การขยายขนาดการให้บริการแบบอัตโนมัติในระบบด็อกเกอร์
An Auto-scaling Mechanism in Docker System

ธีรภัทร์ ขุนเพ็ชร์(Teerapat Khunpech)[1] และสุชา สมานชาติ(Sucha Smanchat)[2]
[1]ภาควิชาการสื่อสารข้อมูลและเครือข่าย [2]ภาควิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
[1]s5707011858053@email.kmutnb.ac.th, [2]sucha.s@it.kmutnb.ac.th

บทคัดย่อ

ปัจจุบันมีเทคโนโลยีการทำเครื่องคอมพิวเตอร์เสมือนหลากหลายรูปแบบเพื่อให้มีการใช้งานเครื่องคอมพิวเตอร์ที่มีสมรรถนะสูงอย่างคุ้มค่าให้มากที่สุดและเทคโนโลยีลินุกซ์คอนเทนเนอร์(Linux Container: LXC) หรือด็อกเกอร์(Docker) กำลังได้รับความนิยมในการใช้งานเนื่องจากสามารถนำไปประยุกต์ใช้งานในรูปแบบการให้บริการต่าง ๆ ที่ไม่ยุ่งยาก เช่น การให้บริการเว็บไซต์ เป็นต้น อย่างไรก็ตามหากมีการให้บริการเพื่อตอบสนองกับผู้ใช้บริการเป็นจำนวนมากจนทำให้ทรัพยากรในการให้บริการไม่เพียงพอต่อการใช้งานจึงมีความจำเป็นที่จะต้องขยายขนาดการให้บริการ ผู้วิจัยจึงได้ทำพัฒนาสถาปัตยกรรมการขยายขนาดการให้บริการแบบอัตโนมัติในระบบด็อกเกอร์ขึ้น ให้มีความพร้อมในการให้บริการตลอดเวลาโดยใช้เทคโนโลยีด็อกเกอร์

งานวิจัยนี้ได้ดำเนินการพัฒนาสถาปัตยกรรมการขยายขนาดการให้บริการอัตโนมัติในระบบด็อกเกอร์ที่ประกอบด้วยส่วนการเฝ้าระวัง ส่วนการตัดสินใจ และส่วนการขยายขนาดการให้บริการโดยใช้ค่าของการใช้งานหน่วยประมวลผลมาเป็นค่าที่ใช้ในการตัดสินใจว่าจะขยายขนาดหรือลดขนาดและมีการทดสอบการทำงานและทดสอบความพร้อมในการให้บริการของกลไกที่ใช้ในงานวิจัยนี้ โดยกลไกสามารถลดและขยายขนาดการให้บริการได้ตามที่ผู้ใช้งานตั้งค่าไว้โดยมีข้อผิดพลาดจากการทำงานเล็กน้อย

คำสำคัญ: การขยายขนาดการให้บริการแบบอัตโนมัติ ด็อกเกอร์ ลินุกซ์คอนเทนเนอร์

Abstract

Currently, there are many options for virtualization of computer systems in order to maximize the utilization of physical computing hardware. Among these technologies, Docker has recently emerged as a leading Linux container implementation, which has gained its popularity for its easiness to be applied to various service applications such as website services. However, as more users access and use a Docker system, the computing resources initially provided for the Docker system may not be able handle such high workload causing the services to be irresponsive. Thus, there is a need for an auto-scaling mechanism to automatically allocate computing resources for the Docker system. This research proposes an architecture of an auto-scaling mechanism to provide high availability in the Docker system.

The proposed auto-scaling architecture is composed of a monitoring component, a scaling decision maker, and a scaling executor. The scaling decision is based on CPU utilization metric in order to scale-out and scale-in the computing resources. The evaluation result shows that the proposed architecture can satisfactorily performs auto-scaling for the Docker system according to the user-defined CPU utilization threshold.

Keyword: Auto-scaling Mechanism, Docker, Linux Container.

