำหรับภาพ 21(f) ภาพ 21(b) และภาพ 21(d) มีการทำงานที่เหมือนกันทั้งนี้ในสภาพแวดล้อมการทดลองนี้ “threat-app” ในภาพ 21(e) และภาพ 21(f) ซึ่งมีชื่อเดียวกันแต่จะมีความแตกต่างกันเนื่องจากมีโปรเซส ID ใน domU ที่แตกต่างกัน (du1 และ du2)

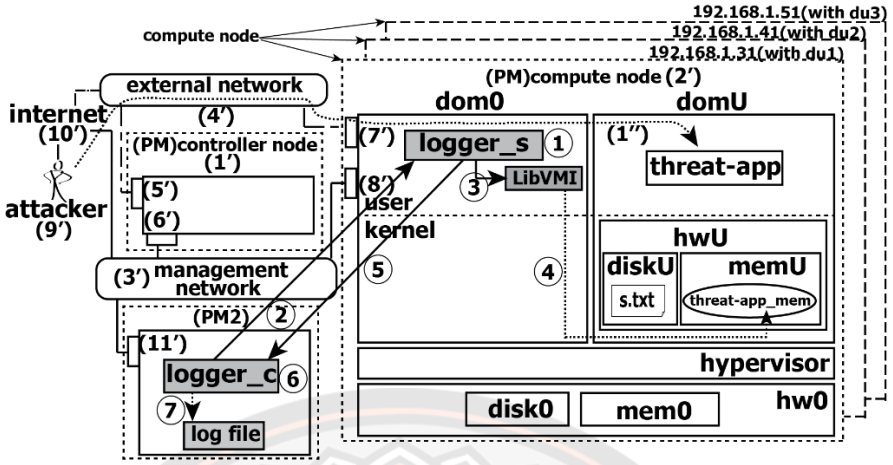
ภาพ 21(d) แสดงให้เห็นว่า logger_c ที่อยู่ในเครื่องคอมพิวเตอร์กายภาพเครื่องเดียวกันกับภาพ 21(c) logger_c ในภาพ 21(c) และในภาพ 21(d) เป็นตัวเดียวกันและวางอยู่ในเครื่องคอมพิวเตอร์กายภาพเดียวกัน ซึ่งเครื่องคอมพิวเตอร์นี้เรียกว่าโหนดคอนโทรเลอร์ ดูกล่องเส้นประที่มีหมายเลข 1’ ทางด้านซ้ายของภาพ 8 ดังนั้นสถาปัตยกรรมของระบบบันทึกเหตุการณ์ที่เสนอในโอเพนสแต็ก IaaS คลาวด์ที่มีการขยายโหนดคอมพิวในภาพ 8 จึงเป็นระบบบันทึกเหตุการณ์แบบรวมศูนย์ นี้เป็นเพราะมี logger_c เพียงตัวเดียวในสถาปัตยกรรม จากนั้น logger_c สามารถร้องขอการเชื่อมต่อไปยัง logger_s สองตัว คือ logger_s ในกล่องที่มีป้าย “a” ของภาพ 21(c) และlogger_s อีกตัวในกล่องที่มีป้าย “a” ของภาพ 21(b) ตามที่กล่าวไว้ข้างต้น

จากหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์นี้ซึ่งกล่าวไว้ข้างต้น ผู้วิจัยขออภิปรายสิ่งที่ได้รับจากการปรับปรุงครั้งนี้ ดังต่อไปนี้

1. สถาปัตยกรรมแบบรวมศูนย์มีการจัดการที่ง่ายกว่าเมื่อเทียบกับสถาปัตยกรรมแบบกระจายศูนย์



ภาพ 22 สถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบกระจายศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์



ภาพ 23 การแยกภาระงานของโหนดคอนโทรเลอร์

เพื่อความชัดเจนในการอภิปรายผู้วิจัยได้แก้ไขภาพ 6 ให้เป็นภาพ 22 และจากภาพ 22 จะเห็นว่ามีเครื่องคอมพิวเตอร์กายภาพ (Physical Machine: PM) จำนวนสามเครื่องที่มีไอพีแอดเดรสลงท้ายด้วย 31 41 และ 51 ดูกล่องแรเงาในภาพนั้นคือ logger ถูกวางไว้ในแต่ละ PM ดังนั้นจะมี logger จำนวนสามตัวสำหรับ PM ทั้งหมด ด้วยเหตุนี้ logger เหล่านี้จึงถูกกระจายในสถาปัตยกรรมทั้งหมดของภาพ 22 ทั้งนี้ผู้วิจัยของเรียกว่า “การกระจายศูนย์” และผู้ใช้งานที่ได้รับอนุญาตจำเป็นต้องกำหนดค่าและจัดการ logger แต่ละตัวในสถาปัตยกรรมแบบกระจายศูนย์นี้เป็นการจัดการที่ยุ่งยากซึ่งได้กล่าวถึงในหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ในภาพ 8 มี logger จำนวนสามตัวทั้งนี้ผู้วิจัยขอเรียแต่ละตัวว่า “logger_s” สำหรับ PM ทั้งหมดโดยมีไอพีแอดเดรสลงท้ายด้วย 31 41 และ 51 อย่างไรก็ตาม logger_s ทั้งสามตัวนี้ถูกจัดการโดยตัวจัดการเพียงตัวเดียว คือ logger_c ดังนั้นตัวจัดการนี้จะไม่ถูกกระจายในสถาปัตยกรรมทั้งหมดของภาพ 8 ผู้ใช้งานที่ได้รับอนุญาตสามารถกำหนดค่าและจัดการ logger_s ทั้งหมดเพียงจุดเดียวผ่าน logger_c ในสถาปัตยกรรมแบบรวมศูนย์นี้ ทำให้การจัดการได้ง่ายกว่าเมื่อเทียบกับภาพ 22

จากหน้าจอผลลัพธ์ภาพ 21(a) ถึง ภาพ 21(f) แสดงผลลัพธ์ของการออกแบบสถาปัตยกรรมแบบรวมศูนย์นี้ไว้ในภาพ 8 โดยเฉพาะอย่างยิ่งภาพ 21(c) และภาพ 21(d) แสดงให้เห็นว่ามีตัวจัดการเพียงตัวเดียว ซึ่งก็คือ logger_c ในกล่องที่มีป้าย “a” จากภาพทั้งสอง จากหน้าจอผลลัพธ์ในหัวข้อนี้ตัวจัดการนี้สามารถจัดการ logger_s เพื่อบันทึกข้อมูลของโปรเซสที่เป็นอันตรายใน PM สองตัวที่มีไอพีแอดเดรสลงท้ายด้วย 31 และ 41 ตัวอย่างของโปรเซสเหล่านี้ คือ “threat-app” ดูกล่องที่มีป้าย “b” ในภาพ 21(e) แอปพลิเคชันนี้อยู่ใน domU ชื่อ du1 ดูกล่องที่มี

ป้าย “a” ในภาพ 21(e) อีกตัวอย่างหนึ่งของโปรเซสที่เป็นอันตราย คือ “threat-app” ดูกล่องที่มีป้าย “b” ในภาพ 21(f) ใน du2 ดูกล่องที่มีป้าย “a” ในภาพ 21(f) โดยสรุปสถาปัตยกรรมของระบบบันทึกเหตุการณ์แบบรวมศูนย์ในโอเพนสแต็ก IaaS คลาวด์ สามารถลดความยุ่งยากในการจัดการ logger_s ที่กระจายในแต่ละ PM ในคลาวด์นี้ได้ ผู้วิจัยเชื่อว่าการจัดการสถาปัตยกรรมแบบรวมศูนย์จึงมีความง่ายกว่าเมื่อเทียบกับสถาปัตยกรรมแบบกระจายศูนย์

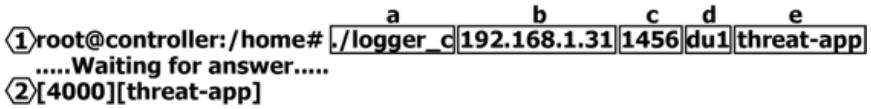
2. การแยกภาระงานของโหนดคอนโทรเลอร์

จากภาพ 8 จะเห็นว่าโหนดคอนโทรเลอร์ คือ หนึ่ง PM ผู้วิจัยขอเรียกว่า “PM1” ดูหมายเลข 1’ ในภาพ 8 PM1 นี้มี logger_c และ log file ดูกล่องแรเงาในโหนดคอนโทรเลอร์ในภาพ 8 ซึ่งสามารถย้าย logger_c และ log file ไปวางใน PM2 ได้ ดูกล่องเส้นประด้านล่างในภาพ 23 การย้ายตำแหน่งนี้สามารถแยกภาระงานของ logger_c และ log file ในโหนดคอนโทรเลอร์หรือ PM1 ได้ ดังนั้น PM1 ก็สามารถทำงานเฉพาะงานหลักที่จำเป็น เช่น แดชบอร์ด อิมเมจไอเด็นติตี้ และอื่น ๆ การแยกนี้ทำได้ในสองขั้นตอนหลักโดยสังเขป คือ ขั้นตอนแรกเป็นการย้าย logger_c และ log file ไปยัง PM2 และขั้นตอนที่สองเป็นการเชื่อมต่อ PM2 กับเครือข่ายการจัดการของ PM1 หรือหมายเลข 3’ ในภาพ 8 ผลลัพธ์ของการเชื่อมต่อเครือข่ายการจัดการนี้ทำให้ PM2 ทำงานร่วมกับส่วนประกอบอื่น ๆ ของสถาปัตยกรรมแบบรวมศูนย์ได้ ซึ่งแสดงในภาพ 23 ทำให้ PM1 และ PM2 แยกจากกันและทำงานเฉพาะงานหลักที่จำเป็นของตนเอง สิ่งนี้สามารถปรับปรุงการจัดการระบบโดยรวมได้และเพิ่มประสิทธิภาพของระบบบันทึกเหตุการณ์ในสถาปัตยกรรมนี้ได้

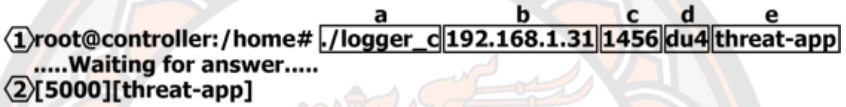
ผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

สำหรับหัวข้อนี้จะนำเสนอหน้าจอผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรดตามหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด เพื่อให้สอดคล้องกับการทำงานในสภาพแวดล้อมของโอเพนสแต็ก และมีความสะดวกในการจัดการมากขึ้น ซึ่งผลลัพธ์การทดลองที่ได้จะสอดคล้องกับหัวข้อขอบเขตด้านวิธีการทดลอง หัวข้อย่อย 2.11 โดยมีหน้าจอผลลัพธ์ดังนี้

ภาพ 24 การเรียกใช้ logger_s **ใน** dom0



ภาพ 25 การเรียกใช้ logger_c **ในโหนดคอนโทรเลอร์ครั้งที่** 1



ภาพ 26 การเรียกใช้ logger_c **ในโหนดคอนโทรเลอร์ครั้งที่** 2

ภาพ 27 การเรียกใช้ threat-app **ใน** domU (du1)

ภาพ 28 การเรียกใช้ threat-app **ใน** domU (du4)

ใน**ภาพ 24** บรรทัดที่มีหมายเลข 1 เป็นรูปแบบคำสั่งสำหรับการเรียกใช้ logger_s เมื่อต้องการตรวจจับชื่อและโปรเซส ID ของโปรเซสเป้าหมายที่ต้องการตรวจสอบหรือ thread-app ใน domU ของผู้ใช้บริการ logger_s ในกล่องที่มีป้าย “a” มีพารามิเตอร์หนึ่งตัว คือ หมายเลขซ็อกเก็ต หรือหมายเลขพอร์ตในกล่องที่มีป้าย “b” หรือ 1456 คำสั่งนี้จะดำเนินการใน dom0 โดยผู้ใช้ที่ได้รับอนุญาต หลังจากนั้น logger_s แล้วก็จะรอการร้องขอการเชื่อมต่อจาก logger_c ซึ่งอยู่ใน

เครื่องคอมพิวเตอร์กายภาพที่เชื่อมต่อกับเครือข่ายการจัดการเดียวกันสำหรับสภาพแวดล้อมของโอเพนสแต็ก

ในภาพ 25 บรรทัดที่มีหมายเลข 1 เป็นรูปแบบคำสั่งการเรียกใช้ logger_c ในกล่องที่มีป้าย “a” ซึ่งทำงานร่วมกับ logger_s เมื่อต้องการตรวจสอบโปรเซสเป้าหมายหรือ threat-app ใน domU ของผู้ใช้บริการ ซึ่ง logger_c มีสี่พารามิเตอร์ ได้แก่ ไอพีแอดเดรส หมายเลขพอร์ท ชื่อของ domU และชื่อของโปรเซสเป้าหมาย โดยไอพีแอดเดรสในกล่องที่ป้าย “b” หรือ “192.168.1.31” คือที่อยู่เฉพาะของอุปกรณ์บนเครือข่ายท้องถิ่นของโหนดคอมพิวในโอเพนสแต็ก IaaS คลาวด์ หมายเลขพอร์ตในกล่องที่มีป้าย “c” หรือ “1456” คือพอร์ตของ logger_s บนโหนดคอมพิวในโอเพนสแต็กและตรงกับหมายเลขพอร์ตในกล่องที่มีป้าย “b” ในภาพ 24 ชื่อของ domU ในกล่องที่มีป้าย “d” หรือ “du1” ซึ่งเป็นชื่อของ domU ที่ต้องการตรวจสอบในโอเพนสแต็ก IaaS คลาวด์ และสุดท้ายชื่อของโปรเซสเป้าหมายในกล่องที่มีป้าย “e” หรือ “threat-app”

ขั้นตอนการทำงานร่วมกันระหว่าง logger_s และ logger_c ได้อธิบายไว้ในหัวข้อสถาปัตยกรรมของระบบบันทึกเหตุการณ์บนพื้นฐานการเขียนโปรแกรมซ็อกเก็ตแล้ว ภาพ 26 จะมีรูปแบบคำสั่งเหมือนกันภาพ 25 แต่จะสามารถมีพารามิเตอร์ที่แตกต่างกันในกล่องที่มีป้าย “d” ตัวอย่างเช่น ภาพ 25 มีพารามิเตอร์แตกต่างจากภาพ 26 ดูที่กล่องที่มีป้าย “d” ของภาพ คือ du1 แทนที่ du4 ภาพ 27 คือ คำสั่งของ threat-app ที่รันใน domU หรือดูบรรทัดที่มีหมายเลข 1 ในภาพจากนั้นชื่อและโปรเซส ID ของ threat-app จะปรากฏขึ้นในพื้นที่หน่วยความจำเสมือนภายใน memU ของ du1 จากนั้น logger_s ใน ภาพ 24 สามารถตรวจจับโปรเซส ID หรือ “4000” และชื่อของโปรเซส หรือ “threat-app” ดูบรรทัดที่มีหมายเลข 2 ของภาพ 25 สำหรับภาพ 28 และภาพ 26 มีการทำงานที่เหมือนกันทั้งนี้ในสภาพแวดล้อมการทดลองนี้ “threat-app” ในภาพ 27 และภาพ 28 ซึ่งมีชื่อเดียวกันแต่จะมีความแตกต่างกันเนื่องจากมีโปรเซส ID ใน domU ที่แตกต่างกัน (du1 และ du4)

จากหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด และผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์นี้ซึ่งกล่าวไว้ข้างต้น ผู้วิจัยขออภิปรายสิ่งที่ได้รับจากการปรับปรุงครั้งนี้ ดังต่อไปนี้

ระบบบันทึกเหตุการณ์ในหัวข้อนี้เป็นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เนื่องจากสถาปัตยกรรมของระบบบันทึกเหตุการณ์นี้หรือภาพ 8 ที่

ประยุกต์ใช้การเขียนโปรแกรมซ็อกเก็ตสำหรับการสื่อสารบนพื้นฐานไคลเอ็นท์/เซิร์ฟเวอร์โดยใช้โพรโตคอล TCP ซึ่งผู้วิจัยเลือกใช้นั้นจำเป็นต้องทราบหมายเลขซ็อกเก็ตของ logger_s ใน dom0 ของโหนดคอมพิวก่อนถึงจะสามารถเรียกใช้ logger_c ที่อยู่ในโหนดคอนโทรเลอร์ให้ร้องขอการเชื่อมต่อไปยัง logger_s ใน dom0 ของโหนดคอมพิวได้ จากภาพ 8 หากต้องการรัน logger_s จำนวนสองตัวบนโหนดคอมพิวที่มีไอพีแอดเดรสลงท้ายด้วย 31 จำเป็นต้องระบุหมายเลขซ็อกเก็ตแตกต่างกัน ตัวอย่างคำสั่งสำหรับรัน logger_s จำนวนสองตัว เช่น ./logger_s 1456 และ ./logger_s 5678 ดูภาพ 9 และ ภาพ 10 ตามลำดับ ภาพทั้งสองจะมีความแตกต่างกันตรงส่วนของกล่องที่มีป้าย “b” (1456 และ 5678) จะเห็นได้ว่าหากต้องการรัน logger_s จำนวนมากในโหนดคอมพิวเดียวกันเป็นเรื่องที่ยุ่งยากสำหรับการจัดการ ดังนั้นการประยุกต์ใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด สำหรับระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์หรือภาพ 8 สามารถแก้ปัญหาที่กล่าวได้ และยังจัดการให้ logger_s ทำงานแบบขนานกันได้ นั่นคือระบบบันทึกเหตุการณ์นี้สามารถตรวจสอบ domU ได้มากกว่าหนึ่งตัวพร้อมกันในเวลาเดียวกันได้ง่ายกว่าระบบบันทึกเหตุการณ์ในหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ นอกจากนี้โดยภาพรวมแล้วการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ยังสามารถปรับปรุงความเร็วของแอปพลิเคชันหรือโปรแกรม โดยอนุญาตให้ใช้ประโยชน์จากซีพียูหลายตัว ดำเนินการกับอุปกรณ์ Input / Output แบบอะซิงโครนัส ดำเนินการหลายงานพร้อมกันได้ (Butenhof, 1997)

ผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

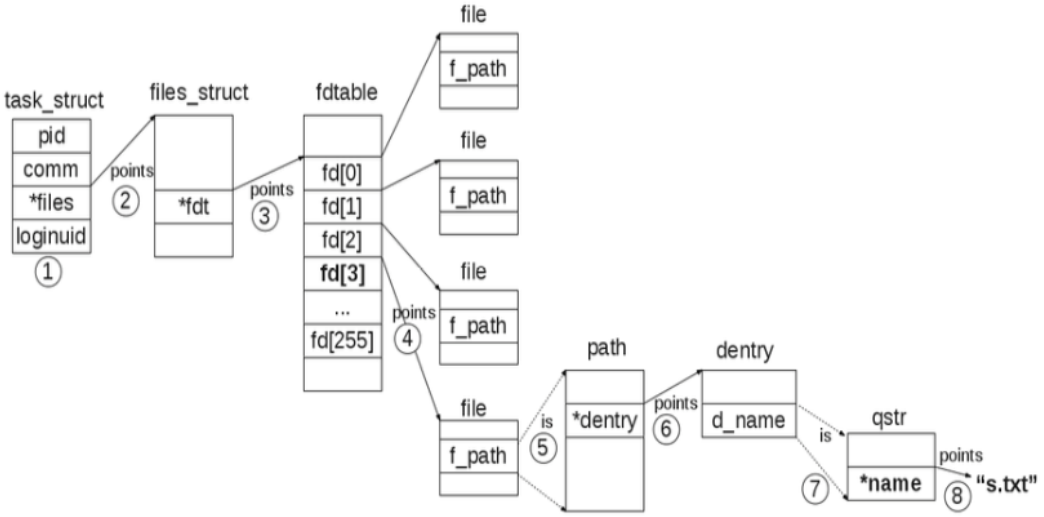
สำหรับหัวข้อนี้จะนำเสนอผลลัพธ์ของการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด โดยการประยุกต์ใช้ตารางแฮชเพื่อปรับปรุงความเร็วในการตรวจสอบหน่วยความจำเสมือนของ domU ของระบบบันทึกเหตุการณ์นี้ จากหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ซึ่งได้วิเคราะห์ชุดคำสั่งของระบบบันทึกเหตุการณ์นี้พบว่าการทำงานส่วนใหญ่เกิดขึ้นที่ส่วนประกอบของ logger_s เพื่อตรวจสอบข้อมูลในหน่วยความจำเสมือนของ domU โดยการเรียกใช้ไลบารี LibVMI เพื่อดึงข้อมูลที่ต้องการมา

ตรวจสอบโดยการเปรียบเทียบว่าข้อมูลที่ได้รับมาจากไลบารี LibVMI เป็นโปรเซสเป้าหมายที่ต้องการตรวจสอบหรือไม่ผ่านคำสั่งของภาษา C คือ ฟังก์ชัน strcmp

จากผลลัพธ์ของการวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์ในหัวข้อนี้หรือภาพ 33 การวัดค่าความถูกต้องของระบบบันทึกเหตุการณแบบรวมศูนย์สำหรับภาพ 32 ในหัวข้อการวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ที่แสดงให้เห็นว่าเมื่อค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app มีค่าตั้งแต่ 2ms ขึ้นไประบบบันทึกเหตุการณ์ในหัวข้อนี้มีค่าความถูกต้อง 100% ตลอด เมื่อเปรียบเทียบกับ ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ซึ่งระบบบันทึกเหตุการณ์เหล่านี้มีค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app มีค่าตั้งแต่ 3ms ขึ้นไปถึงทำให้ระบบบันทึกเหตุการณ์เหล่านี้มีค่าความถูกต้อง 100% ตลอด ดังนั้นสิ่งที่ได้รับจากการปรับปรุงครั้งนี้ทำให้ระบบบันทึกเหตุการณ์ในหัวข้อนี้ทำงานได้เร็วขึ้น 1ms เนื่องมาจากการประยุกต์ใช้ตารางแฮชที่ช่วยลดระยะเวลาในการตรวจสอบของระบบบันทึกเหตุการณ์นี้ให้น้อยลง ทำให้สามารถลดค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ลงได้ 1ms หรืออีกนัยคือระบบบันทึกเหตุการณ์นี้สามารถทำงานได้เร็วขึ้น