1. บทนำ

ปัจจุบันเทคโนโลยีการประมวลผลบนอินเทอร์เน็ตหรือคลาวด์ (Cloud) เป็นที่ได้รับความนิยมเป็นอย่างมาก เนื่องจากเครื่องคอมพิวเตอร์มีสมรรถนะในการประมวลผลที่สูงจึงมีผู้ให้บริการจำนวนมากเช่นกัน ด้วยเหตุผลนี้จึงเป็นข้อดีต่อกับผู้ที่ต้องการใช้งานเนื่องจากผู้ใช้งานสามารถเลือกการใช้งานได้ตาม

ต้องการ เช่น สามารถเลือกหน่วยความจำหลัก (Random Access Memory : RAM) หน่วยความจำสำรอง (Hard Disk Drive) และเครือข่ายคอมพิวเตอร์ (Network) เป็นต้น ให้มีความเหมาะสมในการใช้งาน และด้วยเทคโนโลยีที่คอมพิวเตอร์มีประสิทธิภาพสูงร่วมกับความเร็วของการใช้งานอินเทอร์เน็ตจึงทำให้ผู้ใช้งานสามารถที่จะจัดการสิ่งต่าง ๆ เช่น เลือกทรัพยากรเป็นต้น ได้ด้วยตัวเองในรูปแบบที่ผู้ให้บริการจัดเตรียมไว้ให้

การให้บริการต่างๆ ผ่านระบบคลาวด์นั้นจำเป็นที่จะต้องรักษาข้อกำหนดการให้บริการระหว่างผู้ให้บริการและผู้ใช้บริการในระบบคอมพิวเตอร์ หรือ Service Level Agreements (SLA) ให้มีคุณภาพที่พร้อมให้บริการอยู่ตลอดเวลา ปัจจัยที่จะทำให้ระบบไม่สามารถให้บริการได้ เช่น ทรัพยากรเครื่องคอมพิวเตอร์มีไม่เพียงพอต่อการให้บริการ หรือ แอพพลิเคชั่นมีการทำงานอันผิดพลาดจากการทำงานของตัวมันเอง เป็นต้น สิ่งหนึ่งที่เราสามารถรักษา SLA ให้มีความพร้อมใช้งานตลอดเวลานั้น คือการดูแลและจัดการทรัพยากรของระบบคอมพิวเตอร์ทั้งหมดให้มีการจัดสรรได้ตามความเหมาะสมต่อความต้องการใช้งาน

การเพิ่มจำนวนเครื่องคอมพิวเตอร์หรือเพิ่มทรัพยากรของเครื่องคอมพิวเตอร์ที่ให้บริการ นับว่าเป็นอีกวิธีการหนึ่งที่จะเพิ่มประสิทธิภาพให้กับระบบให้มีความสามารถรองรับกับจำนวนการร้องขอการใช้บริการทรัพยากรจากผู้ใช้บริการ การขยายทรัพยากร (Scaling) เพื่อรองรับการทำงานนั้นมีอยู่ด้วยกัน 2 รูปแบบ คือ แบบการขยายตามแนวนอน เป็นการเพิ่มปริมาณเครื่องคอมพิวเตอร์และอีกรูปแบบ คือ การขยายตามแนวตั้งซึ่งเป็นการเพิ่มประสิทธิภาพให้กับเครื่องคอมพิวเตอร์ เช่น การเพิ่มหน่วยความจำหลัก เพิ่มหน่วยการประมวลผล เป็นต้น โดยทั่วไปแล้วระบบที่มีกลไกการขยายความสามารถในการรอบรับการใช้งาน (Scalability) จะประกอบด้วย 3 ส่วนหลัก ๆ ได้แก่ ส่วนการเก็บข้อมูลทรัพยากรต่างๆ ของการใช้งาน (Agent) ส่วนของการเฝ้าระวังการใช้ทรัพยากรของเครื่องคอมพิวเตอร์ (Monitoring) และส่วนของการตัดสินใจในการขยายระบบ (Decision Maker) [1] กระบวนการนี้จะเป็นตัวขับเคลื่อนให้มีการตัดสินใจในการขยายการรองรับการใช้งาน

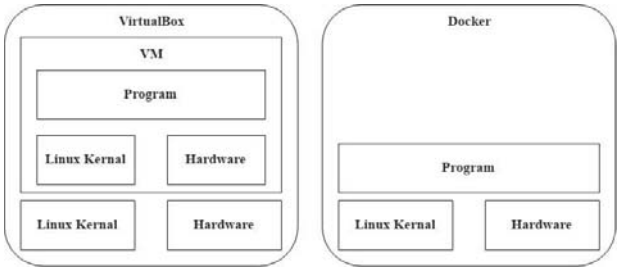
ดังนั้นด้วยความต้องการใช้งานบริการต่างๆ ผ่านทางอินเทอร์เน็ตที่มีมากขึ้นในยุคปัจจุบัน ทำให้ในบางครั้งการจัดการทรัพยากรต่างๆ ในรูปแบบการใช้งาน VM แบบเดิมอาจจะใช้เวลาค่อนข้างมากในการขยายการรองรับการให้บริการให้ตอบสนองความต้องการต่อการใช้งานได้ทันเวลา และเทคโนโลยีที่สะดวกและรวดเร็วต่อการสร้างคอมพิวเตอร์เสมือนด้วย LXC นั้นสามารถที่จะนำมาทดแทนการเพิ่มขยายในรูปแบบเดิมให้ตอบสนองต่อความต้องการของผู้ใช้งานและผู้ให้บริการ ดังนั้นผู้วิจัยจึงมีแนวคิดที่จะพัฒนากลไกการเพิ่มขยายนี้ให้มีรูปแบบการตรวจจับสถานะที่ต้องการทรัพยากรเพิ่มเติมจนถึงการสั่งงานให้มีการเพิ่มขยายการบริการแบบอัตโนมัติ เพื่อที่จะเป็นการลดค่าใช้จ่ายต่างๆ ที่จะเกิดขึ้นในส่วนของผู้ให้บริการได้และเพื่อให้มีการจัดสรรใช้งานทรัพยากรต่างๆ ของเครื่องคอมพิวเตอร์โดยเกิดประโยชน์สูงสุด

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ด็อกเกอร์ (Docker)

ด็อกเกอร์ (Docker) คือ ลินุกซ์คอนเทนเนอร์ชนิดหนึ่งซึ่งถูกพัฒนาเป็นโปรเจกต์โอเพ่นซอร์ส ใช้หลักการเดียวกันกับ Linux Container โดยมีจุดประสงค์ที่จะช่วยให้นักพัฒนาสามารถสร้างแอพพลิเคชั่น นำแอพพลิเคชั่นไปติดตั้งและสั่งให้แอพพลิเคชั่นทำงานได้ง่ายขึ้น โดยที่คอนเทนเนอร์ที่เกิดขึ้นมาแต่ละตัวนั้นจะแยกการทำงานออกจากกันอย่างอิสระ ไม่ว่าจะเป็นโปรเซสการทำงานและระบบเครือข่าย โดยใช้ฮาร์ดแวร์ตัวเดียวกัน โดยด็อกเกอร์กำลังได้รับความนิยมในการพัฒนาและการนำไปใช้งานในหลาย ๆ ด้าน ด้วยเหตุผลที่ว่ามีความยืดหยุ่นและมีความเร็วในการทำงานที่ค่อนข้างสูง [2]