อย่างไรก็ตามการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด โดยการประยุกต์ใช้ตารางแฮชเพื่อปรับปรุงความเร็วในการตรวจสอบหน่วยความจำเสมือนของ domU เป็นเพียงการทดสอบที่ให้ความสนใจ Process-centric log โดยการจัดเก็บโปรเซสการทำงานปกติของระบบปฏิบัติการ รวมถึงโปรเซสการทำงานของแอปพลิเคชันที่ตั้งสมมุติฐานว่าไม่ใช่โปรเซสที่เป็นอันตรายเก็บไว้ในตารางแฮชเท่านั้น ซึ่งจากผลลัพธ์การทดลองสามารถลดระยะเวลาที่ระบบบันทึกเหตุการณ์นี้ใช้เพื่อตรวจสอบข้อมูลในหน่วยความจำเสมือนของ domU จากแนวคิดนี้สามารถนำไปประยุกต์ใช้กับ File-centric log ซึ่งเป็นการจัดเก็บประวัติของเหตุการณ์ของไฟล์ที่ถูกเข้าถึงจากโปรเซสได้ ตัวอย่างเช่น การจัดเก็บประวัติของเหตุการณ์ของไฟล์ (s.txt) ที่ถูกเข้าถึงจากโปรเซสที่รันบนระบบปฏิบัติการของ domU ซึ่งลักษณะนี้ถือว่าเป็น File-centric log

กระบวนการตรวจสอบว่าโปรเซสใด ๆ (threat-app) ที่รันใน domU กำลังเข้าถึงไฟล์ไหนใน domU ของระบบบันทึกเหตุการณ์หรือล็อกเกอร์ที่ยังไม่มีการประยุกต์ใช้ตารางแฮชพอสังเขปมีขั้นตอนดังนี้



ภาพ 29 โครงสร้างข้อมูลของ Linux Kernel **สำหรับหน่วยความจำเสมือนของโปรเซส** threat-app **ที่อ่านไฟล์** s.txt

ขั้นตอนที่ 1 ล็อกเกอร์ดึงข้อมูลชื่อของโปรเซสจาก task_struct ของโปรเซส

ขั้นตอนที่ 2 ล็อกเกอร์ดึงตัวชี้ไฟล์ (*files) ซึ่งอยู่ใน task_struct ดู**ภาพ 29** ตัวชี้ไฟล์นี้จะชี้ไปยัง files_struct ซึ่งภายในจะเก็บตัวชี้ *fdt

ขั้นตอนที่ 3 ล็อกเกอร์ดึงตัวชี้ fdt (*fdt) ใน file_struct ที่ชี้ไปยัง fd[0] ซึ่งเป็นไฟล์แรกที่เปิดอ่านและเก็บไว้ในโครงสร้าง fdtable ซึ่งโครงสร้างนี้จะเก็บตัวชี้ไปยัง fd[0]-fd[255] โดยแต่ละ fd[i] จะเป็นตัวชี้ไปยังแต่ละไฟล์ที่ถูกเปิดโดยโปรเซส threat-app

ขั้นตอนที่ 4 สมมุติว่าโปรเซส threat-app กำลังเปิดอ่านไฟล์ s.txt ดังนั้น s.txt ก็จะมีโครงสร้างข้อมูล file ซึ่งโครงสร้างนี้เป็นตัวชี้ไปยัง fd[3]

ขั้นตอนที่ 5 ในขั้นตอนที่ 4 โครงสร้างข้อมูล file ของ s.txt ที่ถูกชี้ด้วย fd[3] แล้วขั้นตอนที่ 5 จะดึงค่า f_path ที่อยู่ในโครงสร้างข้อมูล file ซึ่งเป็นโครงสร้าง path ที่เก็บตัวชี้ dentry

ขั้นตอนที่ 6 ตัวชี้ *dentry นี้จะชี้ไปยังโครงสร้าง dentry ซึ่งเป็นที่เก็บตัวแปร d_name

ขั้นตอนที่ 7 ตัวชี้ d_name เป็นโครงสร้าง qstr ซึ่งเป็นโครงสร้างที่มีตัวชี้ของตัวแปร *name ที่ชี้ไปยังไฟล์ชื่อ s.txt

ขั้นตอนที่ 8 ล็อกเกอร์ก็จะได้ข้อความชื่อไฟล์ของ s.txt

จากกระบวนการตรวจสอบว่าโปรเซสที่รันใน domU กำลังเข้าถึงไฟล์ไหนใน domU ของระบบบันทึกเหตุการณ์หรือล็อกเกอร์ที่ยังไม่มีการประยุกต์ใช้ตารางแฮชที่กล่าวมาสามารถจัดกลุ่มการทำงานออกเป็นแต่ละส่วน ได้แก่ ส่วนที่ 1 คือ ส่วนของการดึงชื่อโปรเซส ซึ่งดำเนินการในขั้นตอนที่ 1 และส่วนที่ 2 คือ ส่วนที่ใช้สำหรับการค้นหาว่าโปรเซสที่ต้องการตรวจสอบกำลังเปิดอ่านไฟล์อะไรอยู่ ซึ่งดำเนินการในขึ้นตอนที่ 2 ถึงขั้นตอนที่ 8 ตามลำดับ จากกระบวนการดังกล่าวสามารถวิเคราะห์ระยะเวลาที่ล็อกเกอร์ใช้สำหรับการตรวจสอบว่าโปรเซสที่รันใน domU กรณีที่ยังไม่มีการประยุกต์ใช้ตารางแฮชที่กำลังเข้าถึงไฟล์ s.txt คือ TRD_LibVMI + (2+(O(5n)TRD_LibVMI) และ ล็อกเกอร์กรณีที่มีการประยุกต์ใช้ตารางแฮชที่กำลังเข้าถึงไฟล์ s.txt คือ TRD_LibVMI + (1+(O(m) + TRD_Hmem) โดยที่ TRD_LibVMI หมายถึง ระยะเวลาที่ไลบารี LibVMI ใช้ในการดึงข้อมูลจากหน่วยความจำเสมือนของ domU และ TRD_Hmem หมายถึง ระยะเวลาที่ใช้ในการดึงข้อมูลจากตารางแฮช ซึ่งระยะเวลาของ TRD_LibVMI จะมีค่ามากกว่า TRD_Hmem เนื่องจาก TRD_LibVMI เป็นการดึงข้อมูลจาก memory space ของ domU ทำให้ต้องใช้ระยะเวลามากกว่า TRD_Hmem ที่เป็นการดึงข้อมูลจาก memory space ของล็อกเกอร์ และสำหรับ m และ n หมายถึงจำนวนไฟล์ที่โปรเซสหนึ่งกำลังเข้าถึงไฟล์นั้น เมื่อเปรียบเทียบระยะเวลาที่ล็อกเกอร์ใช้สำหรับการตรวจสอบว่าโปรเซสที่รันใน domU พบว่าล็อกเกอร์กรณีที่มีการประยุกต์ใช้ตารางแฮชสามารถทำงานได้เร็วกว่า เนื่องจากกลุ่มการทำงานส่วนที่ 2 ใช้ระยะเวลาในการค้นหาในตาราแฮชเท่ากับ 1+(O(m)) + TRD_Hmem ในขณะที่ล็อกเกอร์กรณีที่ไม่มีการประยุกต์ใช้ตารางแฮชเท่ากับ 2+(O(5n) TRD_LibVMI ประกอบกับระยะเวลาที่ใช้ในการดำเนินการของ TRD_Hmem น้อยกว่า TRD_LibVMI ด้วย

ผลลัพธ์ของการทดลองการวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

สำหรับหัวข้อนี้จะนำเสนอผลลัพธ์การทดลองวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ในวิทยานิพนธ์เล่มนี้ ได้แก่ 1) ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หรือ logger1 2) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ หรือ logger2 3) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด หรือ logger3 และ 4) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช หรือ

logger4 ซึ่งระบบบันทึกเหตุการณ์เหล่านี้เป็นการปรับปรุงประสิทธิภาพต่อเนื่องกัน ได้แก่ logger2 ถูกปรับปรุงต่อมาจาก logger1 logger3 ถูกปรับปรุงต่อมาจาก logger2 และ logger4 ถูกปรับปรุงต่อมาจาก logger3 ตามลำดับ โดยการปรับปรุงนั้นเป็นการเพิ่มคุณสมบัติการทำงานใหม่เข้าไปในระบบบันทึกเหตุกาณ์ก่อนหน้า

1. การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

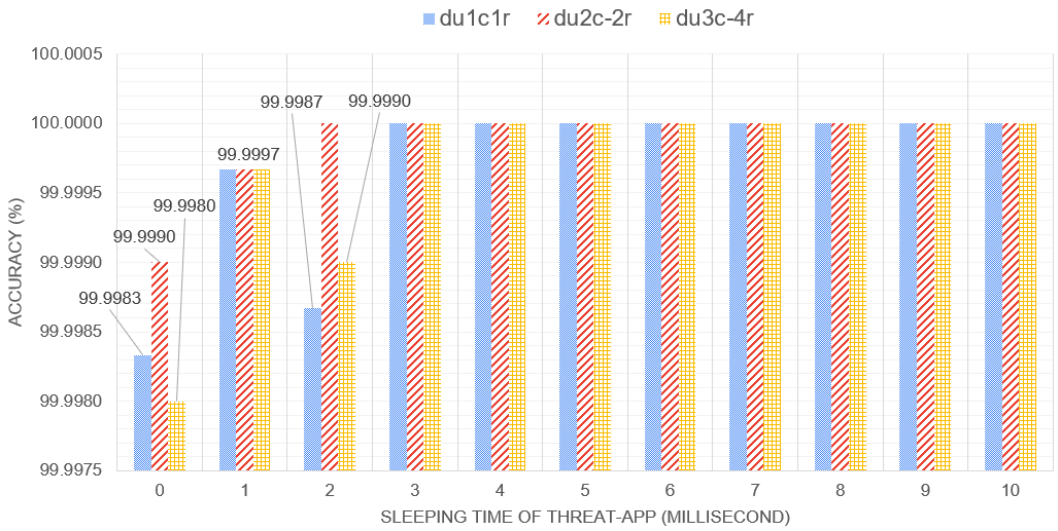
ในหัวข้อนี้จะนำเสนอผลลัพธ์การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยแบ่งออกเป็น 2 กลุ่ม คือ 1) กลุ่มที่มีการรัน domU เพียงหนึ่งตัวในหนึ่งโหนดคอมพิว และ 2) กลุ่มที่มีการรัน domU มากกว่าหนึ่งตัวพร้อมกันในหนึ่งโหนดคอมพิว โดยมีการกำหนดข้อมูลจำเพาะสำหรับสร้าง domU ตามตาราง 1 และตาราง 2 ในหัวข้อการวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยมีรายละเอียดดังต่อไปนี้

1.1 กลุ่มที่มีการรัน domU เพียงหนึ่งตัวในหนึ่งโหนดคอมพิว

จากการทดลองพบว่าค่าความถูกต้องของ logger1 ถึง logger4 มีค่าความถูกต้อง 100% เมื่อโปรเซส threat-app มีค่า sleeping time หรือ SPT อย่างน้อย 0ms ขึ้นไปสำหรับทุก domU ที่มีข้อมูลจำเพาะตามตาราง 1 ไม่แตกต่างกัน หรือหมายความว่า logger1 ถึง logger4 มีความสามารถเหมือนกัน

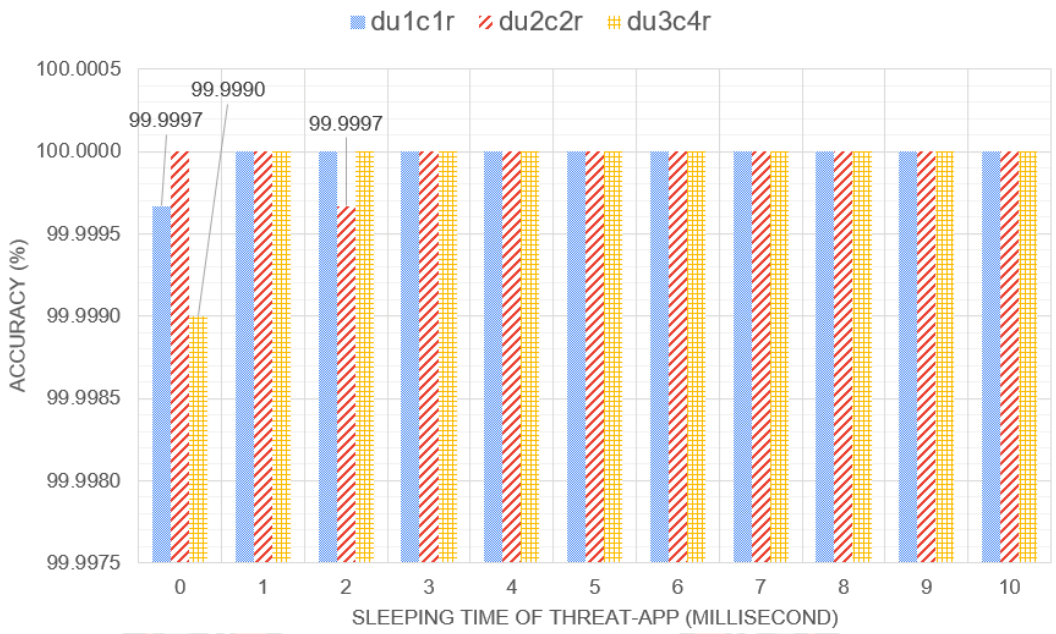
1.2 กลุ่มที่มีการรัน domU มากกว่าหนึ่งตัวพร้อมกันในหนึ่งโหนดคอมพิว

สำหรับการทดลองในกลุ่มนี้ ผู้วิจัยจะกำหนดให้มีการรัน domU จำนวนสามตัวพร้อมกันในโหนดคอมพิวเดียวกัน โดย domU ที่มีการรันพร้อมกันแต่ละตัวจะมีข้อมูลจำเพาะเหมือนกัน เช่น จำนวนคอร์ของซีพียูเสมือน จำนวนหน่วยความจำหลัก และระบบปฏิบัติการเวอร์ชันเดียวกัน โดยกราฟที่นำเสนอนั้นแกน X แสดงถึงค่าของ sleeping time หรือ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ซึ่งมีหน่วยเป็นมิลิวินาที (millisecond: ms) มีค่าเริ่มต้น 0ms ถึง 10ms และมีการเพิ่มค่าขึ้นทีละหนึ่งหน่วยตามลำดับ สำหรับแกน Y แสดงเปอร์เซ็นต์ของค่าความถูกต้องของระบบบันทึกเหตุการณ์ โดยมีรายละเอียดดังต่อไปนี้



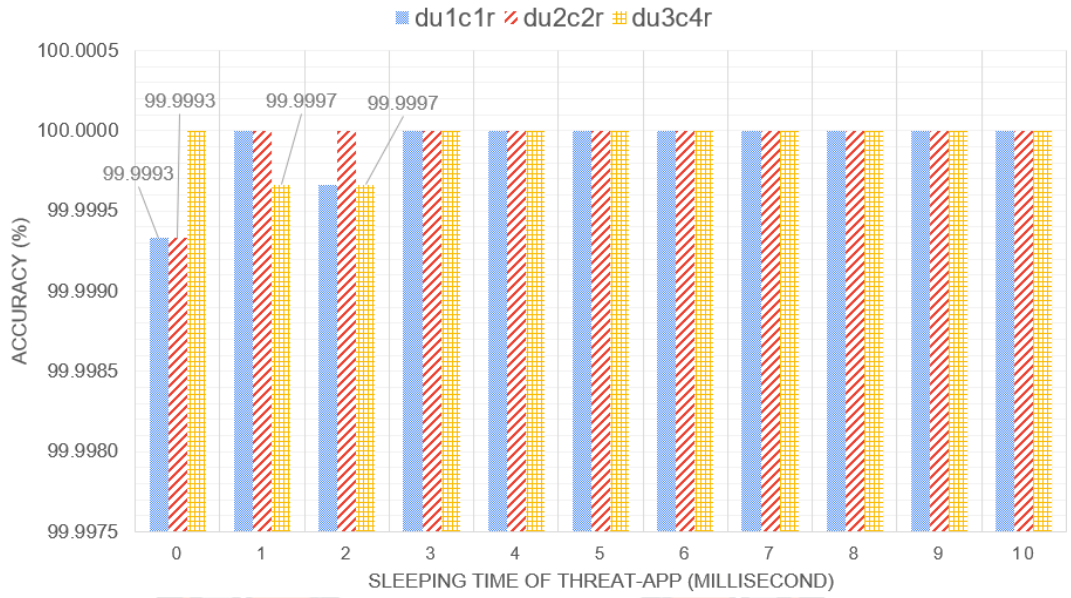
ภาพ 30 การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (logger1)

จากภาพ 30 เป็นผลลัพธ์การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (logger1) จากการทดลองพบว่าเมื่อค่า SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ที่น้อยกว่า 3ms ระบบบันทึกเหตุการณ์นี้มีการทำงานที่ไม่เสถียรซึ่งมีค่าความถูกต้องอยู่ระหว่าง 99.9980% ถึง 100% และเมื่อค่า SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ตั้งแต่ 3ms ขึ้นไประบบบันทึกเหตุการณ์นี้มีการทำงานเสถียรซึ่งมีค่าความถูกต้องที่ 100% ตลอด



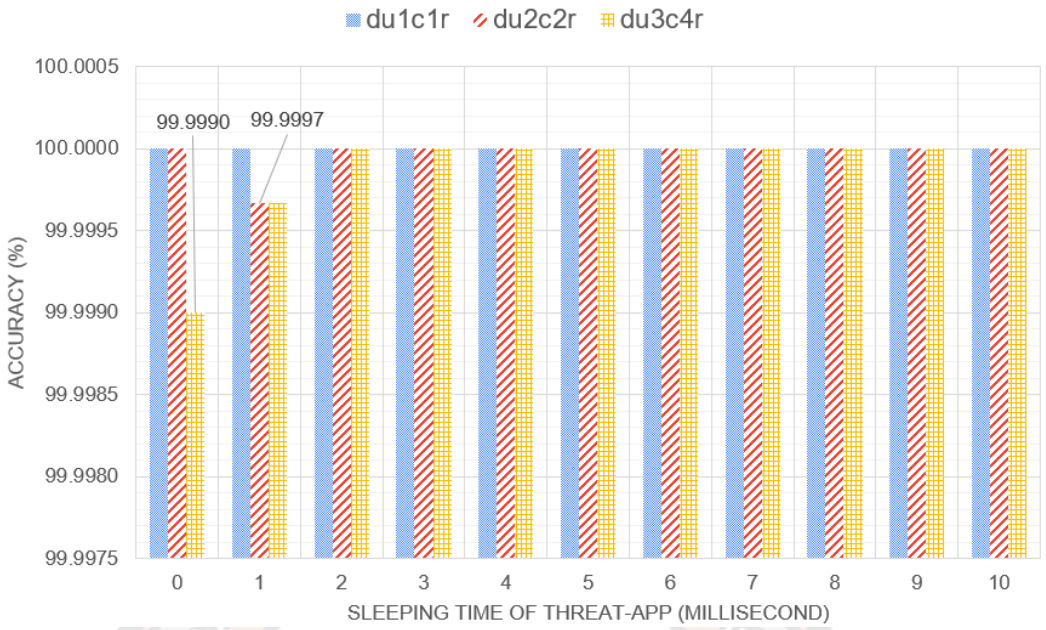
ภาพ 31 การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (logger2)

จากภาพ 31 เป็นผลลัพธ์การทดสอบของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ (logger2) จากการทดลองพบว่าเมื่อค่า SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ที่น้อยกว่า 3ms ระบบบันทึกเหตุการณ์นี้มีการทำงานที่ไม่เสถียรซึ่งมีค่าความถูกต้องอยู่ระหว่าง 99.9990% ถึง 100% และเมื่อค่า SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ตั้งแต่ 3ms ขึ้นไประบบบันทึกเหตุการณ์นี้มีการทำงานเสถียรซึ่งมีค่าความถูกต้องที่ 100% ตลอด



ภาพ 32 การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด (logger3)

จากภาพ 32 เป็นผลลัพธ์การทดสอบของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด (logger3) จากการทดลองพบว่าเมื่อค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ที่น้อยกว่า 3ms ระบบบันทึกเหตุการณ์นี้มีการทำงานที่ไม่เสถียรซึ่งมีค่าความถูกต้องอยู่ระหว่าง 99.9993% ถึง 100% และเมื่อค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ตั้งแต่ 3ms ขึ้นไประบบบันทึกเหตุการณ์นี้มีการทำงานเสถียรซึ่งมีค่าความถูกต้องที่ 100% ตลอด



ภาพ 33 การวัดค่าความถูกต้องของระบบบันทึกเหตุการณแบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช (logger4)

จากภาพ 33 เป็นผลลัพธ์การทดสอบของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช (logger4) จากการทดลองพบว่าเมื่อค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ที่น้อยกว่า 2ms ระบบบันทึกเหตุการณ์นี้มีการทำงานที่ไม่เสถียรซึ่งมีค่าความถูกต้องอยู่ระหว่าง 99.9990% ถึง 100% และเมื่อค่าของ SPT ที่กำหนดให้กับแอปพลิเคชัน threat-app ตั้งแต่ 2ms ขึ้นไประบบบันทึกเหตุการณ์นี้มีการทำงานเสถียรซึ่งมีค่าความถูกต้องที่ 100% ตลอด

2. การทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ในหัวข้อนี้นำเสนอผลลัพธ์การทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ การทดสอบประสิทธิภาพในหัวข้อนี้จะมุ่งเน้นไปที่ส่วนประกอบหลักของระบบบันทึกเหตุการณ์แต่ละตัว คือ logger หรือ logger_s ของระบบบันทึกเหตุการณ์เหล่านี้

ตาราง 3 เปรียบเทียบเวลาที่ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้องของโอเพนสแต็กใน IaaS คลาวด์แต่ละสถาปัตยกรรมใช้เพื่อการตรวจสอบหน่วยความจำเสมือนของ domU

ลำดับที่การทดสอบ	logger1 (µs)	logger2 (µs)	logger3 (µs)	logger4 (µs)
1	46595	34947	33309	16286
2	43881	34884	33372	16818
3	63719	35379	33217	15886
4	42947	35024	33631	15916
5	42745	34348	33035	15847
6	51603	36353	33777	16079
7	45427	34505	33537	15933
8	42946	34800	33021	16638
9	43700	34513	32937	16565
10	43813	34374	33426	21037
ค่าเฉลี่ย	46737.60	34912.70	33326.20	16700.50
ความเร็วที่เพิ่มขึ้น (%)	0	25.30	28.70	64.27

หมายเหตุ: logger1 หมายถึง ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

logger2 หมายถึง ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

logger3 หมายถึง ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

logger4 หมายถึง ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

ตาราง 3 เป็นตารางเปรียบเทียบเวลาที่ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์แต่ละสถาปัตยกรรมใช้เพื่อตรวจสอบหน่วยความจำเสมือนของ domU โดยตารางนี้แสดงการเปรียบเทียบการทำงานของ logger1 ถึง logger4 ที่ใช้ในการตรวจสอบหน่วยความจำเสมือนของ domU จำนวน 100 รอบต่อครั้งและมีการจับเวลาที่ logger แต่ละตัวใช้ในการดำเนินงาน โดยจะทดลองจำนวนทั้งหมด 10 ครั้ง หลังจากนั้นนำค่าที่ได้มาหาค่าเฉลี่ยของแต่ละ logger ซึ่งได้อธิบายไว้ในหัวข้อการวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ จากการทดลองพบว่า logger2 มีความเร็วในการดำเนินงานเพิ่มขึ้น 25.30% เมื่อเทียบกับ logger1 logger3 มีความเร็วในการดำเนินงานเพิ่มขึ้น 28.70% เมื่อเทียบกับ logger1 และ logger4 มีความเร็วในการดำเนินงานเพิ่มขึ้น 64.27% เมื่อเทียบกับ logger1

จากหัวข้อการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และผลลัพธ์ของการทดลองของหัวข้อนี้ซึ่งกล่าวไว้ข้างต้น ผู้วิจัยขออภิปรายสิ่งที่ได้รับจากการปรับปรุงครั้งนี้ คือ จากตาราง 3 พบว่า logger2 มีความเร็วในการดำเนินการเพิ่มขึ้น 25.30% เมื่อเปรียบเทียบกับ logger1 เนื่องจากการแยกภาระงานของ logger2 ออกเป็นสองส่วน คือ logger_s และ logger_c ทำให้ภาระงานของ logger_s ไม่ต้องดำเนินการบันทึกข้อมูลลงใน log file บนดิสก์ของตนเอง ซึ่งการติดต่อกับดิสก์ที่เป็นอุปกรณ์จัดเก็บข้อมูลที่มีความเร็วต่ำจึงช่วยเพิ่มความเร็วในการทำงานของ logger2 ได้ ทั้งนี้ logger_c จะรับภาระงานส่วนนี้แทน logger_s สำหรับ logger3 มีความเร็วในการดำเนินงานเพิ่มขึ้น 28.70% เมื่อเปรียบเทียบกับ logger1 และมีความเร็วในการดำเนินงานเพิ่มขึ้น 4.54% เมื่อเปรียบเทียบกับ logger2 เนื่องจากการปรับปรุงการทำงานของ logger3 โดยการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรดทำให้สามารถลดค่าโสหุ้ย (Overhead) ของการรันโปรแกรมและยังทำให้ขนาดของโปรแกรมเล็กลงทำให้สามารถทำงานได้เร็วขึ้น และสุดท้าย logger4 มีความเร็วในการดำเนินงานเพิ่มขึ้น 64.27% เมื่อเปรียบเทียบกับ logger1 และมีความเร็วในการดำเนินงานเพิ่มขึ้น 49.89% เมื่อเปรียบเทียบกับ logger3 เนื่องจากการปรับปรุงการทำงานของ logger4 โดยการประยุกต์ใช้ตารางแฮชช่วยในการตรวจสอบร่วมกับการทำงานของฟังก์ชัน strcmp

บทที่ 5

บทสรุป

การประมวลผลแบบกลุ่มเมฆหรือคลาวด์คอมพิวติ้ง (Cloud Computing) เป็นแบบจำลองสำหรับการใช้ทรัพยากรคอมพิวเตอร์ เช่น เครือข่าย (Network) เครื่องเซิร์ฟเวอร์ (Server) หน่วยเก็บข้อมูล (Storage) โปรแกรมประยุกต์หรือแอปพลิเคชัน (Application) และบริการอื่น ๆ ซึ่งทรัพยากรเหล่านี้สามารถเข้าถึงได้อย่างสะดวกจากอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ โดยเชื่อมต่อผ่านเครือข่ายได้จากทุกที่ การขอใช้หรือยกเลิกการใช้ทรัพยากรเหล่านี้สามารถทำได้อย่างรวดเร็วและปรับเปลี่ยนไปตามความต้องการ โดยใช้ความพยายามในการจัดการหรือติดต่อกับผู้ให้บริการ (Provider) น้อยที่สุด

อย่างไรก็ตาม CSA (2022); Ristov et al. (2013) และ Wongthai (2014) ได้อธิบายเกี่ยวความกังวลด้านความปลอดภัยของ IaaS คลาวด์ ในประเด็นเรื่องของความลับ (Confidentiality) ความถูกต้องสมบูรณ์ (Integrity) และความพร้อมใช้งาน (Availability) ของไฟล์ที่มีความสำคัญของผู้ใช้บริการ ซึ่งเป็นปัญหาสำคัญที่ขัดขวางการนำคลาวด์ไปใช้งาน ประเด็นปัญหาเหล่านี้ได้รับการพิจารณาโดยนักวิจัยหลายคน โดยเฉพาะอย่างยิ่งองค์กร Cloud Security Alliance หรือ CSA ที่ทำการวิจัย รวบรวม และระบุปัญหาเกี่ยวข้องกับภัยคุกคามต่อคลาวด์มากว่า 14 ปี มีการเผยแพร่รายงานประมาณ 23 ฉบับ และได้รับการแปลเป็นภาษาอื่นอีกห้าภาษา นักวิจัยเหล่านั้นได้ตรวจสอบวิธีการป้องกันและการบรรเทาความเสี่ยงที่เกี่ยวข้องกับประเด็นปัญหาด้านความปลอดภัยที่กล่าวไว้ข้างต้น ซึ่งผลจากการตรวจสอบสามารถเพิ่มความมั่นใจให้กับผู้ใช้บริการหรือองค์กรที่ต้องการนำคลาวด์ไปใช้งาน

สำหรับ IaaS คลาวด์หนึ่งในวิธีการบรรเทาประเด็นปัญหาดังกล่าวข้างต้น คือ ระบบบันทึกเหตุการณ์ (Logging System) ซึ่งระบบนี้สามารถช่วยบรรเทาความเสี่ยงที่เกี่ยวข้องกับปัญหาดังที่ได้กล่าวไว้ใน Auxsorn et al. (2020); Jaiboon et al. (2020); Wiriya et al. (2020); Wongthai et al. (2013b); Wongthai & Van Moorsel (2016a, 2016b) อย่างไรก็ตามงานวิจัยของ Auxsorn et al. (2020); Wiriya et al. (2020); Wongthai et al. (2013b); Wongthai & Van Moorsel (2016a, 2016b) ได้จัดเตรียมระบบบันทึกเหตุการณ์สำหรับ IaaS คลาวด์ในห้องปฏิบัติการ แทนที่จะเป็น IaaS คลาวด์ที่ใช้ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ดังนั้นเนื้อหาในวิทยานิพนธ์เล่มนี้ให้ความสำคัญกับการประยุกต์ใช้ระบบบันทึกเหตุการณ์และการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สามารถทำงาน

ใน IaaS คลาวด์ที่ใช้งานในการผลิตจริง เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine) ตามหัวข้อวัตถุประสงค์ของการศึกษา และสอดคล้องกับหัวข้อปัญหาวิจัยหลักของวิทยานิพนธ์เล่มนี้ โดยวัตถุประสงค์ของการศึกษา มีดังต่อไปนี้

1. พื่อประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อช่วยสร้างหลักฐานสำหรับตรวจสอบและระบุเกี่ยวกับบุคคลที่ต้องรับผิดชอบสำหรับภาระความรับผิด (Accountability) เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ในประเด็นปัญหาการละเมิดข้อมูล (Data Breach) เกี่ยวกับการเปิดอ่าน (View) ไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ในเครื่องเสมือน (Virtual Machine)

2. เพื่อออภิปรายผลลัพธ์การออกแบบ (Design) และการทำให้เกิดผล (Implementation) ของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ว่าทำไมยังคงสามารถทำงานในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ที่สร้างจากโอเพนสแต็ก และระบบบันทึกเหตุการณ์ดังกล่าวสามารถเป็นอีกทางเลือกหนึ่ง สำหรับการบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA

3. เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ให้สอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตามรายงานขององค์กร CSA

4. เพื่อทดสอบและประเมินประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตามรายงานขององค์กร CSA

สรุปผลการวิจัย

ในหัวข้อนี้ผู้วิจัยจะขอสรุปผลการวิจัยที่ได้ดำเนินการในวิทยานิพนธ์เล่มนี้ ซึ่งสอดคล้องกับปัญหาวิจัยหลัก เป้าหมายของงานวิจัย และวัตถุประสงค์ของการศึกษา โดยแยกเป็นประเด็นดังต่อไปนี้

1. สร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

มีวัตถุประสงค์เพื่อศึกษาองค์ประกอบ ขั้นตอนการทำงานของ IaaS คลาวด์ และระบบบันทึกเหตุการณ์ใน IaaS คลาวด์ ซึ่งการทดลองนี้ดำเนินการบนเครื่องคอมพิวเตอร์กายภาพจำนวนหนึ่งเครื่อง ซึ่งใช้ระบบปฏิบัติการ Ubuntu Server 16.04 LTS โดยขั้นตอนการติดตั้ง IaaS คลาวด์และระบบบันทึกเหตุการณ์นี้ได้ถูกอธิบายไว้ในหัวข้อการสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ และผลลัพธ์ของการทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองสร้าง IaaS คลาวด์และการติดตั้งระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของห้องปฏิบัติการ

2. สร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

มีวัตถุประสงค์เพื่อศึกษาองค์ประกอบ ขั้นตอนการทำงาน และสถาปัตยกรรมของโอเพนสแต็ก ที่ใช้สำหรับการสร้าง IaaS คลาวด์ ซึ่งระบบดังกล่าวถูกนำไปใช้ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ซึ่งการทดลองนี้ดำเนินการบนเครื่องคอมพิวเตอร์กายภาพจำนวนสองเครื่องและมีการเชื่อมต่อกันผ่านเครือข่าย และใช้ระบบปฏิบัติการ Ubuntu Server 16.04 LTS โดยขั้นตอนการติดตั้ง IaaS คลาวด์โดยใช้ซอฟต์แวร์โอเพนสแต็กได้ถูกอธิบายไว้ในหัวข้อการสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และผลลัพธ์ของการทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์การทดลองสร้างสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

3. การติดตั้งและทดสอบการทำงานของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

มีวัตถุประสงค์เพื่อประยุกต์ใช้ระบบบันทึกเหตุการณ์ให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และทดสอบการทำงานของระบบบันทึกเหตุการณ์ยังสามารถตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ตามที่กำหนดไว้ได้ โดยขั้นตอนการออกแบบและติดตั้งระบบบันทึกเหตุการณ์ในหัวข้อนี้ได้อธิบายไว้ในหัวข้อการสร้างระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และผลลัพธ์ของการทดลองในหัวข้อนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่าระบบบันทึกเหตุการณ์นี้สามารถตรวจสอบและบันทึกข้อมูลของเหตุการณ์ต่าง ๆ ได้ตามวัตถุประสงค์ และตอบโจยท์ปัญหาวิจัยหลักเกี่ยวกับการประยุกต์ใช้ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ได้ และมีการออกแบบและการทำให้เกิดผลของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพสนแต็กใน IaaS คลาวด์ รวมถึงมีการแสดงผลลัพธ์และ

การอภิปรายเกี่ยวกับระบบบันทึกเหตุการณ์ในหัวข้อนี้เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA สุดท้ายได้รวบรวมข้อมูลที่ได้นำเสนอเป็นบทความวิชาการ

4. การสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

มีวัตถุประสงค์เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ให้สามารถทำงานสอดคล้องกับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ รวมถึงการทดสอบการทำงานของระบบบันทึกเหตุการณ์นี้ ซึ่งสภาพแวดล้อมนี้มีการเชื่อมต่อผ่านระบบเครือข่าย ดังนั้นการปรับปรุงระบบบันทึกเหตุการณ์ในหัวข้อนี้จะประยุกต์ใช้เทคนิคการเขียนโปรแกรมซ็อกเก็ต เพื่อแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์ที่อยู่ในเครือข่ายเดียวกันได้ โดยขั้นตอนการออกแบบและทำให้เกิดผลได้อธิบายไว้ในหัวข้อการสร้างระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ และผลลัพธ์ของการทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่าสามารถปรับปรุงระบบบันทึกเหตุการณ์ในหัวข้อนี้ให้ทำงานแบบรวมศูนย์และสอดคล้องกับการทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กได้ และตอบโจทย์วัตถุประสงค์ของการศึกษาเกี่ยวกับการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุณ์ อีกทั้งการปรับปรุงครั้งนี้สามารถช่วยกระจายปริมาณงาน (Workload) ของระบบบันทึกเหตุการณ์และส่งผลให้ระบบบันทึกเหตุการณ์นี้ทำงานได้เร็วขึ้น สามารถรองรับการขยายโหนดคอมพิว (Compute Node) สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์และยังง่ายต่อการบริการจัดการสำหรับการเรียกใช้งานระบบบันทึกเหตุการณ์นี้เมื่อเปรียบเทียบกับระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์เดิม

5. ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

มีวัตถุประสงค์เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ เพื่อให้ระบบบันทึกเหตุการณ์นี้สามารถทำงานแบบขนาน (Parallel) ได้โดยใช้หลักการของการโปรแกรมแบบขนานด้วยโมเดลเธรด รวมถึงการทดสอบระบบบันทึกเหตุการณ์นี้ โดยขั้นตอนออกแบบและการทำให้เกิดผลสำหรับการปรับปรุงประสิทธิภาพได้อธิบายไว้ในหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียน

โปรแกรมแบบขนานด้วยโมเดลเธรด และผลลัพธ์ของการทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่าสามารถปรับปรุงระบบบันทึกเหตุการณ์ในหัวข้อมูนี้ทำงานแบบขนานและสอดคล้องกับการทำงานในสภาพแวดล้อมของโอเพนสแต็กได้ และตอบโจทย์วัตถุประสงค์ของการศึกษาเกี่ยวกับการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์ อีกทั้งการปรับปรุงครั้งนี้ ทำให้ระบบบันทึกเหตุการณ์ในหัวข้อนี้เพิ่มความสามารถตรวจสอบหน่วยความจำเสมือนของ domU ได้มากกว่าหนึ่งตัวที่อยู่ในโหนดคอมพิวเดียวกันพร้อม ๆ กันได้ อีกทั้งเพิ่มความสะดวกในการทำงานร่วมกันระหว่าง logger_c และ logger_s กรณีการระบุหมายเลขพอร์ทหรือหมายเลขซ็อกเก็ตของ logger_s บนโหนดคอมพิวเดียวกันสามารถระบุเพียงหมายเลขเดียวเท่านั้นสำหรับรองรับการร้องของการเชื่อมต่อจาก logger_c หลาย ๆ ครั้ง

6. ปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

มีวัตถุประสงค์เพื่อปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด โดยปรับปรุงอัลกอริทึมสำหรับการตรวจสอบข้อมูลในหน่วยความจำเสมือนของ domU ด้วยการประยุกต์ใช้ตารางแฮช เพื่อเพิ่มความเร็วในการตรวจสอบหน่วยความจำเสมือนของ domU โดยขั้นตอนออกแบบและการทำให้เกิดผลสำหรับการปรับปรุงประสิทธิภาพได้อธิบายไว้ในหัวข้อการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช และผลลัพธ์ของการทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่าระบบบันทึกเหตุการณ์ในหัวข้อนี้สามารถลดค่า SPT หรือ Sleeping Time ที่กำหนดให้น threat-app (SPT จะแทนความหมายถึงระยะเวลาที่ threat-app เปิดอ่านไฟล์ที่มีความสำคัญของผู้ใช้บริการที่อยู่ใน domU) ลงได้ 1ms เมื่อเปรียบเทียบกับการทำงานของระบบบันทึกเหตุการณ์ในหัวข้อก่อนหน้าในสภาพแวดล้อมเดียวกัน ซึ่งหมายความว่าระบบบันทึกเหตุการณ์ในหัวข้อนี้ทำงานได้เร็วขึ้น

7. ทำการวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

มีวัตถุประสงค์เพื่อวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ทั้งหมดในวิทยานิพนธ์เล่มนี้ ตามวัตถุประสงค์การศึกษา โดยแนวทางการวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ในหัวข้อนี้ได้ถูกอธิบายไว้ในหัวข้อการวัดค่าความถูกต้องและทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ โดยแบ่งเป็น 2 ส่วน คือ 1) การวัดค่าความถูกต้องของระบบบันทึกเหตุการณ์ โดยให้ความสำคัญสำหรับค่าความถูกต้องที่ 100% และ 2) การทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์ โดยให้ความสำคัญที่ความเร็วในการตรวจสอบหน่วยความจำเสมือนของ domU ของระบบบันทึกเหตุการณ์แต่ละแบบที่เสนอไว้ สำหรับผลลัพธ์การทดลองนี้ได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองการวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่าสามารถประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยก่อนหน้าให้สามารถทำงานสำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ได้ และการปรับปรุงระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์แต่ละแบบที่เสนอในวิทยานิพนธ์เล่มนี้ มีความสามารถและความเร็วเพิ่มขึ้นตามลำดับ

อภิปรายผล

ในหัวข้อนี้ผู้วิจัยจะเสนอการอภิปรายที่เกี่ยวข้องกับระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์แต่ละแบบที่เสนอในวิทยานิพนธ์เล่มนี้ ได้แก่ 1) ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ 2) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ 3) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด และ 4) ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช ตามลำดับ ดังมีรายละเอียดต่อไปนี้

1. ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ระบบบันทึกเหตุการณ์ในหัวข้อนี้เป็นการประยุกต์ใช้ระบบบันทึกเหตุการณ์จากงานวิจัยของ Auxsorn et al. (2020) และ Wongthai (2014) ให้สามารถทำงานสำหรับ

สภาพแวดล้อมล้องของโอเพนสแต็กใน IaaS คลาวด์ เพื่อบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA โดยมีประเด็นที่จะอภิปรายดังต่อไปนี้

1.1 ระบบบันทึกเหตุการณ์ในหัวข้อนี้สามารถทำงานได้ใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็กที่มีโหนดคอมพิวเดียว

แม้ว่าระบบบันทึกเหตุการณ์จากงานก่อนหน้า (Auxsorn et al., 2020; Wongthai, 2014) สามารถทำงานได้ดีในสภาพแวดล้อมห้องปฏิบัติการหรือภาพ 1 แต่ยังไม่เคยนำระบบบันทึกเหตุการณ์ดังกล่าวไปใช้กับ IaaS คลาวด์ในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) หรือภาพ 6 หลังจากการทดลองประยุกต์ใช้ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ดังแสดงในภาพ 6 ผลลัพธ์การทดลองในภาพ 19 และ ภาพ 20 เป็นเครื่องพิสูจน์ว่าระบบบันทึกเหตุการณ์จากงานก่อนหน้า (Auxsorn et al., 2020; Wongthai, 2014) สามารถนำไปใช้ใน IaaS คลาวด์ที่สร้างจากโอเพน สแต็กที่สร้างขึ้นใหม่หรือภาพ 6 ดังนั้นระบบบันทึกเหตุการณ์ในภาพ 6 จึงมีความสามารถในการตรวจจับและบันทึกข้อมูลของโปรเซสที่เป็นอันตรายซึ่งสาเหตุของภัยคุกคามตาม CSA จากนั้นระบบบันทึกเหตุการณ์ในภาพ 6 จะสามารถจัดเก็บข้อมูลและบันทึกข้อมูลลงในล็อกไฟล์ (Log File) รูปแบบของคำสั่ง logger ของระบบบันทึกเหตุการณ์นี้ในหัวข้อผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ยังสามารถแก้ไขเพื่อตรวจจับหาโปรเซสใหม่และไฟล์ใด ๆ ที่โปรเซสใหม่อ่านหรือดำเนินการอื่น ๆ กับไฟล์ เช่น การลบไฟล์ (Chan-In & Wongthai, 2016; Wongthai, 2014) อย่างไรก็ตามผู้วิจัยเชื่อว่าสถาปัตยกรรมระบบบันทึกเหตุการณ์ที่เสนอหรือภาพ 6 สามารถนำไปใช้และทำงานใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็ก แล้วระบบบันทึกเหตุการณ์นี้สามารถเป็นหนึ่งในแนวทางบรรเทาความเสี่ยงที่เกี่ยวข้องกับภัยคุกคามตาม CSA ใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็กที่มีการนำไปใช้ในการผลิตจริง และสามารถมีโหนดคอมพิวมากกว่าหนึ่งโหนดได้