ด็อกเกอร์มีการทำงานในสถาปัตยกรรมแบบแม่ข่ายและลูกข่าย (Client-Server) โดยจะมีด็อกเกอร์ไคลแอนต์ (Docker client) ทำหน้าที่เชื่อมต่อกับระบบประมวลผลด็อกเกอร์ (Docker engine) เพื่อที่จะสั่งให้แอพพลิเคชั่นต่างๆ ทำงานได้ ตัวระบบประมวลผลด็อกเกอร์นั้นสามารถรองรับการสั่งงานผ่าน RESTful APIs จึงทำให้ง่ายต่อการสั่งงานทางไกลหรือพัฒนาแอพพลิเคชั่นต่างๆ มาเรียกใช้งาน ด็อกเกอร์จะทำงานได้ต้องมีส่วนประกอบต่างๆ ดังต่อไปนี้ Docker host, Docker client, Docker images, Docker registries, Docker Containers และ Dockerfile ซึ่งด็อกเกอร์มีคุณสมบัติที่แตกต่างกับ Virtual Machine ต้องที่มีขนาดเล็กกว่า ทำให้ใช้ทรัพยากรที่น้อยกว่าอีกด้วย แสดงดังภาพที่ 1



ภาพที่ 1: การเปรียบเทียบระหว่าง VM กับ Docker [2]

2.2 Kubernetes

Kubernetes เป็นระบบที่บริษัท Google พัฒนาขึ้นมาเพื่อใช้สำหรับจัดการเกี่ยวกับลินุกซ์คอนเทนเนอร์ที่อยู่ในระบบสภาพแวดล้อมแบบกลุ่ม (Cluster) โดยหลักแล้วจะถูกนำมาใช้ในขั้นตอนส่วนของการนำเอาลินุกซ์คอนเทนเนอร์ไปใช้งาน (Deployment) ซึ่ง Kubernetes จะจัดการเรื่องระบบเครือข่ายและจัดภาระการทำงานให้กับลินุกซ์คอนเทนเนอร์ [3] ผู้ใช้งานสามารถสั่งงานผ่านเครื่องสั่งงานด้วยโปรแกรม kubectl และการให้บริการต่าง ๆ จะให้บริการบนเครื่องสั่งงาน ซึ่งรายละเอียดของ Kubernetes นั้นสามารถทำงานได้ด้วยส่วนประกอบดังต่อไปนี้ Master server, Worker Server และหน่วยการทำงานของ Kubernetes [4]

2.2.1 เครื่องสั่งงาน (Master server)

เป็นหน่วยในการควบคุมการทำงานของ Kubernetes แบบกลุ่ม ซึ่งผู้ที่จะใช้งานจะสั่งงานต่าง ๆ ผ่านทางเครื่องสั่งงานนี้ไปยังเครื่องที่ทำหน้าที่รับงาน (Worker server) โดยเครื่องสั่งงานจะมีโปรแกรมที่คอยตอบโต้กับระบบ Kubernetes ดังนี้

2.2.1.1 Etcd

Etcd จะเหมือนกับส่วน Etcd ของ CoreOS เป็นการให้บริการการเก็บข้อมูลแบบkey-value ซึ่งแต่ละเครื่อง Server จะใช้การให้บริการนี้เป็นตัวกลางในการสื่อสารกันภายในกลุ่ม เช่น สถานการณ์แสดงผลต่าง ๆ ของเครื่อง Server ภายในกลุ่ม โดยการใช้งาน Etcd นั้นจะมีการเก็บข้อมูลในรูปแบบของ HTTP/JSON เมื่อต้องการเรียกดูข้อมูลก็จะเรียกข้อมูลของ key ที่ต้องการและ Etcd จะตอบกลับค่าที่ต้องการกลับมาให้

2.2.1.2 API Service Server

ส่วนของ API Service Server ถือว่าเป็นส่วนสำคัญของเครื่องสั่งและจัดการงานต่าง ๆ ของเครื่องภายในกลุ่ม โดยจะอ่านค่าสถานะต่าง ๆ ขึ้นมาจากส่วน Etcd และการใช้งานการบริการ API Service Server นั้นสามารถใช้การเรียก RESTful ได้ ถือว่าเป็นข้อดีที่ทำให้นักพัฒนาสามารถที่จะพัฒนาเครื่องมือขึ้นมาติดต่อกับ API Service Server ได้เองโดยผ่าน RESTful และในงานวิจัยนี้ก็ได้พัฒนาโปรแกรมขึ้นมาด้วยการใช้ RESTful ติดต่อกับ API Service Server

2.2.1.3 Controller Manager Server

ส่วนการให้บริการในส่วนนี้จะใช้ในการจัดการงานเกี่ยวกับการสร้างคอนเทนเนอร์ โดยจะอ่านค่าสถานะต่าง ๆ ของคอนเทนเนอร์ขึ้นมาจากส่วน Etcd และส่วนนี้ก็จะมีหน้าที่ในการจัดการลดหรือขยายการให้บริการด้วยเช่นกัน

2.2.1.4 Scheduler Server

เป็นส่วนในการจัดการบริหารงานในแต่ละเครื่อง Server ภายในกลุ่ม ส่วนนี้จะตรวจสอบไม่ให้มีการใช้งานทรัพยากรเกินกว่าทรัพยากรว่างที่มีอยู่ เมื่อมีคอนเทนเนอร์เกิดขึ้นมาใหม่ ส่วนนี้จะจัดการให้ว่าเครื่องใดจะเป็นเครื่องที่รองรับคอนเทนเนอร์ตัวที่เกิดขึ้นมาใหม่นี้

2.2.2 เครื่องรับงาน (Worker Server)

เครื่องรับงานจะทำหน้าที่รอรับงานจากเครื่องสั่งงานโดยจะเชื่อมต่อผ่านระบบเครือข่าย บนเครื่องรับงานนี้จะมีโปรแกรมดีอกเกอร์ทำงานอยู่ เมื่อมีการสร้างงานขึ้นมาใหม่จากเครื่องสั่งงานแล้วเครื่องรับงานจะดำเนินการสร้างดีอกเกอร์คอนเทนเนอร์ขึ้นตามที่ได้รับงานมา ประกอบด้วยส่วนการทำงานดังต่อไปนี้

2.2.2.1 Kubelet Service

Kubelet จะเป็นส่วนที่ติดต่อรับข้อมูลกับเครื่องสั่งงานว่าเครื่องสั่งงานนั้นสั่งให้สร้างคอนเนอร์แบบใดมาและจะคอยรับคำสั่งต่าง ๆ จากเครื่องสั่งงาน

2.2.2.2 Proxy Service

จะทำหน้าที่ส่งการร้องขอการให้บริการจากดีอกเกอร์โฮสต์เข้าไปยังดีอกเกอร์คอนเทนเนอร์และส่งผลลัพท์กลับคืน ซึ่งส่วนของ Proxy service นี้สามารถที่จะจัดการการให้บริการในรูปแบบการกระจายงานได้ (Load Balancing)

2.2.3 หน่วยการทำงานของ Kubernetes

ในการทำงานของแอพพลิเคชั่นคอนเทนเนอร์ที่ทำงานอยู่ในสภาพแวดล้อม Kubernetes จะประกอบด้วยหน่วยการทำงานดังต่อไปนี้

2.2.3.1 Pods

Pods คือชื่อเรียกของลินุกซ์คอนเทนเนอร์ที่มีแอพพลิเคชั่นทำงานอยู่ ในหนึ่ง Pods อาจจะมีลินุกซ์คอนเทนเนอร์อยู่เป็นจำนวนหลายตัวก็ได้แต่โดยทั่วไปแล้วจะนิยมให้มีแค่หนึ่งตัวในการรองรับการให้บริการของแอพพลิเคชั่น และในแอพพลิเคชั่นก็อาจจะมีจำนวนมากกว่าหนึ่ง Pods ขึ้นอยู่กับความสามารถในการรองรับการให้บริการของบริการนั้น ซึ่งในงานวิจัยนี้จะเน้นไปในการลดและขยายการให้บริการแบบอัตโนมัติด้วยการเพิ่มและลดจำนวน Pods ที่จะเข้ามารองรับการให้บริการ