1.2 เหตุใดระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ยังคงสามารถทำงานใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็กในโลกแห่งความเป็นจริง (Real-world) ที่มีโหนดคอมพิวมากกว่าหนึ่งโหนด

ในภาพ 6 มีโหนดคอมพิวเพียงหนึ่งโหนดสำหรับการทดลองนี้ อย่างไรก็ตามเมื่อมีความต้องการเพิ่มจำนวนโหนดคอมพิวจากหนึ่งโหนดเป็นสองหรือมากกว่านั้นใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็กในโลกแห่งความเป็นจริง และจากการทดลองในภาพ 6 รวมทั้งผลลัพธ์จากภาพ 19 และ ภาพ 20 ผู้วิจัยเชื่อว่าระบบบันทึกเหตุการณ์นี้ยังคงสามารถทำงานใน IaaS คลาวด์

ที่สร้างจากโอเพนสแต็ก ที่สร้างขึ้นใหม่ในโลกแห่งความเป็นจริงที่มีโหนดคอมพิวมากกว่าหนึ่งโหนด และระบบบันทึกเหตุการณ์ในหัวข้อนี้ยังสามารถตรวจจับและบันทึกข้อมูลโปรเซสที่เป็นอันตรายตามที่ได้กล่าวไว้ในหัวข้ออภิปรายผล หัวข้อย่อยระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ นี้เป็นเพราะการเพิ่มจำนวนโหนดคอมพิวใหม่ใน Iaas คลาวด์ที่สร้างจากโอเพนสแต็ก ไม่ส่งผลกระทบต่อการทำงานของโหนดคอมพิวที่มีอยู่หรือกล่องเส้นประทางด้านขวาของภาพ 6 เพราะว่าโหนดคอมพิวแต่ละโหนดใน IaaS คลาวด์เป็นอิสระจากกัน ยิ่งไปกว่านั้นระบบบันทึกเหตุการณ์ในโหนดคอมพิวที่เพิ่มใหม่จะมีฟังก์ชันเดียวกันกับระบบบันทึกเหตุการณ์ในโหนดคอมพิวก่อนหน้าในภาพ 6 ดังนั้นผู้วิจัยเชื่อว่าระบบบันทึกเหตุการณ์ในภาพ 6 ซึ่งมีล็อกไฟล์ที่สร้างจาก logger และ LibVMI ยังสามารถบรรเทาความเสี่ยงเกี่ยวกับภัยคุกคามตาม CSA ใน IaaS คลาวด์ที่สร้างจากโอเพนสแต็กในสภาพแวดล้อมที่ใช้ในการผลิตจริง (Real-world Production Environment) ไม่ใช่เพียงแต่ในสภาพแวดล้อมจำลองหรือห้องปฏิบัติการเท่านั้น

1.3 ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์สามารถทำงานร่วมกับไฮเปอร์ไวเซอร์ได้หลากหลาย

ระบบบันทึกเหตุการณ์ในภาพ 6 ยังสามารถทำงานร่วมกับไฮเปอร์ไวเซอร์ต่าง ๆ ได้หลากหลาย ซึ่ง Sefraoui et al. (2012) ได้กล่าวว่า โอเพนสแต็กสามารถรองรับไฮเปอร์ไวเซอต์ได้หลากหลาย เช่น VMware Elastic Sky X หรือ ESX Microsoft Hyper-V Linux Container หรือ LXC Kernel-based Virtual Machine หรือ KVM Quick Emulator หรือ QEMU XEN และ XenServer เป็นต้น ด้วยเหตุนี้จึงทำให้โอเพนสแต็กถูกนำไปใช้ในการสร้าง IaaS คลาวด์ด้วยไฮเปอร์ไวเซอร์ที่หลากหลาย สำหรับการทดลองนี้ผู้วิจัยเลือกใช้ไฮเปอร์ไวเซอร์ของโหนดคอมพิวในภาพ 6 คือ Xen เท่านั้น อย่างไรก็ตาม Wang, Estrada, Pham, Kalbarczyk & Iyer (2015) ได้กล่าวว่า ไฮเปอร์ไวเซอร์ของโหนดคอมพิวสามารถเปลี่ยนจาก Xen เป็น KVM หรือ QEMU ตามที่กล่าวไว้ข้างต้น นอกจากนี้ระบบบันทึกเหตุการณ์หรือ logger ในภาพ 6 ทำงานร่วมกับ LibVMI ที่สามารถทำงานร่วมกับ KVM หรือ QEMU ได้ไม่ใช่เพียงแค่ Xen (Payne, Maresca, Lengyel, & Saba, 2015) ดังนั้น logger ยังคงสามารถตรวจจับและบันทึกข้อมูลของโปรเซสที่ต้องการตรวจสอบหรือที่เป็นอันตรายได้ ทั้งในสภาพแวดล้อมของ KVM หรือ QEMU ไม่ใช่เฉพาะ Xen สิ่งนี้ทำให้สามารถขยายระบบบันทึกเหตุการณ์นี้ให้ทำงานในไฮเปอร์ไวเซอร์อย่างน้อยสองตัว คือ KVM และ QEMU ใน IaaS คลาวด์ที่สร้างจากซอฟต์แวร์โอเพนซอร์ส เช่น โอเพนสแต็ก ดังนั้นระบบบันทึกเหตุการณ์นี้ควรทำงานได้กับ ESX Hyper-V LXC และ XenServer ด้วย

1.4 สถาปัตยกรรมของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพน สแต็กใน IaaS คลาวด์เป็นทางเลือกเพิ่มเติมนอกเหนือจากคุณสมบัติด้านความปลอดภัยของโอเพนสแต็ก เพื่อบรรเทาความเสี่ยงเกี่ยวกับภัยคุกคามตาม CSA

ตามที่กล่าวไว้ในหัวข้ออภิปรายผล หัวข้อย่อยระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ รูปแบบของคำสั่ง logger ของระบบบันทึกเหตุการณ์นี้ สามารถแก้ไขเพื่อตรวจจับชื่อโปรเซสที่เป็นอันตรายอ่านไฟล์ที่มีความสำคัญใน domU ตามที่ CSA (2019) ระบุว่าภัยคุกคามแรกหรือที่รุนแรงที่สุด คือ การละเมิดข้อมูล ซึ่งหมายถึง เมื่อมีเหตุการณ์เกิดขึ้นกับไฟล์ข้อมูลที่มีความสำคัญใน IaaS คลาวด์หรือแม้แต่คลาวด์ประเภทอื่น ๆ เช่น PaaS หรือ SaaS ข้อมูลที่มีความละเอียดอ่อน (Sensitive Data) หรือมีความสำคัญใน IaaS คลาวด์ เช่น ข้อมูลด้านสุขภาพและการเงิน ไม่ควรถูกเปิดเผย ถูกขโมย หรือถูกใช้งานโดยผู้ใช้ที่ไม่ได้รับอนุญาต (CSA, 2019) โอเพนสแต็กเป็นซอฟต์แวร์โอเพนซอร์สสำหรับสร้างระบบการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการโครงสร้างพื้นฐานทางคอมพิวเตอร์ ข้อมูลในคลาวด์นี้อาจได้รับผลกระทบจากภัยคุกคามตาม CSA ที่กล่าวถึงในหัวข้อปัญหาภัยคุกคามของ IaaS คลาวด์ สำหรับการอภิปรายนี้ Benjamin et al. (2017) ได้กล่าวว่าโอเพนสแต็ก IaaS คลาวด์มีคุณสมบัติสามประการอยู่แล้วในการรักษาความปลอดภัยของข้อมูลบนคลาวด์นี้ อย่างไรก็ตามคุณสมบัติเหล่านี้อาจยังมีข้อเสีย ตามที่จะกล่าวถึงทีละรายการในที่นี้ 1) คุณสมบัติแรก คือ การจัดการคีย์ (Key Management) ซึ่งช่วยจัดการกลไกห้าอย่างเพื่อรักษาความลับของข้อมูลในโอเพนสแต็ก IaaS คลาวด์ที่สร้างขึ้นใหม่ คุณสมบัตินี้อาจจะลดประสิทธิภาพของเซิร์ฟเวอร์การจัดการคีย์ของคลาวด์นี้ เหตุผลมีดังต่อไปนี้ ผู้วิจัยขออภิปรายเฉพาะสองกลไกแรกในการวิจัยนี้เพื่อวัตถุประสงค์เกี่ยวกับการลดประสิทธิภาพ กลไกแรก คือ การสร้างคีย์เพื่อใช้ในการเข้ารหัสข้อมูลในโอเพนสแต็ก IaaS คลาวด์ และ กลไกที่สอง คือ การจัดเก็บและค้นคืนคีย์ที่สร้างขึ้นซึ่งในการเข้ารหัสและถอรหัส รวมถึงการสร้างใบรับรองของข้อมูลนี้ในระบบคลาวด์นี้ Benjamin et al. (2017) ยังได้กล่าวอีกว่า กลไกทั้งสองที่กล่าวถึงข้างต้นต้องเผชิญกับสองประเด็นหลัก ประเด็นแรก คือ เมื่อมีคำร้องขอพร้อมกันมากเกินไปจากคอมพิวเตอร์ของผู้ใช้บริการผ่านอินเทอร์เน็ตไปยังเซิร์ฟเวอร์การจัดการคีย์ คำร้องขอเหล่านี้ยังต้องการกลไกการสร้างคีย์และค้นคืนคีย์พร้อมกันหลายตัว ซึ่งอาจทำให้เกิดปัญหาคอขวด (Bottleneck) ของเซิร์ฟเวอร์การจัดการคีย์ ประเด็นที่สอง คือ กลไกการสร้างคีย์นั่นคือกระบวนการคำนวณอย่างเข้มข้นบนเซิร์ฟเวอร์การจัดการคีย์

ดังนั้น คุณสมบัติแรกในการรักษาความปลอดภัยของข้อมูลในโอเพนสแต็ก IaaS คลาวด์ซึ่งอาศัยการจัดการคีย์อาจลดประสิทธิภาพของเซิร์ฟเวอร์การจัดการคีย์ของคลาวด์นี้ คุณสมบัติที่สอง คือ การเข้ารหัสหน่วยเก็บข้อมูลแบบบล็อก (Block Storage Unit) ซึ่งหน่วยเก็บข้อมูลนี้เป็นวิธีการจัดเก็บข้อมูลในสภาพแวดล้อม Storage-area Network หรือ SAN สำหรับโอเพนสแต็ก IaaS คลาวด์จำเป็นต้องใช้ SAN และข้อมูลของคลาวด์นี้จะถูกจัดเก็บไว้ใน Volume หรือ Block การเข้ารหัสข้อมูลแบบ Block นี้จะทำงานที่โหนดคอมพิวหรือกล่องเส้นประด้านขวาของภาพ 6 การเข้ารหัสนี้อาจก่อให้เกิดข้อเสียเพราะหน่วยจัดเก็บข้อมูลแบบบล็อกไม่สามารถใช้ประโยชน์จากฟังก์ชันการบีบอัดข้อมูลได้ ซึ่งฟังก์ชันการบีบอัดข้อมูลเป็นหนึ่งในฟังก์ชันหลักเพื่อลดขนาดของข้อมูลได้ถึง 90% ในพื้นที่จัดเก็บของผู้ให้บริการ (Benjamin et al., 2017) หากไม่มีฟังก์ชันนี้ผู้ให้บริการอาจต้องลงทุนงบประมาณเพิ่มขึ้นสำหรับหน่วยจัดเก็บข้อมูลของตน คุณสมบัติสุดท้าย คือ ความถูกต้องสมบูรณ์ของอิมเมจ (Image Integrity) ซึ่งหมายถึง การตรวจสอบความสมบูรณ์ของไฟล์อิมเมจของ VM หรือ domU ว่าไฟล์นี้ถูกแก้ไขโดยโค๊ดอันตรายหรือถูกบุกรุกโดยผู้โจมตีหรือไม่ (ไฟล์อิมเมจของ VM คือ ไฟล์ที่สามารถนำมาเรียกใช้งานแล้วจะกลายเป็นเครื่องเสมือนหรือ domU) โดยที่ไฟล์นี้อยู่ในโหนดคอนโทรเลอร์หรือกล่องเส้นประด้านซ้ายของภาพ 6 อย่างไรก็ตาม Benjamin et al. (2017) ได้ทำการวัดค่าโสหุ้ย (Overhead) ของความถูกต้องสมบูรณ์ของอิมเมจตามระยะเวลาที่ใช้ในการเริ่มทำงานของ domU โดยมีและไม่มีการตรวจสอบความถูกต้องสมบูรณ์ของอิมเมจ ซึ่งพวกเขาพบว่าระยะเวลาเปิดไฟล์อิมเมจที่มีการตรวจสอบความถูกต้องสมบูรณ์ของอิมเมจสูงกว่าระยะเวลาเปิดไฟล์อิมเมจที่ไม่มีการความถูกต้องสมบูรณ์ของอิมเมจ ทั้งนี้เนื่องจากการเปิดไฟล์อิมเมจที่มีการความถูกต้องสมบูรณ์ของอิมเมจต้องใช้กระบวนการคำนวณขนาดใหญ่เพื่อตรวจสอบความปลอดภัยของไฟล์อิมเมจ ดังนั้นคุณสมบัตินี้อาจลดความเร็วของการดำเนินการโดยรวมในโอเพนสแต็ก IaaS คลาวด์ และอาจส่งผลต่อการทำงานของ domU ในโครงสร้างพื้นฐานของคลาวด์นี้ช้าลง

โอเพนสแต็ก IaaS คลาวด์มีคุณสมบัติสามประการอยู่แล้วในการรักษาความปลอดภัยของข้อมูลบนคลาวด์นี้ อย่างไรก็ตามคุณสมบัติเหล่านี้อาจยังมีข้อเสียอยู่ ระบบบันทึกเหตุการณ์ที่ได้จากหัวข้อสามารถเป็นทางเลือกเพิ่มเติมนอกเหนือจากคุณสมบัติด้านความปลอดภัยของโอเพนสแต็ก เพื่อบรรเทาความเสี่ยงเกี่ยวกับภัยคุกคามร้ายแรงอันดับต้น ๆ ของ CSA

1.5 สถาปัตยกรรมของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ไม่มีกลไกการเข้ารหัสและถอดรหัส และมีการคำนวณที่น้อยกว่า

ระบบบันทึกเหตุการณ์ในหัวข้อนี้สามารถทำงานในสภาพแวดล้อมของโอเพนสแต็ก IaaS คลาวด์ได้ และมีความสามารถในการตรวจจับและบันทึกข้อมูลของโปรเซสที่ต้องการตรวจสอบหรือไฟล์สำคัญของผู้ใช้บริการใน domU จากนั้นก็จัดเก็บข้อมูลบันทึกลงล็อกไฟล์ซึ่งเนื้อหาของล็อกไฟล์สามารถนำมาวิเคราะห์หรือตรวจสอบหาสาเหตุที่อาจเกิดขึ้นได้ เช่นเดียวกับที่ทำในงานวิจัยของ Chan-In & Wongthai (2016); Wongthai (2014) ดังนั้นผู้วิจัยเชื่อว่าระบบบันทึกเหตุการณ์นี้สามารถเป็นทางเลือกเพิ่มเติมเพื่อบรรเทาความเสี่ยงที่เกี่ยวกับภัยคุกคามตาม CSA ระบบบันทึกเหตุการณ์นี้สามารถทำงานได้โดยไม่ใช้เทคนิคการเข้ารหัสและถอดรหัส เช่นเดียวกับคุณสมบัติสามประการเพื่อรักษาความปลอดภัยของข้อมูลในสภาพแวดล้อมของโอเพนสแต็กตามที่กล่าวไว้ด้านบน สิ่งนี้ทำให้ระบบบันทึกเหตุการณ์นี้มีการคำนวณที่น้อยกว่า

2. ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

ระบบบันทึกเหตุการณ์ในหัวข้อนี้เป็นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ให้สามารถดำเนินการแบบรวมศูนย์ได้ โดยการประยุกต์ใช้การเขียนโปรแกรมแบบซ็อกเก็ต เพื่อทำให้การจัดการระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ได้ง่ายขึ้น และได้อภิปรายผลการวิจัยไว้ในส่วนท้ายของหัวข้อผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

3. ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

ระบบบันทึกเหตุการณ์ในหัวข้อนี้เป็นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ให้สามารถดำเนินการแบบขนานได้ โดยการประยุกต์ใช้การเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด และได้อภิปรายผลการวิจัยไว้ในส่วนท้ายของหัวข้อผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด

4. ระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

ระบบบันทึกเหตุการณ์ในหัวข้อนี้เป็นการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรด เพื่อเพิ่มความเร็วในการตรวจสอบหน่วยความจำเสมือน

ของ domU โดยการประยุกต์ใช้ตารางแฮช และได้อภิปรายผลการวิจัยไว้ในส่วนท้ายของหัวข้อ ผลลัพธ์ของการทดลองระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

5. การวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์

จากผลลัพธ์ของการวัดค่าความถูกต้องและการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ซึ่งเป็นการทดสอบประสิทธิภาพของระบบบันทึกเหตุการณ์แต่ละสถาปัตยกรรมในวิทยานิพนธ์เล่มนี้ โดยมุ่งเน้นไปที่ส่วนประกอบหลักของระบบบันทึกเหตุการณ์แต่ละตัว คือ logger หรือ logger_s ของระบบบันทึกเหตุการณ์เหล่านี้ในเรื่องของเวลาที่ระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์แต่ละสถาปัตยกรรมใช้เพื่อตรวจสอบหน่วยความจำหลักของ domU จาก

ตาราง 3 พบว่า logger2 มีความเร็วในการดำเนินงานเพิ่มขึ้น 25.30% เมื่อเทียบกับ logger1 เนื่องจากการแยกภาระงานของระบบบันทึกเหตุการณ์ออกเป็นสองส่วน คือ logger_s และ logger_c ทำให้ภาระงานของ logger_s ไม่ต้องดำเนินการบันทึกข้อมูลลงใน log file บนดิสก์ของตนเองซึ่งการติดต่อกับดิสก์ที่เป็นอุปกรณ์จัดเก็บข้อมูลที่มีความเร็วต่ำ จึงช่วยเพิ่มความเร็วในการทำงานของ logger2 ได้ ทั้งนี้ logger_c จะรับภาระงานส่วนนี้แทน logger_s logger3 มีความเร็วในการดำเนินงานเพิ่มขึ้น 28.70% เมื่อเทียบกับ logger1 และความเร็วในการดำเนินงานเพิ่มขึ้น 3.40% เมื่อเทียบกับ logger2 เนื่องจากการปรับปรุงการทำงานของระบบบันทึกเหตุการณ์โดยการเขียนโปรแกรมแบบขนานด้วยโมเดลเธรดทำให้ลดค่าโสหุ้ย (Overhead) และยังทำให้ขนาดโปรแกรม logger3 มีขนาดเล็กลงซึ่งทำให้โปรแกรมสามารถทำงานได้เร็วขึ้น logger4 มีความเร็วในการดำเนินงานเพิ่มขึ้น 64.27% เมื่อเทียบกับ logger1 และความเร็วในการดำเนินงานเพิ่มขึ้น 35.57% เมื่อเทียบกับ logger3 เนื่องจากการปรับปรุงอัลกอริทึมของระบบบันทึกเหตุการณ์โดยการประยุกต์ใช้ตารางแฮชเพื่อช่วยตรวจสอบร่วมกับของการทำงานของฟังก์ชัน strcmp ซึ่งได้อธิบายไว้ในหัวข้อผลลัพธ์ของการทดลองการปรับปรุงประสิทธิภาพของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์บนพื้นฐานตารางแฮช

ข้อเสนอแนะ

จากการออกแบบ ทดสอบ และปรับปรุงประสิทธิภาพของสถาปัตยกรรมของระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ผู้วิจัยมีข้อเสนอแนะในการพัฒนาระบบบันทึกเหตุการณ์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ ให้มีประสิทธิภาพเพิ่มขึ้นและสามารถนำไปใช้งานในเชิงพาณิชย์ได้ในอนาคต

1. การเพิ่มรายละเอียดของข้อมูลที่ต้องการบันทึกในล็อกไฟล์ จากการทดลองในวิทยานิพนธ์เล่มนี้รายละเอียดข้อมูลที่ต้องการบันทึกในล็อกไฟล์ยังมีรายละเอียดเพียงเล็กน้อยเท่านั้น ทั้งนี้หากต้องการนำไปใช้ในเชิงพาณิชย์ได้ จำเป็นต้องเพิ่มข้อมูลรายละเอียดให้มากขึ้น เช่น ตำแหน่งที่ตั้งของไฟล์ในเครื่องเสมือน ข้อมูลการดำเนินการที่เกี่ยวข้องกับไฟล์ เป็นต้น

2. การแยกปริมาณงานของโหนดคอนโทรเลอร์ ออกจากระบบบันทึกเหตุการณ์แบบรวมศูนย์ เพื่อให้โหนดคอนโทรเลอร์ดำเนินการเฉพาะงานที่เป็นหน้าที่หลักของตนเองเท่านั้น ซึ่งจะส่งผลให้ประสิทธิภาพโดยรวมของโหนดคอนโทรเลอร์ดีขึ้น

3. การสร้างส่วนต่อประสานของระบบบันทึกเหตุการณ์แบบรวมศูนย์สำหรับสภาพแวดล้อมของโอเพนสแต็กใน IaaS คลาวด์ จากการทดลองในวิทยานิพนธ์เล่มนี้ การควบคุมและการเรียกใช้คำสั่งล็อกเกอร์ยังคงใช้วิธีการคีย์คำสั่ง ดังนั้นเพื่อให้ง่ายต่อการนำไปใช้ จำเป็นจะต้องมีส่วนต่อประสานของระบบบันทึกเหตุการณ์นี้ เช่น ระบบบริการจัดการผ่านเว็บแอปพลิเคชัน

บรรณานุกรม

ปกรณ์ จันทร์อินทร์. (2561). *วิธีบันทึกเหตุการณ์บนการประมวลผลแบบกลุ่มเมฆประเภทการให้บริการแพลตฟอร์ม* (วิทยานิพนธ์ปริญญาดุษฎีบัณฑิต). พิษณุโลก: มหาวิทยาลัยนเรศวร.

Aalam, Z., Kumar, V., & Gour, S. (2021). A review paper on hypervisor and virtual machine security. *Journal of Physics: Conference Series, 1950*(1), 012027.

Albaroodi, H., Manickam, S., & Bawa, P. S. (2014). Critical Review of OpenStack Security: Issues and Weaknesses. *Journal of Computer Science, 10*(1), 23-33.

Alsuwaiyel, M. H. (2021). *Alsuwaiyel, M. H. (2021). Algorithms: Design techniques and analysis* (2nd ed). Toh Tuck Link, Singapore: Word Scientific Publishing.

Auxsorn, T., Wongthai, W., Porka, T., & Jaiboon, W. (2020). The Accuracy Measurement of Logging Systems on Different Hardware Environments in Infrastructure as a Service Cloud. *ICIC Express Letters, Part B: Applications, An International Journal of Research and Surveys, 11*(5), 427 – 437.

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing, 1*(1), 11-33.

Barney, B., Livermore Computing (Retired), & Frederick, D. (2019). *Introduction to Parallel Computing Tutorial.* Retrieved August 18, 2019, from https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial#:~:text=In%20the%20simplest%20sense%2C%20parallel,to%20a%20series%20of%20instructions

Benjamin, B., Coffman, J., Esiely-Barrera, H., Farr, K., Fichter, D., Genin, D., . . . Reller, N. (2017, 25-30 June 2017). Data Protection in OpenStack. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)* (p. 560-567). Honolulu, HI: IEEE.

Bucur, S., Kinder, J., & Candea, G. (2013). Making automated testing of cloud applications an integral component of PaaS. In *4th Asia-Pacific Workshop on Systems* (p. 1-

7). New York: United States Association for Computing Machinery.

Chan-In, P., & Wongthai, W. (2016). Logging solutions to mitigate risks associated with security issues in platform as a service cloud models. *Information (Japan), 19*(10), 4883-4890.

Chan-In, P., & Wongthai, W. (2017). Performance improvement considerations of cloud logging systems. *ICIC Express Letters, 11*(1), 37-43.

Cico, O., & Dika, Z. (2014). Performance and load testing of cloud vs. classic server platforms (case study: social network application). In *2014 3rd Mediterranean Conference on Embedded Computing (MECO)* (p. 301-306). Budva, Montenegro: IEEE.

Cotroneo, D., Paudice, A., & Pecchia, A. (2016). Automated root cause identification of security alerts: Evaluation in a SaaS Cloud. *Future Generation Computer Systems, 56*, 375-387.

CSA. (2010). Top threats to cloud computing, version 1.0. *Cloud Security Alliance, Tech. Rep., March*.

CSA. (2017). Security Guidance for Critical Areas of Focus in Cloud Computing v4.0 *Cloud Security Alliance (CSA)*.

CSA. (2019). Top Threats to Cloud Computing: Egregious Eleven *Cloud Security Alliance (CSA)*.

CSA. (2022). Top threats to cloud computing pandemic eleven, *Cloud Security Alliance (CSA)*.

Cunha, M., Mendonca, N., & Sampaio, A. (2013). A declarative environment for automatic performance evaluation in iaas clouds. In *2013 IEEE Sixth International Conference on Cloud Computing* (p. 285-292). Santa Clara, CA: IEEE.

Diaz, J., Munoz-Caro, C., & Nino, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on parallel and distributed systems, 23*(8), 1369-1386.

Felici, M., & Pearson, S. (2015). *Accountability for data governance in the cloud. Accountability and Security in the Cloud: First Summer School, Cloud*

Accountability Project. Malaga: A4Cloud.

Fernando, E., Murad, D. F., & Wijanarko, B. D. (2018). Classification and advantages parallel computing in process computation: A systematic literature review. In *2018 International Conference on Computing, Engineering, and Design (ICCED)* (p. 143-147). Bangkok, Thailand: IEEE.

Gartner. (2017). *Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017*. Retrieved February 22, 2017, from https://www.gartner.com/newsroom/id/3616417

Gartner. (2022). *Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023.* Retrieved Febuary 12, 2023, from https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023

Hammad, J. (2015). A comparative study between various sorting algorithms. *International Journal of Computer Science and Network Security (IJCSNS), 15*(3), 11.

Jaiboon, W., Wongthai, W., Phoka, T., & Auxsorn, T. (2020). A logging system in openstack environment to mitigate risks associated with threats in infrastructure as a service cloud. *ICIC Express Letters, 14*(4), 387-397. doi: 10.24507/icicel.14.04.387

Ko, R. K., Jagadpramana, P., & Lee, B. S. (2011). Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications* (p. 765-771). Changsha, China: IEEE.

Maata, R. L. R., Cordova, R., Sudramurthy, B., & Halibas, A. (2017). Design and implementation of client-server based application using socket programming in a distributed computing environment. In *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)* (p. 1-4). Coimbatore, India: IEEE.

Marszatek, Z., Woźniak, M., & Połap, D. (2018). Fully flexible parallel merge sort for multicore architectures. *Complexity, 2018*, 1-19.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing.* Gaithersburg, MD : Computer Security Division Information Technology Laboratory National Institute of Standards and Technology

Melvin, A. A. R., Kathrine, G. J. W., Ilango, S. S., Vimal, S., Rho, S., Xiong, N. N., & Nam, Y. (2022). Dynamic malware attack dataset leveraging virtual machine monitor audit data for the detection of intrusions in cloud. *Transactions on Emerging Telecommunications Technologies, 33*(4), e4287.

Mishra, P., Verma, I., & Gupta, S. (2020). KVMInspector: KVM Based introspection approach to detect malware in cloud environment. *Journal of Information Security and Applications, 51*, 102460.

Molyneaux, I. (2014). *The art of application performance testing: from strategy to tools.* Sebastopol, CA: O'Reilly Media,

OpenStack.org. (2015). *Ocata openstack summit recap.* Retrieved March 12, 2022, from https://docs.openstack.org/developer/performance-docs/summits_recaps/ocata_summit.html

OpenStack.org. (2017, 11-10-2017). *OpenStack installation tutoial for Ubuntu*. Retrieved August 20, 2018, from https://docs.openstack.org/ocata/install-guide-ubuntu/

OpenStack.org. (2019). *Welcome to openStack documentation.* Retrieved May 10, 2021, from https://docs.openstack.org/stein/

OpenStack.org. (2022). *OpenStack survey Report.* Retrieved Febuary 16, 2023, from https://www.openstack.org/analytics/

Payne, B., Maresca, S., Lengyel, T. K., & Saba, A. (2015). *LibVMI Virtual Machine Introspection Fast, Portable, Simple.* Retrieved May 5, 2023, from https://libvmi.com/contact/

Phalke, S., Vaidya, Y., & Metkar, S. (2022). Big-O Time Complexity Analysis Of Algorithm. In *2022 International Conference on Signal and Information Processing (IConSIP)* (p. 1-5). Pune, India:IEEE.

Ram, N., Ranjan, R., Chakrabarti, S., & Samanta, D. (2015). Application of Data Structure in the field of Cryptography. In *International Conference, Computational Systems*

for Health\& Sustainability, RV College of Engineering (p. 65-68).

Ristov, S., Gusev, M., & Donevski, A. (2013). Openstack cloud security vulnerabilities from inside and outside. *Cloud Computing*, 101-107.

Rocha, F., Abreu, S., & Correia, M. (2011). The final frontier: Confidentiality and privacy in the cloud. *Computer, 44*(9), 44-50.

Runathong, W., Wongthai, W., & Panithansuwan, S. (2017). A system for classroom environment monitoring using the internet of things and cloud computing. In *Information Science and Applications 2017: ICISA 2017 8* (p. 732-742). Singapore: Springer.

Saghir, A., & Masood, T. (2019). Performance evaluation of openstack networking technologies. In *2019 International Conference on Engineering and Emerging Technologies (ICEET)* (p. 1-6). Lahore, Pakistan: IEEE.

Sefraoui, O., Aissaoui, M., & Eleuldj, M. (2012). OpenStack: toward an open-source solution for cloud computing. *International Journal of Computer Applications, 55*(3).

Shah, S., & Shaikh, A. (2016). Hash based optimization for faster access to inverted index. In *2016 International Conference on Inventive Computation Technologies (ICICT)* (p. 1-5). Coimbatore, India: IEEE.

Sonuç, E., & Özcan, E. (2023). An adaptive parallel evolutionary algorithm for solving the uncapacitated facility location problem. *Expert Systems with Applications, 224*, 119956.

Spiliotis, I. M., Bekakos, M. P., & Boutalis, Y. S. (2020). Parallel implementation of the image block representation using OpenMP. *Journal of Parallel and Distributed Computing, 137*, 134-147.

Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications, 34*(1), 1-11.

Sujatha, K., Rao, P. N., Rao, A. A., Sastry, V., Praneeta, V., & Bharat, R. K. (2015). Multicore parallel processing concepts for effective sorting and searching. In *2015 International Conference on Signal Processing and Communication*

Engineering Systems (p. 162-166). Guntur, India: IEEE.

Thakur, N., Kumar, S., & Patle, V. (2014). Comparison of serial and parallel searching in multicore systems. In *2014 International Conference on Parallel, Distributed and Grid Computing* (p. 334-338). Solan, India: IEEE.

Tsai, W.-T., Huang, Y., & Shao, Q. (2011). Testing the scalability of SaaS applications. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)* (p. 1-4). Irvine, CA: IEEE.

Vailshery, L. S. (2022). Current usage of private cloud platform services worldwide from 2017 to 2022, by service. Retrieved December 1, 2022, from https://www.statista.com/statistics/511526/worldwide-survey-private-coud-services-running-application/

Wang, G., Estrada, Z. J., Pham, C. M., Kalbarczyk, Z. T., & Iyer, R. K. (2015). Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring (Master's thesis). Illinois: University of Illinois at Urbana-Champaign.

Wiriya, S., Wongthai, W., & Phoka, T. (2020). The enhancement of logging system accuracy for infrastructure as a service cloud. *Bulletin of Electrical Engineering and Informatics, 9*(4), 1558-1568.

Wongthai, W. (2014). *Systematic support for accountability in the cloud* (Dortoral dissertation). Newcastle: Newcastle University,

Wongthai, W., Rocha, F., & Van Moorsel, A. (2013a). Logging solutions to mitigate risks associated with threats in infrastructure as a service cloud. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on* (p. 163-170). Fuzhou, China: IEEE.

Wongthai, W., Rocha, F. L., & van Moorsel, A. (2013b). A generic logging template for infrastructure as a service cloud. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on* (p. 1153-1160). Barcelona, Spain: IEEE.

Wongthai, W., & Van Moorsel, A. (2016a). Performance measurement of logging systems in infrastructure as a service cloud. *ICIC Express Letters, 10*(2), 347-354.

Wongthai, W., & van Moorsel, A. (2016b) Quality analysis of logging system components in the cloud. *Vol. 376. Lecture Notes in Electrical Engineering* (pp. 651-662).

Wongthai, W., & Van Moorsel, A. (2017). Logging system architectures for infrastructure as a service cloud. *Journal of Telecommunication, Electronic and Computer Engineering, 9*(2-4), 35-40.

Wu, C.-S., & Lee, Y.-T. (2012). Automatic SaaS test cases generation based on SOA in the cloud service. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (p. 349-354). Taipei, Taiwa: IEEE.

Xue, M., & Zhu, C. (2009). The socket programming and software design for communication based on client/server. In *2009 Pacific-Asia Conference on Circuits, Communications and Systems* (p. 775-777). Chengdu, China IEEE.

Yao, J., Maleki Shoja, B., & Tabrizi, N. (2019). An overview of cloud computing testing research. In Cloud Computing–CLOUD 2019: 12th International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 12 (p. 303-313). San Diego, CA: Springer.

Zhou, J., Zhou, B., & Li, S. (2014). Automated model-based performance testing for PaaS cloud services. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops* (p. 644-649). San Diego, CA: IEEE.

ภาคผนวก

ภาคผนวก ก ผลงานตีพิมพ์เผยแพร่ งานที่ 1

ICIC Express Letters
Volume **14**, Number **4**, April **2020**

ICIC International ©2020 ISSN 1881-803X
pp. 387–397

A LOGGING SYSTEM IN OPENSTACK ENVIRONMENT TO MITIGATE RISKS ASSOCIATED WITH THREATS IN INFRASTRUCTURE AS A SERVICE CLOUD

Wichep Jaiboon[1], Winai Wongthai[1,2,*], Thanathorn Phoka[1] and Thongrob Auxsorn[3]

[1]Department of Computer Science and Information Technology
[2]Research Center for Academic Excellence in Nonlinear Analysis and Optimization
Faculty of Science
Naresuan University
99 Moo 9, Tambon Tha Pho, Muang, Phitsanulok 65000, Thailand
*Corresponding author: winaiw@nu.ac.th

[3]Department of Information Technology
Faculty of Science and Technology
Pibulsongkram Rajabhat University
156 Mu 5 Plaichumpol Sub-district, Muang District, Phitsanulok 65000, Thailand

Received October 2019; accepted January 2020

Abstract. *An Infrastructure as a Service or IaaS cloud is rentable virtual computing resources such as networking, servers, and storage that are offered by a cloud provider to its customers. The customers can use these resources through the Internet. However, the Cloud Security Alliance (CSA) indicates that there are security concerns of an IaaS such as confidentiality, integrity, and availability of customers' files in the IaaS. These concerns can affect both providers and customers. Currently, open-source software for creating IaaS clouds such as OpenStack has increasingly been applied in the production. However, an IaaS cloud that is built from software also yields the same security concerns as above. Thus, for the aim to mitigate the risks associated with the most serious CSA threat, this paper applied our previous logging system into OpenStack environment. This threat is a data breach which can be an incident when IaaS customer's sensitive or critical files are unauthorizedly opened, stolen, or used by unauthorized users. These files may contain sensitive data, such as health information. The results of this paper are that we delivered a design and implantation of a logging system architecture in OpenStack. Then, we illustrated the results of the delivered system and the discussions of the results. In real-world IaaS cloud production, the log files that were produced from the proposed logging system can be one of the solutions to assist the providers and customers in analyzing the risks above. Thus, the proposed logging system in the OpenStack environment for an IaaS cloud can help to mitigate risks associated with this CSA threat. This increases the reliability of an IaaS cloud to benefit both the providers and customers.*
Keywords: Cloud security, Logging system, OpenStack, Monitoring system, IaaS, Hypervisor

1. **Introduction.** National Institute of Standards and Technology [1] states that cloud computing or cloud is a model for using computing resources (such as networks, servers, storage, and applications). It also argues that these resources can be conveniently accessed from a variety of devices and connected via the network from anywhere. The provision and cancellation of these resources can be done quickly and adjusted according to needs, with minimum management effort or contact with the service provider [1]. The cloud is increasingly used in the organization because it can be scalability and elasticity as argued in [2]. The cloud can reduce the cost of information technology management in

DOI: 10.24507/icicel.14.04.387

an organization as discussed in [3, 4, 5, 6, 7, 8, 9, 10]. We focus on a public cloud. This cloud is provided for open use by the general public or cloud customers, can be operated, managed, and owned by business companies or cloud providers [1]. In this paper, the word 'a cloud' refers to 'a public cloud'. Infrastructure as a Service or IaaS cloud is provided by a provider for customers processing, storage, networks, applications, operating systems and other fundamental computing resources which the customers can adjust according to their needs [1]. This paper refers to 'an IaaS cloud' as only 'an IaaS'. An IaaS is increasingly used in many applications domains. Gartner predicts that in 2021, an IaaS market growth can be $83.5 billion [11].

However, there are still security concerns when using the IaaS cloud on issues of confidentiality, integrity, and availability of customer's sensitive files as argued by [8, 12]. The CSA published the treacherous 12: cloud computing top threats in 2016 report [13] that identifies the security concerns above. The concerns from the report can affect both the IaaS cloud providers and customers. The customers want to know where their files are stored or who has accessed the files. The provider should also provide the log data or evidence as proof for the customers when they need in case there is something wrong with the files [8]. To build an IaaS cloud, a provider can use a cloud software platform such as OpenStack to build this cloud with both copyright and open-source. This paper focuses on OpenStack, which is also the increasing trend of using open-source software in the business organization, as argued in [14]. From the security concerns mentioned above, we proposed in our previous work [8] a logging system to mitigate risks associated with CSA threats for IaaS cloud.

Research gaps: Firstly, rather experiment in OpenStack, logging systems from our previous work [8] were experimented in a simulated cloud environment in a single physical computer machine. These logging systems worked well; however, they need to work in a real-world production environment such as in OpenStack. Thus, in this paper, we aim to enable our logging systems from our previous work to perform in an OpenStack environment with the primary objective of mitigating the risks associated with a CSA threat for IaaS cloud. Secondly and thus, from the first research gap, there is no design and implementation of a logging system in OpenStack in the literature. We provide both the design and implantation. Thirdly and then, there are no results and discussions to illustrate how the results from design and implementation could help in mitigating the risks above. With an IaaS perspective, thus our research question is how can we enable our previous logging systems to work in an OpenStack environment.

Summary of contributions: This paper has the following four contributions. The first, in Section 2, we discuss how an IaaS cloud and our previous logging systems can be compatible with OpenStack environment. This is to ensure that our previous systems could be applied to the new OpenStack environment. Secondly, based on the first contribution, in Section 3, we apply our previous logging system into the new OpenStack environment. This application includes the design and implementation of a logging system in the OpenStack environment for an IaaS cloud. This ensures the technical details of the experiment to apply our previous works to the OpenStack environment. The third, Section 4 illustrates the results from the design and implementation from Section 3. The results confirm that our previous logging systems can be applied to the OpenStack environment or an IaaS OpenStack cloud. Finally, the fourth, Section 5 discusses the design and implementation from Section 3 and the results from Section 4. The examples of the discussions are why our logging system can still work in a real-world IaaS cloud which is built from OpenStack or an IaaS OpenStack cloud, and our logging system in an IaaS OpenStack cloud can be alternative solutions apart from OpenStack security features to mitigate risks associated with the CSA top severe threat. Interestingly, we found other interesting discussions such as our logger can work with many hypervisors. All the contributions may help to mitigate risks associated with the first CSA threat or data breathes

an organization as discussed in [3, 4, 5, 6, 7, 8, 9, 10]. We focus on a public cloud. This cloud is provided for open use by the general public or cloud customers, can be operated, managed, and owned by business companies or cloud providers [1]. In this paper, the word 'a cloud' refers to 'a public cloud'. Infrastructure as a Service or IaaS cloud is provided by a provider for customers processing, storage, networks, applications, operating systems and other fundamental computing resources which the customers can adjust according to their needs [1]. This paper refers to 'an IaaS cloud' as only 'an IaaS'. An IaaS is increasingly used in many applications domains. Gartner predicts that in 2021, an IaaS market growth can be $83.5 billion [11].

However, there are still security concerns when using the IaaS cloud on issues of confidentiality, integrity, and availability of customer's sensitive files as argued by [8, 12]. The CSA published the treacherous 12: cloud computing top threats in 2016 report [13] that identifies the security concerns above. The concerns from the report can affect both the IaaS cloud providers and customers. The customers want to know where their files are stored or who has accessed the files. The provider should also provide the log data or evidence as proof for the customers when they need in case there is something wrong with the files [8]. To build an IaaS cloud, a provider can use a cloud software platform such as OpenStack to build this cloud with both copyright and open-source. This paper focuses on OpenStack, which is also the increasing trend of using open-source software in the business organization, as argued in [14]. From the security concerns mentioned above, we proposed in our previous work [8] a logging system to mitigate risks associated with CSA threats for IaaS cloud.

Research gaps: Firstly, rather experiment in OpenStack, logging systems from our previous work [8] were experimented in a simulated cloud environment in a single physical computer machine. These logging systems worked well; however, they need to work in a real-world production environment such as in OpenStack. Thus, in this paper, we aim to enable our logging systems from our previous work to perform in an OpenStack environment with the primary objective of mitigating the risks associated with a CSA threat for IaaS cloud. Secondly and thus, from the first research gap, there is no design and implementation of a logging system in OpenStack in the literature. We provide both the design and implantation. Thirdly and then, there are no results and discussions to illustrate how the results from design and implementation could help in mitigating the risks above. With an IaaS perspective, thus our research question is how can we enable our previous logging systems to work in an OpenStack environment.

Summary of contributions: This paper has the following four contributions. The first, in Section 2, we discuss how an IaaS cloud and our previous logging systems can be compatible with OpenStack environment. This is to ensure that our previous systems could be applied to the new OpenStack environment. Secondly, based on the first contribution, in Section 3, we apply our previous logging system into the new OpenStack environment. This application includes the design and implementation of a logging system in the OpenStack environment for an IaaS cloud. This ensures the technical details of the experiment to apply our previous works to the OpenStack environment. The third, Section 4 illustrates the results from the design and implementation from Section 3. The results confirm that our previous logging systems can be applied to the OpenStack environment or an IaaS OpenStack cloud. Finally, the fourth, Section 5 discusses the design and implementation from Section 3 and the results from Section 4. The examples of the discussions are why our logging system can still work in a real-world IaaS cloud which is built from OpenStack or an IaaS OpenStack cloud, and our logging system in an IaaS OpenStack cloud can be alternative solutions apart from OpenStack security features to mitigate risks associated with the CSA top severe threat. Interestingly, we found other interesting discussions such as our logger can work with many hypervisors. All the contributions may help to mitigate risks associated with the first CSA threat or data breathes

in real-world IaaS cloud production, and maybe in other types of cloud computing such as PaaS and SaaS. Thus, to truly enhance security in a real-world IaaS cloud, this paper can be a guideline for mitigating first CSA threat and other eleven CSA threats.

2. Background.

2.1. Infrastructure as a Service architecture.

Figure 1 shows the IaaS architecture, which consists of two parties: the provider side and the customer side. The provider side is an organization (such as Amazon Elastic Compute Cloud) that offers various rentable services such as virtual machines or VMs to customers. The customer side is an organization or person that can access the services via the Internet. The main components of an IaaS architecture are hw, hypervisor, dom0, and domU. Hw is a physical computer machine to host all the rentable VM services, see the box at the bottom of the provider side of Figure 1. It is owned, managed, maintained by the provider. A hypervisor is a software that can enable the hw to run multiple VMs at the same time and in one hw, see the box in the middle on the provider side of Figure 1. See the box in the top left corner on the provider side of Figure 1, a dom0 stands for domain 0 and is a manager of all the VMs or domUs. It is an exclusive privilege domain of hypervisor. This means that it can access directly to the hw and can manage all the domUs. This domain will be started at the system boot. A domU or user domain, see domU1 and domUn in Figure 1, is an un-privilege domain for the customers to rent. A domU is running over the hypervisor and cannot access directly to the hw.

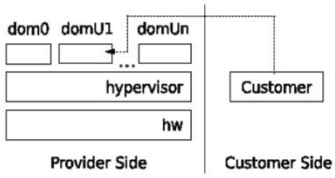


FIGURE 1. IaaS architecture from [8]

2.2. An OpenStack architecture.

From Figure 1, we will focus on the provider side by using OpenStack to create an IaaS public cloud to be close to the real-world production environment, in Section 2.3 and 3. [15] has compared the software used to create an IaaS public cloud such as Eucalyptus, OpenNebula, and OpenStack, using conditions including popularity, community, modularity, openness, and open-source. The results of this comparison show that OpenStack is the most appropriate option with scalable, compatible and flexible, and open-source features. OpenStack is for service providers, government agencies, enterprises, and academic institutions that want to build a cloud. This paper discusses OpenStack based on an IaaS provider's perspective. [16] states that OpenStack is a cloud operating system or OS that controls large pools of storage, networking, and compute resources in every part of a data center. The resources are managed via a dashboard that can enable the cloud provider's administrators to control while allowing their users to the resources via a web interface [16]. OpenStack is also open-source software that can be used to build an IaaS public cloud, and it can be used in the real-world cloud production environment. This cloud software has an increasing use in large business organizations [14]. Thus, we act as an IaaS provider then use OpenStack to create an IaaS public cloud to be close to a real-world cloud production environment. Then, we can and will apply our existing logging systems to this environment for our experiment in Section 3.1. See five boxes in Figure 2, there are five services of the architecture of OpenStack for an IaaS cloud [15], for short an IaaS OpenStack. These

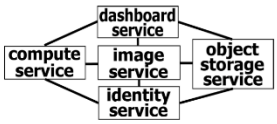


FIGURE 2. OpenStack architecture (from [15])

services are the compute service, image service, object storage service, dashboard service, and identity service.

Each service is connected to others through a communication network or a solid line in Figure 2. The details of each service are the following and mainly from the OpenStack official documentation website [17]. The first is the compute service that is the box on the leftmost of Figure 2. It can enable a provider to control an IaaS cloud. This service can also allow the provider in controlling over instances or VMs and networks and in handling access to the cloud via projects and users. It also defines drivers that interact with underlying virtualization mechanisms or a hypervisor that is run on the provider's host OS or dom0. This service also publishes its functionality for other services via a web-based Application Programming Interfaces or APIs. The compute service is usually designed to use the image service. The second is the image service that can store VM images and manage a list of accessible images. A VM image is a file that can be instanced then became a VM or domU.

The third is the object storage service that is used for scalable, redundant data storage using clusters of standardized servers to store petabytes or PB of accessible data. This service is a long-term storage system for large amounts of static data which can be updated and retrieved. The fourth is at the box on the top of Figure 2 or the dashboard service. It is a web-based interface that allows a provider to manage OpenStack resources and services, and to interact with the compute service cloud controller using the OpenStack APIs. The last service is the identity service that is the default identity management system for OpenStack. After the IaaS provider installs this service, the provider can configure the service and can also initialize data into this service.

2.3. **The logging system in OpenStack.** Our previous work [8] illustrated the IaaS architecture and logging system architectures. Section 2.2 also illustrated the OpenStack architecture. However, no research illustrates how IaaS and logging system architectures can be applied to or combined with the OpenStack architecture. This section briefly introduces this combination, which is the innovative part of this work. The examples of the usefulness of this combination are: i) to be used as a fundamental to construct a real-world IaaS OpenStack cloud with the logging systems and with more than one compute node as will be discussed in Section 3.2.3, and ii) to be alternative solutions apart from OpenStack security features to mitigate risks associated with the CSA top severe threat as will be discussed in Section 4.2.4. The full details of the combination are in Section 3. [8, 12] discuss the security concerns in a public cloud on issues of confidentiality, integrity, and availability that relate to the security issues in the CSA report [13]. This report indicates an effect on the provider and customer on a public cloud. From the security concerns, [8] proposes a method to mitigate risks associated with CSA threats for an IaaS public cloud by using logging systems. [8] describes the logging system for IaaS as a system in the cloud infrastructure of the provider that collects and stores logging data of infrastructure's components such as VMs or domUs. A logging system consists of a logging process and log file. The logging process is responsible for recording data, whereas the log file used for storing data obtained from the logging process. In Figure 3, consider only the part of the compute node or a dotted square box on the right of this figure, the compute node is a computer machine with a hypervisor. A logging system is inside

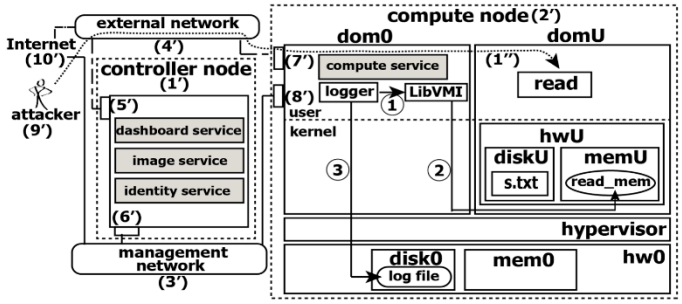


FIGURE 3. The architecture of the logging system in the OpenStack for IaaS

the compute node in the OpenStack environment. Section 2.1 described each component of IaaS architecture. This section will describe the additional components related to a logging system. In the compute node in Figure 3, any word ending with 0 indicates that this word or component is physically owned and managed by a provider, and ending with U indicates that the component is virtually owned and managed by a customer. Hw0 is a physical computer to be used as an IaaS VM host system, the same as hw discussed in Section 2.1. HwU is virtual hardware of a domU. Disk0 is the physical disk of dom0, and diskU is the virtual disk of domU. Mem0 is the main memory of hw0 or dom0 and menU is virtual main memory of hwU or domU. In Figure 3 see the logger box in the user level in dom0, it is a logger or logging process of a logging system that is run in dom0 to capture logging data. In Figure 3 see the box in the user level of domU, it is read or read process. In the experiment, we assume that this process can maliciously read IaaS customer's sensitive file or s.txt, see the box inside diskU. This file is normally stored in diskU that is owned by the customer or the owner of domU. This file can be an asset and valuable for a business organization of a customer as argued in [8]. Finally, the compute service is an application in the user level of dom0 for controlling the hypervisor, which is not related to the logging system.

3. The Experiment.

3.1. The aim of the proposed logging system in the OpenStack environment for IaaS.

[8] proposed logging system architecture to mitigate risks associated with CSA threats for IaaS. The main components of the architecture are the logging system or logger in Figure 3 and log file in the same figure. This paper defines the appropriate locations for placing the logging system components in the OpenStack environment to detect the target process or the read process on the IaaS customer's domU. We focus on the compute service, see the box on the leftmost of Figure 2. [18] states that the compute service is the most vulnerable part and the main component of the OpenStack. It is also a channel that leads to other critical IaaS resources such as VM or domU in Figure 3. For processes that are detected, it is a process which we suspect that the attacker will use it to access customer's sensitive files in diskU.

3.2. The system architecture of the proposed logging system in the OpenStack environment for IaaS in the experiment.

3.2.1. *The IaaS architecture on public cloud in the OpenStack environment.* In Figure 3, we illustrate the perspective of the IaaS architecture in the OpenStack environment, for short the IaaS OpenStack cloud. We run Ubuntu 16.04 LTS OS on a computer or hw0 as the compute node. Then we install OpenStack Ocata version in this OS. We map each

service of the OpenStack architecture in Figure 2 onto two physical computers. According to the function in the installation manual [19], one of these computers is the compute node, and the other one is the controller node. See the dotted box with number 1' on the left side of Figure 3; it is the controller node for execution of the three services of Figure 2, including identity service, image service, and dashboard service. These services were described in Section 2.2. See the dotted box with number 2' on the right side of Figure 3; it is the compute node execution of the compute service from Figure 2. This service is used to control hypervisor or Xen to manage dom0 and domUs. The object storage service in Figure 2 is not related to this experiment. There can be many of these nodes in the real-world environment. However, to simplify the experiment, from the architecture mentioned above or Figure 3, we use only one controller node, and one compute node. Both of the nodes use two network adapters (5' to 8') to connect the external network (4') and the management network (3'), respectively. See the dotted line with number 1", the external network is used for the customers to connect to domUs via the Internet or 10', in this IaaS OpenStack environment. In the experiment, this external network may also be somehow a channel that attacker or 9' can use this channel to connect to domUs with the credentials and passwords that the attacker may steal through phishing and fraud as argued in [8]. The management network is used for installing applications and for updating the patch of the security of the operating system of the controller node and the compute node. After building IaaS by using OpenStack or we called it the IaaS OpenStack cloud for the experiment, we got our IaaS OpenStack cloud. Then, we simulate incidents that were assumed to be taken place by the attacker. Thus, the attacker can connect to a domU in this built IaaS OpenStack cloud over the Internet using the credentials and password mentioned above. Then, the attacker may use read application or process (see the box in the user level of domU in Figure 3) to access to the sensitive file in diskU. We aim to locate the existing logging system or the logger (see the logger box in the user level of dom0) and the log files into this built IaaS OpenStack cloud. So, this logger has been designed to detect read application (read process). We consider the read process as a target process that the logger wants to capture the appearance of this process for the detection of such incidents.

3.2.2. *The architecture of the proposed logging system in the OpenStack environment.* Figure 3 is mapping our existing logging system architecture onto the built IaaS OpenStack cloud environment. Thus, the internal of the compute node in Figure 3 is the proposed logging system architecture and the main components of this logging system that were described in Section 2.3. In this research, we focus on the compute node because it is the most vulnerable part and the main components of the OpenStack (which has given the reason in Section 3.1). In this section, we illustrate the locations of the logger and log file of the proposed logging system in the built IaaS OpenStack cloud. There are three main components: the logger, LibVMI, and the log file. The LibVMI is a library to read data from memU in domU. The steps of the logger were discussed in our previous work [8] as the following. The first step or number 1 in Figure 3, see the logger box in the user level of dom0, the logger calls the LibVMI library to read data from read_men in memU. The second step, see number 2 in Figure 3, the library retrieves appropriate logging data from read_mem. The data is generated when the read process is executed. The examples of the data are the name of a process such as 'read' and Id of the process such as '1265'. See number 3 in Figure 3 the third step, the logger stores the captured data from step 2 into the log file in disk0.

3.2.3. *To construct a real-world IaaS OpenStack cloud with more than one compute node.* The proposed architecture in Figure 3 can be expanded to construct a real-world IaaS OpenStack cloud with more than one compute node. We also believe that our logging system (the logger, LibVMI, and log file in Figure 3) should also be ready to work in a

production environment (more than one compute node). We will discuss this in Section 4.2.2. [15] states that OpenStack can support a large number of compute nodes and storage units. Thus, in an OpenStack environment, a real-world IaaS OpenStack cloud is when this cloud is at least constructed with many compute nodes, not with only one compute node that will be discussed in Section 4.2.1. When constructing a real-world IaaS OpenStack cloud with our logging system, this can be done by step 1 or duplicating the whole compute node in Figure 3 with all its components including the logger, LibVMI, and log file. In step 2, we can connect the new duplicated compute node obtained from step 1 to the controller node or the dotted box at the left of Figure 3. Then, this cloud was extended from one compute node to two compute nodes and become a real-world IaaS OpenStack cloud. Both steps can be continuously repeated to construct a real-world IaaS OpenStack cloud with our logging system. Both steps are based on the builders or providers requirements on how many compute nodes in this IaaS cloud they need. This is our future work and out of the scope of this paper.

4. **The Results and Discussions.**

4.1. **The results.** In Figure 4, the line with number 1 is when the logger wants to capture the name and the Id of the monitored process or read inside domU. Thus, the logger command or 'logger' in the box with 'a' is executed in dom0 by the provider. Also, there are two parameters of this command including the name of domU or 'instance-00000007' (see in the box with 'b') and the name of the monitored process or 'read' (see in the box labeled 'c') on domU. The logger command will check read_mem until it found the read process has appeared in read_mem. When the read process is executed in domU or see the box in line with number 1 in Figure 5, this is when the name and Id data of read process have appeared in read_mem. Then, the logger captures the Id of read as '1265' and the name of read as 'read', see the boxes with 'a' and 'b' in line 2 of Figure 4, respectively. Then, the logger command will be terminated.

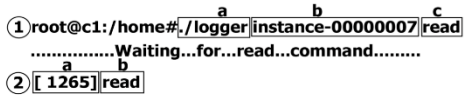


FIGURE 4. The logger in dom0

FIGURE 5. The read command in domU

4.2. **Discussions.**

4.2.1. *Our previous logging system working in an IaaS OpenStack cloud with one compute node.* Although the logging systems in our previous work [8] can work well in the laboratory environment or Figure 1, we never apply the systems to a real-world IaaS cloud like in an IaaS OpenStack cloud or Figure 3. After the experiment to apply our existing logging system to an IaaS OpenStack as illustrated by Figure 3, the results of the experiment in Figures 4 and 5 in Section 4.1 is the proof that our existing logging system from our previous work [8] can be applied to and work in the new built IaaS OpenStack cloud or Figure 3. Thus, this logging system in Figure 3 has the essential abilities to detect and record a monitored or malicious process caused by the CSA threat. Then it can store the captured data or logging data into the log file. The code of logger command of the logging system in Section 4.1 can also be modified to detect the new process and any file

that the new process maliciously reads or performs other operations with the file such as maliciously deleting the file as agreed in [8, 10]. This modification is out of the scope of this paper. However, we believe that our proposed logging system architecture or Figure 3 could be applied to and works in any IaaS cloud that is built by OpenStack. Then, this logging system can be one of the solutions to help in mitigating the risks associated with the CSA threats in a real-world IaaS OpenStack cloud production. Moreover, Section 3.2.3 already discussed to construct a real-world IaaS OpenStack cloud with more than one compute node.

4.2.2. *Why the existing logging system can still work in a real-world IaaS OpenStack cloud with more than one compute node.* There is only one compute node in the experiment or Figure 3. However, (when one desires to increase the number of compute nodes from one (in the experiment) to two or more to build a real-world IaaS OpenStack cloud as just discussed in Section 3.2.3 above, and from the experiment or Figure 3 and the results from Figures 4 and 5), we believe that our logging system can still work in this new built real-world IaaS OpenStack cloud with more than one compute node. The system can also still detect and record a monitored or malicious process, as discussed in Section 4.2.1. This is because of that when constructing a real-world IaaS OpenStack cloud with more than one compute node as discussed in Section 3.2.3, increasing the number of compute nodes by adding a new compute node into this newly built real-world IaaS cloud does not affect the operations of the existing compute node or the dotted box at the right of Figure 3. Also, it is because of that each compute node in this newly built IaaS environment is independent of one another. Moreover, a logging system in the new added or duplicated compute node will have the same functions as the logging system in the previous compute node in Figure 3. Thus, we believe that our logging system in Figure 3 which has the log file that is produced from the logger and LibVMI can still mitigate risks associated with the CSA threats in a real-world IaaS OpenStack cloud in a production environment, not only in the simulation or laboratory environment, as also argued in [8].

4.2.3. *Our logging system compatible with a variety of hypervisors.* Our logging system in Figure 3 can also be compatible with a variety of hypervisors. [15] states that OpenStack can support a variety of hypervisors such as VMware Elastic Sky X or ESX, Microsoft Hyper-V, Linux Containers or LXC, Kernel-based Virtual Machine or KVM, Quick Emulator or QEMU, Xen, and XenServer. Thus, this makes OpenStack to be used to build an IaaS cloud with a variety of hypervisors. In the experiment, the hypervisor of our compute node in Figure 3 is only Xen. However, [20] states that a hypervisor of a compute node can be changed from Xen to be KVM or QEMU, as also discussed above. Moreover, our logging system or the logger in Figure 3 works with LibVMI that is compatible with and can work with KVM or QEMU, not only with Xen, as agreed in [21]. Thus, our logger can still work to detect and record a monitored or malicious process (discussed in Section 4.2.1) in both new KVM or QEMU environment, not only in Xen. This can expand our logging system to work in at least two more hypervisors or KVM and QEMU of IaaS clouds which are built from open-source software such as OpenStack. This software is increasingly used in the IaaS cloud ecosystem, as agreed by Red Hat company in its report [14]. The company also offers its OpenStack version called 'Red Hat OpenStack Platform' to its clients such as Cathay Pacific. Thus, our logging system should also work with ESX, Hyper-V, LXC, and XenServer.

4.2.4. *The proposed architecture as alternative solutions apart from OpenStack security features to mitigate risks associated with the CSA top severe threat.* As discussed in Section 4.2.1, the code of our logger can be modified to detect when a malicious process reads a sensitive file that is belonged to a domU. The CSA [13] states that the first or most severe threat is data breaches. A data breach is when incidents have occurred with sensitive

data in an IaaS cloud and even the other types of the cloud such as Platform as a Service or PaaS [13]. This sensitive data in the IaaS cloud (such as health and financial information) should not be exposed, stolen, or used by an unauthorized user [13]. OpenStack is an open-source software to create an IaaS cloud. The data in this cloud may also be affected by this CSA thereat discussed above. When building an IaaS cloud by OpenStack, we will call this cloud as 'an IaaS OpenStack cloud' in this discussion. [22] agrees that an IaaS OpenStack cloud already had three features to maintain data security of the data of this cloud. However, these features may still yield disadvantages, as will be discussed one by one here. The first feature is the key management which helps to manage five mechanisms to maintain the confidentiality of the data in a newly built IaaS OpenStack cloud. This feature may decrease the performance of a key management server of this cloud. The reasons are the following. We will discuss only the first two mechanisms in this paper for the purposes of this discussion of decreasing the performance. The first mechanism is the key generation that can produce keys to be used to perform cryptographic of the data in an IaaS OpenStack cloud. The second mechanism is storing and retrieving the produced keys that are used to perform encryption and decryption, and certificate generation of this data in this cloud. [22] agrees that both mechanisms mentioned above face mainly two issues. The first issue is when too many simultaneous requests from computers of this cloud's customers via the Internet to a key management server. These requests also need many key generating and key retrieving mechanisms simultaneously. This may cause bottlenecks of this key management server. The second issue is that the key generating mechanism is an intensive computation process on the key management server.

Thus, the first feature to maintain data security of the data in an IaaS OpenStack cloud which relies on the key management may decrease the performance of the key management server of this cloud. The second feature is the block storage encryption, that is the encryption and decryption of a block storage unit. A block storage unit is a method of storing the data in a storage-area network (SAN) environment. An IaaS OpenStack cloud needs to deploy SAN, and the data of this cloud is stored in a volume or block. The block storage encryption operates in the compute node or the dotted box at the right side of Figure 3. Operating encryption on the compute node may yield disadvantages. This is because the block storage unit cannot apply the advantage of the compression function. It is because this function is one of the keys functions to reduce 90% of data in the provider storage [22]. Without this function, the provider may need to invest more budget for its storage units. The final feature is called the image integrity that can check the integrity of an image file of a VM or domU whether this file is modified by malicious code or compromised by an attacker or not. (A VM image is a file that can be instanced then became a VM or domU.) This file is in the controller node or the dotted box at the left of Figure 3. However, Benjamin et al. [22] measured overheads of the image integrity feature by the length of time required to launch a domU with and without signature verification. Then, they found the launch time of an image file with a signature verification higher than another launch time without the signature verification. It is because this higher launch time needs a massive computing process for the secure hash algorithm of the image file, as a part of the signature verification [22]. Thus, this feature may reduce the speed of overall operations in an IaaS OpenStack cloud. Then, this may also slow down the activities of domUs in this cloud infrastructure. An IaaS OpenStack cloud already had three features to maintain data security of the data of this cloud. However, these features may still yield disadvantages. Our logging system in an IaaS OpenStack cloud can be alternative solutions apart from OpenStack security features to mitigate risks associated with the CSA top severe threat.

4.2.5. *The proposed architecture without encryption and decryption mechanisms and with a lightweight computation.* Sections 3.2.3, 4.1, 4.2.1, 4.2.2, and 4.2.3 discussed how our

existing logging system from our previous work [8] can perform in a real-world IaaS OpenStack cloud, see in Figure 3. This system has abilities to detect and record a monitored process in or customer's critical files in a domU. Then, it can store the captured data or logging data into the log file. The contents of the log file can be analyzed or checked for examining any incident that may occur, as also done in [8, 10]. Thus, we believe this logging system can be one of the solutions to mitigate risks associated with the CSA threat one. This logging system in this paper can usually work without cryptographic techniques not with these techniques as done by three features to maintain data security of the data in the OpenStack environment as just discussed in Section 4.2.4 above. Then, this should allow our logging system to have a lightweight computation.

5. **Conclusions.** An Infrastructure as a Service cloud can offer rentable virtual machines or VMs to customers. However, CSA indicates that there are threats of an IaaS such as confidentiality, integrity, and availability of customers' files in the VMs. Thus, for the aim to mitigate the risks associated with the most severe CSA threat or threat one (the example of threat one is when an IaaS customer's file is unauthorizedly viewed), this paper applied our previous logging system into a new IaaS environment. A logging system can be one of the solutions to mitigate these risks. The system can capture activities of a malicious process that is reading a customer file in a VM. However, logging systems in our previous work, such as [8] cannot work in a real-world IaaS cloud that is built from widely used OpenStack software. We call this cloud an IaaS OpenStack cloud. Our logging systems can only work in a simulation or laboratory environment. Then, this paper successfully applied a previous logging system to an IaaS OpenStack cloud. We also found many useful aspects of the application of this logging system in this paper, such as the system can work with the other virtualization softwares such as Kernel-based Virtual Machine or KVM rather than only Xen. Thus, this paper can be an alternative solution apart from three OpenStack built-in features that are used to maintain data security of an IaaS cloud which is built from OpenStack. Then, this paper could truly enable our logging systems from our ten years of work [5, 6, 7, 8, 23] to be integrated and compatible with the widely used OpenStack software. This can help to mitigate risks associated with CSA threat one in real-world IaaS cloud production, and maybe in other types of the cloud such as Platform as a Service or PaaS. This paper also can be a guideline for mitigating other critical CSA threats such as the second most severe threat or 'Insufficient Identity, Credential and Access Management'. Our future work can be to measure the performance of the logging system of this paper.

REFERENCES

[1] T. Grance and P. Mell, *The NIST Definition of Cloud Computing*, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[2] *Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017*, 2017.

[3] H. Albaroodi, S. Manickam and P. Singh, Critical review of OpenStack security: Issues and weaknesses, *Journal of Computer Science*, vol.10, p.23, 2013.

[4] W. Runathong, W. Wongthai and S. Panithansuwan, A system for classroom environment monitoring using the internet of things and cloud computing, *Lecture Notes in Electrical Engineering*, vol.424, pp.732-742, 2017.

[5] W. Wongthai and A. van Moorsel, Logging system architectures for infrastructure as a service cloud, *Journal of Telecommunication, Electronic and Computer Engineering*, vol.9, nos.2-4, pp.35-40, 2017.

[6] W. Wongthai and A. van Moorsel, Quality analysis of logging system components in the cloud, *Lecture Notes in Electrical Engineering*, vol.376, pp.651-662, 2016.

[7] W. Wongthai and A. van Moorsel, Performance measurement of logging systems in infrastructure as a service cloud, *ICIC Express Letters*, vol.10, no.2, pp.347-354, 2016.

[8] W. Wongthai, *Systematic Support for Accountability in the Cloud*, Ph.D. Thesis, Newcastle University, 2014.

[9] P. Chan-in and W. Wongthai, Performance improvement considerations of cloud logging systems, *ICIC Express Letters*, vol.11, no.1, pp.37-43, 2017.
[10] P. Chan-in and W. Wongthai, Logging solutions to mitigate risks associated with security issues in platform as a service cloud models, *Information (Japan)*, vol.19, no.10, pp.4883-4890, 2016.
[11] C. Coles, *Cloud Market in 2018 and Predictions for 2021*, 2018.
[12] S. Ristov, M. Gusev and A. Donevski, OpenStack cloud security vulnerabilities from inside and outside, *Cloud Computing*, pp.101-107, 2013.
[13] C. S. Alliance, *The Treacherous 12: Cloud Computing Top Threats in 2016*, 2016.
[14] redhat, *Digital Transformation – The Open Source Way*, 2017.
[15] O. Sefraoui, M. Aissaoui and M. Eleuldj, OpenStack: Toward an open-source solution for cloud computing, *International Journal of Computer Applications*, vol.55, no.3, pp.38-42, 2012.
[16] OpenStack, *What is OpenStack?*, 2019.
[17] OpenStack.org, *Welcome to OpenStack Documentation*, 2019.
[18] I. A. Elia, N. Antunes, N. Laranjeiro and M. Vieira, An analysis of OpenStack vulnerabilities, *European Dependable Computing Conference*, pp.129-134, 2017.
[19] OpenStack.org, *OpenStack Installation Tutoial for Ubuntu*, 2017.
[20] G. Wang, Z. J. Estrada, C. Pham, Z. Kalbarczyk and R. K. Iyer, *Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring*, USENIX Workshop on Offensive Technologies, 2015.
[21] B. Payne, *About the VMI Tools Project*, 2013.
[22] B. Benjamin, J. Coffman, H. Esiely-Barrera, K. Farr, D. Fichter, D. Genin, L. Glendenning, P. Hamilton, S. Harshavardhana, R. Hom, B. Poulos and N. Reller, Data protection in OpenStack, *International Conference on Cloud Computing*, 2017.
[23] W. Wongthai and A. van Moorsel, An approach to defining and identifying logging system patterns for infrastructure as a service cloud, *ICIC Express Letters*, vol.12, no.10, pp.1009-1016, 2018.

ภาคผนวก ข ผลงานตีพิมพ์เผยแพร่ งานที่ **2**

ICIC Express Letters

An International Journal of Research and Surveys

March 16, 2023

ACCEPTANCE LETTER (Paper ID: ICICEL-2212-021)

Dr. Winai Wongthai
Naresuan University
Thailand
Email: winaiw@nu.ac.th

Dear Dr. Winai Wongthai,

We are happy to inform you that after peer review, your following paper,

Reference No.: ICICEL-2212-021
Title: Enabling a Centralized Logging System in an Infrastructure as a Service Cloud
Author(s): Wichep Jaiboon, Pakorn Chan-in, Winai Wongthai and Thanathorn Phoka

has been formally accepted for publication in ICIC Express Letters – An International Journal of Research and Surveys (ISSN 1881-803X).

We are planning tentatively to publish your paper in Volume 17, Number 11, November 2023.

Thank you for submitting your work to ICIC-EL.

ICIC-EL Editorial Office
Tokai University, Kumamoto Campus
9-1-1, Toroku, Kumamoto 862-8652, Japan
Tel: +81-96-386-2666
E-mail: office@icicel.org
URL: http://www.icicel.org

ICIC Express Letters
Volume **4**, Number **5**, October **2022**

ICIC International ©2010 ISSN 1881-803X
pp. 1–ISII2010-000

ENABLING A CENTRALIZED LOGGING SYSTEM IN AN INFRASTRUCTURE AS A SERVICE CLOUD

Wichep Jaiboon[2] Pakorn Chan-in[3] Winai Wongthai[1,2,*] and Thanathorn Phoka[1,2]

[1]Research center for Academic Excellence in Nonlinear Analysis and Optimization,
[2]Department of Computer Science and Information Technology,
Faculty of Science, Naresuan University, Phitsanulok 65000, Thailand, and
[3]Faculty of Sciences and Agricultural Technology, Rajamangala University of Technology Lanna Nan

Abstract. *This paper proposes a new architecture to enable a centralized logging system in an Infrastructure as a Service cloud. The main contribution of this work can improve the implementation and management of the current decentralized logging systems. This is where only a socket programming method is required as a primary method of the improvement. Achieving the new architecture is by redesigning the architecture of our own existing logging system, which was already in an OpenStack environment. Based on the patterns and behaviours of this environment, the redesigning process mainly includes relocating the main components of the logging system and generating a new architecture to enable a centralized logging system in this cloud type. To examine the successes of our new architecture, the OpenStack environment is also the main technology to be used to implement a logging system based on the new architecture and for the experiments. Then, this logging system is used to test the main logging tasks, including detecting malicious processes inside a customer cloud. The results of the tests show that we can successfully detect malicious processes inside the customer cloud using the logging system of the new-centralized architecture. The logging systems with the previous-decentralized architecture can also detect this kind of process, but with more than one manager, compared to only one manager of the new architecture. Thus, with only one manager, the centralized architecture could be easier to be implemented and be managed than the decentralized one. As a central contribution to the cloud security field, this centralized architecture can help in mitigating the cloud security issues, such as malicious process activities inside the cloud. The issues are the critical one that prevents the adoption of the cloud.*

Keywords: Cloud Security, Infrastructure as a Service, Centralized Logging System, OpenStack, Malicious Process

1. **Introduction.** Cloud computing or the cloud offers visualized computing, storage, and networking resources over the Internet to organizations or customers. The cloud service rental model is flexible. The model allows easily reducing, adding, or discontinuing the services. This makes the cloud to be interesting in organizations bringing it to their IT department. This paper focuses on the Infrastructure as a Service or IaaS cloud type. IaaS mainly offers rentable virtual machines (VMs) to consumers. However, cloud security is a critical issue that prevents the adoption of the cloud. The issue has been considered by many researchers. This is especially about 14 years-consideration in about 23 reports with five languages of the Cloud Security Alliance (CSA). An example is the provision of the 'Top Threats to Cloud Computing, version 1.0' report [1] in 2010 to 'Top Threats to Cloud Computing Pandemic Eleven' in 2022 [2]. Researchers have also been investigating how to prevent and mitigate the risks associated with the issue. The results from the investigation can enhance the confidence of consumers or organizations that want to adopt the cloud. In an IaaS cloud, one of the mitigating solutions is a logging system. This system can help in mitigating risks associated with the issue, as discussed in [3–6]. This system facilitates investigating suspicious events, for example, who is accessing a cloud customer's sensitive file in an IaaS environment. Thus, the system can detect and report what unauthorized users do with the file.

Motivation: thus, this file has its own history produced by this logging system. This system can be categorized as a file-centric logging system rather than a system-centric one that only logs a machine's health data, such as the total percentage of CPU usage, not a sensitive file's history. A file-centric log can be an event of, for example, which file is accessed by which process. Oppositely, this can be

seen as a process-centric log perspective as to which process is accessing which file. Both file-centric and process-centric logs can be produced by a virtual machine introspection (VMI). It can introspect into a main memory of a VM to capture a process's information and behaviours. In recent years, VMI has been useful for intrusion detection systems (IDS) in cloud computing as agreed and implemented by [7] and [8]. However, the systems from both references are decentralized. [9] states that it is tough to manage the decentralized log-files of a logs management in cloud computing. We also believe that the logger components of these decentralized systems are difficult to be managed. Thus, in an IaaS cloud, our motivation is to enable logger components of a logging system to be centralized. This system can be considered a new logging system architecture in IaaS. Our paper aims to expand the capabilities of existing logging systems. To have efficient recording capabilities of the events in the IaaS cloud, the study of the functions of the logging system can be vital.

However, the previous works [4–6] provide the logging systems in a laboratory IaaS environment rather than in a production IaaS cloud environment. Thus, this paper focused on enabling the existing logging system in [4–6] to be able to work in the production IaaS cloud environment. We simulated this environment with a cloud operating system (OS) called OpenStack [10]. This cloud OS has increasingly been applied in a cloud production business [11, 12]. Then, we used the socket programming method [13] to enable the existing logging system to work in this simulated environment. The socket programming method is a technique for exchanging messages between processes on the same computer or across a network. This method can reside on the same system or different systems on a different network. Finally, in our previous work [3], we managed to enable the existing logging system to work in the simulated production IaaS cloud environment. This enabled logging system can help in mitigating risks associated with the cloud issue in this environment. In [3], one of the main components of our architecture is a compute node. It is a physical computer/machine with hypervisor software. The software can create more than one Virtual Machine (VM) in this physical machine. A logging system is a kind of monitoring system that can be distributed into many compute nodes. This distribution can cause difficulty in the management of the nodes.

Research gaps and objectives: there are three research gaps and three objectives. Firstly, in the OpenStack architecture, there can be more than one compute node in the architecture, as briefly discussed above and thoroughly discussed in [3]. Thus, there can be more than one logging system in these compute nodes of this architecture. This is a distributed/decentralized architecture (such as the ones of [7] and [3]) for the logging system. This decentralized architecture is difficult to be managed based on our primitive experiments of enabling logging systems into the OpenStack architecture. Thus, the first objective is that we aim to enable the existing logging system from [3] to work in an OpenStack architecture by using the socket programming method. This enables a centralized architecture for the logging system, which could be easier to be managed. From the first research gap, the second gap is that there are no design and implementation of the logging system using the socket programming method in the OpenStack architecture in the literature. Thus, the second objective is that we will provide both the design and implementation. The third gap is that there are no results and discussions to illustrate how the results from the design and implementation work in the OpenStack architecture. Then, the final objective is to provide these results and discussions.

Summary of contributions: this paper has the following four contributions. The first is that we discuss an IaaS cloud and how the existing logging system in the OpenStack architecture works. Then, we illustrate that the existing logging system from [3] is a decentralized architecture and challenging to manage in the OpenStack architecture. The second contribution is that, based on the first contribution, we redesigned the existing logging system to support centralized architecture via the socket programming method. This includes designing and implementing the logging system in the OpenStack architecture of an IaaS cloud. This design and implementation ensure the technical details of redesigning the existing logging system to apply in the OpenStack architecture. The third one is that we illustrate the results from the design and implementation of the second contribution. The results confirm that the redesign of the existing logging system can still detect malicious processes like our previous logging system but with easier management compared to the previous system. The last contribution is the discussions of the design and implementation and the results. The examples of the discussions are why the redesign of the existing logging system can still work in the real-world IaaS cloud, which is built from OpenStack or an IaaS-OpenStack cloud, and this logging system in the IaaS-OpenStack cloud can be easier to manage than the previous existing logging system.

The organization of this paper is the following. The background is described in Section 2, including IaaS architecture, the existing logging system architecture in the laboratory IaaS cloud environment, an OpenStack architecture, and the architecture of the logging system in the OpenStack for IaaS. The system design and implementation of the proposed architecture are discussed in Section 3, including the hardware and software of the experimental environment, the aim of the new logging system architecture

based on the socket programming method for IaaS, and the proposed new logging system architecture in IaaS-OpenStack cloud. The results and discussions of the proposed architecture are in Sections 4 and 5, respectively. Section 6 is the conclusion and future work of this paper.

2. Background.

2.1. Infrastructure as a Service Architecture.

Figure 1 shows an IaaS cloud and logging system architectures. In this paper, both architectures are mainly adapted from our previous work [4]. All the white boxes, an ellipse, and text file shape in Figure 1 are the components of the IaaS cloud architecture. This section will describe these components. All the shaded boxes in Figure 1 are the logging system's main components. Section 2.2 will describe these main components. This section here will shortly explain the IaaS cloud architecture as followings. The white box components include hypervisor, dom0, hw0, disk0, domU, hwU, diskU, and memU. A component name ending with '0' implies that this component is physically owned and managed by an IaaS cloud provider. However, ending with 'U' implies that the component is virtually owned and managed by a cloud customer. See the box with number 2 in Figure 1, and it is a hypervisor or software that grants a physical computer to host more than one VM.

The box inside the domU, the ellipse in memU, and the text file shape in Figure 1 are the components for the simulation of some incidents in our experimental purposes in Section 3. A dom0 or domain 0 is a manager of the entire VMs that were created by the customers; see the topmost left white box in Figure 1. This domain is also a VM and is launched by the hypervisor at the system booting time. It completely accesses and manipulates hw0 and all the created VMs or domUs. Hw0 is the physical hardware of and managed by a dom0; see the bottom box in Figure 1. A domU or user domain is a user VM that is created by dom0; see the topmost right white box in Figure 1. It runs on top of the hypervisor and is an IaaS cloud product that the provider can rent to the customers. HwU is physically located in hw0 and is the virtual hardware of a domU. It is physically owned and managed by a dom0 or by the provider. However, it is virtually owned and managed by a domU owner or a customer. A disk0 is a physical disk of a dom0, and a diskU is a virtual disk of a domU. Finally, memU is domU's virtual main memory.

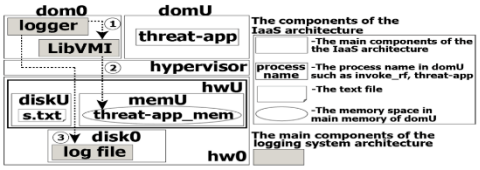


FIGURE 1. The IaaS cloud architecture and logging system architecture, adapted from [4]

FIGURE 2. An OpenStack Architecture from [3]

2.2. The existing logging system architecture in laboratory IaaS cloud environment.

The IaaS cloud architecture and logging system architecture in Figure 1 are in a non-real-world production environment. This environment was built on one computer machine to simulate an IaaS cloud in [4–6,14]. The existing logging system architecture in Figure 1 can log incidents that happened in a customer domU or VM, such as who has access to or what happens with a customer file in a disk of the VM [14, 15]. A logging system can compose of a logging process and a log file [14]. This paper will call the logging process a logger. The system architecture of the logging system is from our previous work [14] and is also illustrated in Figure 1. In this paper here, the architecture will be applied to the proposed experiment. The box inside the domU in Figure 1 is the threat-app process. For the purposes of the experiment, we assume that, somehow, this process can be controlled by an attacker. As a result, he/she can maliciously read a sensitive file or s.txt of an IaaS customer in diskU, and see the text file shape inside the diskU. A threat-app_mem in memU represents a reserve memory space for the threat-app process provided by the operating system (OS) that hosts this process. The right bottom shaded box in the dom0 is LibVMI, the new name of XenAccess [16]. It is a C library that is installed in the dom0. Then, it can access the threat-app_mem in memU to detect the malicious activities of the threat-app process that is reading s.txt. We did not focus on this reading s.txt action in this paper.

However, the action is also referred to in proposing logging solutions in the cloud. In [15, 17] consider this action when proposing their logging solutions in the cloud. For example, [15] assumes that when a user 'Alice' creates a sensitive file (such as s.txt) and modifies this file. Then, somehow, a user, 'Bob,' can maliciously read the file without Alice's permission. From Figure 1, the three working steps of the logger

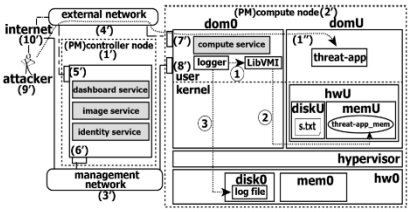


FIGURE 3. The architecture of the logging system in the OpenStack for IaaS, adapted from [3]

are represented by the circles with the numbers 1 to 3. Step 1, the logger in the dom0 calls LibVMI to access memU to obtain the logging data from the threat-app_mem (Step 2). This data includes i) a file name of s.txt or the string 's.txt', ii) the process ID of the threat-app process. Then, LibVMI accesses memU to obtain this data in the threat-app_mem. Then, it returns the obtained data to the logger. Finally, the logger manages the data and then writes (Step 3) the data into the log file.

2.3. **An OpenStack Architecture.** The IaaS cloud architecture and logging system architecture in Figure 1 are in a non-real-world production environment. This environment was built on one computer machine to simulate an IaaS cloud in [4, 14]. We will use OpenStack to simulate an IaaS public cloud to be the same as a real-world IaaS production environment. OpenStack is a cloud operating system that controls larger pools of storage, networking, and compute resources in all parts of a data center. The resources are managed via a dashboard that is a web interface [18]. OpenStack is also open-source software that can be used to build an IaaS public cloud, and it can be used in the real-world cloud production environment. It has increasingly been used in large business organizations [19]. Thus, we use OpenStack to simulate an IaaS public cloud to be the same as a real-world IaaS production environment. See the five boxes in Figure 2. There are five services of the architecture of an IaaS-OpenStack cloud. These services are the compute service, image service, object storage service, dashboard service, and identity service. All of these services are connected through the communication network; see the solid lines in Figure 2. The details of each service are mainly from the OpenStack official documentation website [20]. This paper relates to only three services. The first service is the compute service; see the box on the leftmost of Figure 2. The compute service can enable a provider to control an IaaS cloud. This service allows the provider to control instances (VMs/domUs) and networks and to handle access to the cloud by users. This service also defines drivers that interact with a hypervisor that is run on the host OS or dom0 of the provider cloud infrastructure. This service also publishes its functionality for other services via web-based OpenStack Application Program Interfaces (APIs). The second is the object storage service, the box on the rightmost of Figure 2. It is a long-term storage system for many static data, which can be updated and retrieved. The last service is the box on the top of Figure 2. It is the dashboard service, which is a web-based interface allowing the provider to manage OpenStack resources and services. It also interacts with the compute service via the APIs.

2.4. **The existing architecture of the logging system in the OpenStack for IaaS.** The architecture of the logging system in the OpenStack for IaaS was from our previous work [3] to prove that the logging system (Figure 1) can work in the OpenStack environment (Figure 2). In Figure 3, we illustrated the perspective of 'the architecture of the logging system in the OpenStack for IaaS'. We will call this architecture the ' logging system in IaaS-OpenStack cloud'. However, this figure without the logging components (logger, LibVMI, and log file) is called 'an IaaS-OpenStack cloud'. We run Ubuntu Server 16.04 LTS OS on a computer or hw0 as the compute node. Then we installed the OpenStack Ocata version in this OS. We mapped each service, except the object storage service, of the OpenStack architecture in Figure 2 onto two physical machines (PMs).

According to the functions in the installation manual of the OpenStack [21], one of the two physical computers is the compute node, and another one is the controller node. The dotted box with the number 1' on the left side of Figure 3 is the controller node for the execution of the three services of Figure 2. The services are the identity service, image service, and dashboard service; see the shaded rectangles in the dotted box. These services were described in Section 2.3. Then, the dotted box with the number 2' on the right side of Figure 3 is the compute node for the execution of the compute service from Figure 2. This service is used to control the hypervisor to manage dom0 and domUs. Both of the nodes use two network adapters (5' to 8') to connect to two networks. They are the management network (3') and the external network (4'). The management network is used for installing applications and for updating the

patch of the security of the OSes of the controller node and the compute node. In this IaaS-OpenStack environment, the external network is used for the customers to connect to domUs via the Internet or 10'. In the architecture, this external network may also be somehow a channel that the attacker or 9' can use this channel to connect to domUs with the credentials and passwords that the attacker may steal through phishing and fraud as argued in [14], see the dotted line with number 1". Now, we got our IaaS-OpenStack cloud.

Then, we simulated incidents that were assumed to be taken place by the attacker. Thus, the attacker can connect to a domU in this built IaaS-OpenStack cloud over the Internet using the credentials and password mentioned above. See the box in the user level of domU in Figure 3. Then, the attacker may use a 'threat-app' process to access the sensitive file in diskU. We aim to locate the existing logging system or the logger (see the logger box in the user level of dom0) and the log file in Figure 1 into this built IaaS-OpenStack cloud. So, this logger has been designed to detect a threat-app application (threat-app process). We consider the threat-app process as a target process that the logger wants to capture the appearance of this process for the detection of such incidents. Now, we got the architecture of the logging system in the OpenStack for IaaS, adapted from [3], see Figure 3. Then, this architecture can detect the threat-app process, as discussed in [3].

The IaaS-OpenStack cloud architecture can expand many compute nodes in the real-world IaaS production environment, as discussed in [3] and [22] has illustrated the expanding of the compute nodes as 1,000 nodes in an OpenStack architecture. The expansion can be done by adding other compute nodes into the architecture of the IaaS-OpenStack cloud in Figure 3. All new nodes' details are the same as a compute node in the architecture of the IaaS-OpenStack cloud. The new nodes can be set up based on OpenStack Installation Guide 15.0 in [21]. Thus, there can be more than one logger in these compute nodes, and executing the logger in each compute node is difficult to manage. Therefore, we will call this logging system in Figure 3 a decentralized logging system. Thus, we will enable a centralized architecture for a logging system in the IaaS-OpenStack cloud architecture or a centralized logging system. It is our primary objective.

3. Design and implementation.

3.1. The hardware and software of the experimental environment. This section will explain the experimental environment by expanding the information of Figure 3, just discussed in Section 2.4. This can form the experimental IaaS-OpenStack cloud. We then set up this cloud using two PMs. The first PM composes of Intel Core-i7 3.2 GHz 64-bit four cores, DDR3-SDRAM 8 GB of main memory, and 500 GB of secondary storage. This PM is the controller node; see number 1' in Figure 3. The controller node is configured with the default settings based on the OpenStack Installation Guide 15.0 [21]. The compute node is based on another PM of Intel Xeon 3.4 GHz with CPU 64-bit eight cores. This PM has DDR3-SDRAM 8 GB of main memory and 500 GB of secondary storage; see number 2' in Figure 3. A Ubuntu Server 16.04 LTS is the OS of both PMs. Then, we installed Xen 4.6.5 as the hypervisor in this compute node. This creates the dom0 in the figure. We also installed the LibVMI library (version 0.12-rc3), the shaded box inside the dom0 in Figure 5, in the compute node. Finally, in the compute node, we then created the domU in Figure 3, with 1 CPU core, 1 GB of memU, and 20 GB of diskU.

3.2. The aim of the new logging system architecture based on the socket programming method for IaaS. The new logging system architecture can do the centralized architecture for a logging system in the IaaS-OpenStack cloud based on the socket programming method for IaaS. This centralized architecture needs to modify the existing logging system in Figure 1. From the first research gap in Section 1 that the existing logging system in Figure 3 is a decentralized architecture. This architecture is difficult to manage, and this gap. It is because the IaaS-OpenStack cloud architecture in Figure 3 can be more than one compute node in the real-world IaaS production, as discussion in [3]. Thus, there can also be more than one logging system in these compute nodes.

3.2.1. *The components and parameters.* We will redesign the existing logging system in Figure 1 in Section 2.2 to support centralized architecture via a socket programming method and based on Figure 2 and Figure 3. The socket programming method or socket method is one of the best methods in distributed computing, as argued in [23]. Figure 4 is the new logging system architecture based on the socket programming method for IaaS. This figure is proof that all the existing logging components (LibVMI and log file) and the new-modified components (logger_c and logger_s) can be fitted in the IaaS architecture (or Figure 1). The architecture in Figure 4 is split into two physical machines. We call the first one a 'server-side', and the one a 'client-side'. All the components and their functionalities on the server-side are the same as the ones in Figure 1, except the logger_s. The logger_s in Figure 4 is different from the logger in Figure 1. It is because we added the socket method into the logger_s. This enables logger_s to exchange messages between itself (in a server-side physical machine) and logger_c (in

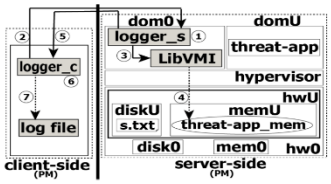


FIGURE 4. The new logging system architecture based on the socket programming method for IaaS, not in IaaS-OpenStack cloud

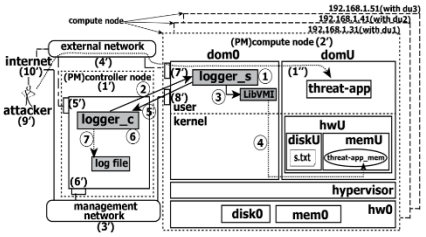


FIGURE 5. The proposed centralized architecture for a logging system in IaaS-OpenStack cloud

a client-side physical machine) across a network. A client-side is a physical machine with the components of logger_c and log file. The details of both are below. The client-side machine and the server-side machines can communicate via a local area network (LAN). The main logging components of the new logging system based on the socket method for IaaS are all gray boxes in Figure 4. They are logger_s, log file, logger_c, and LibVMI. A logger_s has been designed to detect threat-app processes in domU. It works the same as the logger in Figure 1. However, the logger_s is designed to receive a connection from logger_c from the client-side to exchange the data, such as domU's name and process name of the target process, such as threat-app. See the gray box on the top side of the client-side in Figure 4; it is logger_c that is designed to request then connect to logger_s in the server-side. See the gray boxes that are labeled with "LibVMI" and "log file". Both components were functioning as the Figure 1 and were described in Section 2.2. As mentioned above, Figure 4 is a new logging system architecture based on the socket programming method for IaaS or centralized logging system architecture.

3.2.2. *The working steps of the new logging system architecture.* There are seven working steps of the new logging system architecture based on the socket method for IaaS or centralized logging system architecture. Step 1 is when the logger_s was started and is waiting for a connection request from the logger_c. Step 2 is when the logger_c is successfully connected to the logger_s; then, the logger_c sends a message to the logger_s. The message includes i) a target domU's name in the server-side machine (such as 'du1') and ii) the target process's name in the domU (such as 'threat-app'). Step 3 is when the logger_s gets the message from the logger_c, and logger_s uses the message to set up the initial parameters for calling LibVMI. Then, Step 4 is when LibVMI retrieves appropriate logging data from the memU of the target domU based on the parameters. Step 5 is after the logger_s gets the appropriate logging data (the process' id and process' name of threat-app), the logger_s then sends this data to the logger_c. Step 6 is when the logger_c gets the data (message) from Step 5. The logger_c then pre-processes the data before writing the processed data to the 'log file' in Step 7.

3.3. **The proposed new logging system architecture in IaaS-OpenStack cloud with expanding the compute nodes.** Figure 5 without all the shaded boxes is the IaaS-OpenStack cloud with expanding the compute nodes. The IaaS-OpenStack cloud architecture and the expanding compute nodes were described in Section 2.4. In this experiment, we add the two additional compute nodes into the IaaS-OpenStack cloud in Figure 3. Thus, on the right side of Figure 5, expanding compute nodes are the dashed boxes with IP addresses ending with 41 and 51. The proposed new logging system architecture in the IaaS-OpenStack cloud with expanding the compute nodes is all the shaded boxes in Figure 5. This logging system has the main components and the location of these components as Figure 4. The dotted box with the number 1' on the left side of Figure 5 is the controller node as a physical computer in the IaaS-OpenStack cloud. First, we map the two components and location of the new logging system architecture in Figure 4, including logger_c and log file into the controller node in Figure 5. Then we map the component and location of logger_s and LibVMI in Figure 4 into the compute node; see the dotted box with the number 2' on the right side of Figure 5. Furthermore, each expanding compute node in Figure 5 has a logger_s, a LibVMI component, and the location of these components in side itself as the logger_s and LibVMI in the dotted box with the number 2' on the right side of Figure 5. The detail of the main components of the proposed new logging system architecture in the IaaS-OpenStack cloud with expanding the compute nodes are described in Section 3.2.1. Now, we got the proposed new logging system architecture in the IaaS-OpenStack cloud with expanding compute nodes.

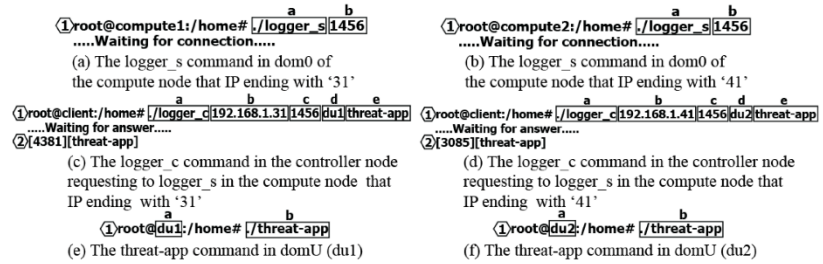


FIGURE 6. The command line patterns of logger_s, logger_c, and threat-app, and the commands' results

4. **The results.** The operational results of the proposed new logging system architecture of Figure 5 are shown by Figure 6 (Figures 6(a) to 6(f)). The figure shows the command line patterns of logger_s, logger_c, and threat-app, and the commands' results. The first part of the results are Figures 6(a) to 6(d), which were mainly generated from the six working steps of the new logging system architecture in Section 3.2.2. The second part of the results is Figures 6(e) to 6(f). They were mainly generated from the working steps of the threat-app, discussed in Section 2.2. In Figure 6(a), the line with number 1 is a command line pattern of the logger_s when it wants to capture the name and Id of the target monitored process or thread-app on the IaaS customer's domU. The logger_s in the box with 'a' has one parameter as the port number in the box with 'b' or 1456. This command is executed in dom0 by an authorized user. After the logger_s is executed, it will be waiting for a connection request from logger_c. Logger_c needs to be placed on a PM that connects with the same management network (see 3' in Figure 5) of the IaaS-OpenStack cloud. In Figure 6(c), the line with the number 1 is a command line pattern of the logger_c in the box with 'a'. This logger_c works with the logger_s together when it wants to monitor the target process or threat-app on the IaaS customer's domU. The logger_c has four parameters: internet protocol address (IP address), port number, domU's name, and target process's name. The IP address in the box with 'b' or '192.168.1.31' is a unique address that identifies a device on a local network of a compute node in the IaaS-OpenStack cloud. The port number in the box with 'c' or '1456' identifies a port of logger_s service on the compute node in the IaaS-OpenStack cloud. This port number needs to be the same number in the box with 'b' of Figure 6(a).

The domU's name in the box with 'd' or 'du1' is a target domU's name in the compute nodes in the IaaS-OpenStack cloud. Finally, the target process's name in the box with 'e' or 'threat-app' is a target process's name in the domU. The logger_s wants to monitor this name. The cooperation working steps between the logger_s and the logger_c were described in Section 3.2.2. Then, Figure 6(d) has the same command line patterns as Figure 6(c). The differences can be the parameters of boxes 'b', 'c', 'd', and 'e'. For example, Figure 6(d) has two different parameters from Figure 6(c). See boxes 'b' and 'd' of Figure 6(d); they are the IP ending with 41 (instead of 31) and du2 (instead of du1). Figure 6(e) is when the threat-app command is executed in domU or, see the line with number 1 in the figure. Then, the name and the Id of the threat-app appeared in the reserve memory space inside du1's memU. This space is called threat-app_mem; see the oval shape in Figure 1, already discussed in Section 2.2. Then, the logger_s (in Figure 6(a)) can capture the Id of threat-app as '4381' and the name of threat-app as 'threat-app'; see the line with number 2 of Figure 6(c). This is the same mechanisms with Figures 6(f),6(b), and 6(d). Note that, in our experiment environment, the 'threat-app' in Figure 6(e) and the one in Figure 6(f) have the same name. However, they are different malicious applications with different process Ids in different domUs (du1 and du2). Figure 6(d) illustrates that the logger_c is in the same physical computer logger_c in Figure 6(c). Logger_c in Figure 6(c) and Figure 6(d) is the same one and is placed in the same physical computer. This computer is called the controller node; see the dotted box with the number 1' on the left side of Figure 3. Thus, the proposed new logging architecture in the IaaS-OpenStack cloud with expanding the compute nodes in Figure 5 is a centralized logging system. This is because there is only one logger_c in the architecture. Then, this logger_c can make requests to, for example, two logger_s(s). One logger_s is in box 'a' of Figure 6(a), and another one is also in box 'a' of Figure 6(b), just discussed above.

5. **Discussions.**

5.1. **Easier management of the centralized compared to decentralized architectures.** This discussion is against the three research gaps discussed in Section 1. To be clearer for the discussions,

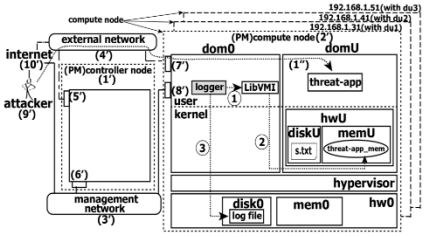


FIGURE 7. The existing decentralized architecture for a logging system in IaaS-OpenStack cloud

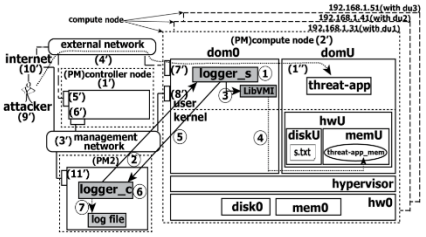


FIGURE 8. Separation of the workloads of the controller node

we modified Figure 3 to become Figure 7. From Figure 7, we can see that there are three PMs with three IPs ending with 31, 41, and 51. A logger (see the shaded box in the figure) is actually placed in each of the PMs. Thus, there will be three loggers for all the PMs. Consequently, these loggers are distributed (we called 'decentralized') into the whole architecture of Figure 7. The authorized user needs to configure and manage each logger one by one in this decentralized architecture. This should be difficult management and discussed in the introduction section as the main researcher gap. In Figure 5, there are actually also three loggers (we called each one 'logger_s') for all PMs, with three IPs ending with 31, 41, and 51. However, all these three logger_s(s) are managed by only one manager (called logger_c). Thus, this manager is not distributed (we called it 'centralized') into the whole architecture of Figure 5. The authorized user can only configure and manage all the logger_s with one point at the logger_c in this centralized architecture. This should be easier management, compared to Figure 7.

Moreover, based on the results in Section 4, Figures 6(a) to 6(f) illustrated the results of the implementation of the design of this centralized architecture in Figure 5. Especially, Figures 6(c) and 6(d) show that only one manager, which is logger_c, in boxes with 'a' from both figures. From the results in Section 4, this manager can manage two logger_s(s) to record/log the existence of the malicious processes in the two PMs, with two IPs ending with 31 and 41. An example of these processes is the 'threat-app' (see box 'b' in Figure 6(e)). This application is in domU called du1; see du1 in the box 'a' in Figure 6(e). Another example of these malicious processes is also the 'threat-app' (see box 'b' in Figure 6(f)) in du2 (see box 'a' in Figure 6(f)). To sum up, our proposed centralized architecture for a logging system in the IaaS-OpenStack cloud could be mitigated the main research gap, which is the difficulty of the management of the decentralized loggers in this cloud. We believe that it could be easier to manage centralized compared to decentralized architectures.

5.2. **Separation of the workloads of the controller node.** From Figure 5, we can see that the controller node is one PM (we will call PM1; see number 1' in Figure 5). This PM1 has the logger_c and log file (the shaded boxes in the controller node in Figure 5). We can actually relocate the logger_c and log file to place into the new available PM2; see the left-lower dotted box in Figure 8. This relocation can separate the workloads of the controller node, or PM1, from the workloads of the logging jobs. Thus, PM1 can perform only its own busy necessary main jobs (such as dashboard, image, identity service, etc.). Now, this PM1 has no management of the logging jobs with the logger_c and log file anymore. Briefly, this separation can be done by the two main steps. The first one is to move the logger_c and log file into PM2. The second step is to connect the PM2 to PM1's management network or to number 3' in Figure 5. The results of this connection are the management network or number 3' in Figure 8. The network allows PM2 to be connected to or worked with the other components of our proposed centralized architecture, shown by Figure 8. This enables PM1 and PM2 to separately perform their own native necessary tasks. To sum up, the final results of both steps are in Figure 8, which can be the separation of the workloads of the controller node. This could enhance the overall system management and performance of this architecture.

6. **Conclusions.** This paper aims to enable a centralized logging system in an Infrastructure as a Service (IaaS) cloud on a cloud operating system (OS) or environment. This paper focused on the OpenStack environment. The experiments confirmed that we successfully enabled the centralized logging system in this IaaS cloud. The results from the experiments indicate that we can use the socket programming

method to allow us to redesign the architecture of and relocate the main components of the existing logging systems. This enabled the decentralized and difficult-management logging system to be the centralized and easier-management logging system or 'a centralized logging system'. This paper makes the main contribution to the field of cloud security, which is a critical issue that prevents the adoption of the cloud. As discussed by and shown in Figure 8, the future work is to implement the separation of the workloads of the main physical machine (the controller node) in our centralized logging system.

REFERENCES

[1] CSA, "Top threats to cloud computing, version 1.0," The Cloud Security Alliance (CSA), Tech. Rep., 2010.

[2] ——, "Top threats to cloud computing pandemic eleven," The Cloud Security Alliance (CSA), Tech. Rep., 2022.

[3] W. Jaiboon, W. Wongthai, T. Phoka, and T. Auxsorn, "A logging system in openstack environment to mitigate risks associated with threats in infrastructure as a service cloud," *ICIC Express Letters*, vol. 14, no. 4, pp. 387–397, 2020.

[4] S. Wiriya, W. Wongthai, and T. Phoka, "The enhancement of logging system accuracy for infrastructure as a service cloud," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1558–1568, 2020.

[5] T. Auxsorn, W. Wongthai, T. Porka, and W. Jaiboon, "The accuracy measurement of logging systems on different hardware environments in infrastructure as a service cloud," *ICIC Express Letters, Part B: Applications*, vol. 11, no. 5, pp. 427–437, 2020.

[6] W. Wongthai, F. L. Rocha, and A. van Moorsel, "A generic logging template for infrastructure as a service cloud," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2013, pp. 1153–1160.

[7] P. Mishra, I. Verma, and S. Gupta, "Kvminspector: Kvm based introspection approach to detect malware in cloud environment," *Journal of Information Security and Applications*, vol. 51, p. 102460, 2020.

[8] A. A. R. Melvin, G. J. W. Kathrine, S. S. Ilango, S. Vimal, S. Rho, N. N. Xiong, and Y. Nam, "Dynamic malware attack dataset leveraging virtual machine monitor audit data for the detection of intrusions in cloud," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 4, p. e4287, 2022.

[9] V. Agrawal, D. Kotia, K. Moshirian, and M. Kim, "Log-based cloud monitoring system for openstack," in *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE, 2018, pp. 276–281.

[10] O. Sefraoui, M. Aissaoui, M. Eleuldj *et al.*, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[11] T. Kurek. (2022) Openstack is dead? the numbers speak for themselves.

[12] OpenStack.org. (2022) Openstack survey report.

[13] ibm.com. (2013) Ibm i version 7.2 programming socket programming.

[14] W. Wongthai, "Systematic support for accountability in the cloud," Ph.D. dissertation, Newcastle University, 2014.

[15] R. K. L. Ko, P. Jagadpramana, and B. S. Lee, "Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments," in *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011.

[16] B. D. Payne, M. D. P. D. A. Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 2007, pp. 385–397.

[17] S. Sundareswaran, D. Lin, and A. C. Squicciarini, "Ensuring distributed accountability for data sharing in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 04, pp. 555–567, 2012.

[18] OpenStack.org. (2018) What is openstack?

[19] Redhat. (2020) Digital transformation-the open source way.

[20] OpenStack.org. (2019) Welcome to openstack documentation.

[21] ——. (2017) Openstack installation tutoial for ubuntu.

[22] ——. (2015) Ocata openstack summit recap.

[23] R. Maata, R. Cordova, B. Sudramurthy, and A. Halibas, "Design and implementation of client-server based application using socket programming in a distributed computing environment," in *IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 2017, pp. 1–4.