2.2.3.2 Services

Services คือชื่อเรียกส่วนการเชื่อมต่อกับลินุกซ์คอนเทนเนอร์ โดยส่วนนี้จะนำส่งการร้องขอใช้บริการต่าง ๆ ไปให้ยังลินุกซ์คอนเทนเนอร์ที่อยู่ใน Pods อาจจะกล่าวอีกอย่างหนึ่งว่า Services คือส่วนที่เป็นอินเตอร์เฟสไว้เชื่อมต่อระหว่างผู้ใช้บริการกับลินุกซ์คอนเทนเนอร์ที่ให้บริการ

2.2.3.3 Replication Controllers

Replication Controllers เป็นส่วนที่คอยจัดการ Pods ในเรื่องของการขยายขนาดการให้บริการในแนวนอน (Horizontal) โดยปกติเมื่อมีการสร้าง Pods ขึ้นแล้วส่วนของ Replication Controller จะถูกเปิดการใช้งานขึ้นมาพร้อมกัน หน้าที่อีกหนึ่งอย่างของ Replication Controllers คือการจัดการในเรื่องรุ่นของแอพพลิเคชั่น (Version) ที่ทำงานในคอนเทนเนอร์ เช่นหากเดิมมี Pods ที่ให้บริการแอพพลิเคชั่นในรุ่นที่หนึ่งและหากผู้ที่ให้บริการต้องการนำแอพพลิเคชั่นในรุ่นที่สองใช้งานแทนที่แอพพลิเคชั่นรุ่นที่หนึ่งทำงานอยู่ ส่วนของ Replication Controllers จะนำแอพพลิเคชั่นรุ่นที่สองขึ้นมาทำงานและเมื่อตรวจสอบแล้วว่าแอพพลิเคชั่นรุ่นสองทำงานได้แล้ว Replication Controllers ก็จะดำเนินการปิดแอพพลิเคชั่นรุ่นหนึ่งลงโดยอัตโนมัติ

2.3 ระบบฐานข้อมูลแบบอนุกรมเวลา (Time Series Databases)

ฐานข้อมูลที่มีความสัมพันธ์กับอันตรภาคเวลา โดยทั่วไปแล้วนิยมใช้กับการเก็บข้อมูลที่ต้องมีช่วงเวลามาเป็นตัวชี้วัดระดับ สำหรับงานวิจัยนี้ได้เลือกใช้ระบบจัดการฐานข้อมูลแบบอนุกรมเวลาที่มีชื่อว่า InfluxDB ซึ่งเป็นโปรแกรมโอเพ่นซอร์สที่มีความสามารถจัดการฐานข้อมูลแบบอนุกรมเวลา โดยสามารถจัดเก็บข้อมูลได้หลากหลายและมีความยืดหยุ่นในการเรียกดูหรือสืบค้นข้อมูลขึ้นมาแสดงผล การใช้งานโปรแกรมจัดการฐานข้อมูลอนุกรมเวลา InfluxDB สามารถใช้คำสั่ง SQL เหมือนกับการใช้งานโปรแกรมจัดการฐานข้อมูลเชิงสัมพันธ์ทั่วไปในการเรียกดูข้อมูลขึ้นมาแสดงผล และยังสามารถให้ผู้ใช้งานติดต่อกับฐานข้อมูลผ่านทาง HTTP RESTful ได้อย่างสะดวก [5]

2.4 การขยายขนาดการให้บริการ

2.4.1 การขยายการรองรับการใช้งานในยุคก่อนและหลังการใช้ Cloud

การขยายการรองรับการใช้งาน ได้มีการศึกษาและดำเนินการมาอย่างยาวนาน ปัจจัยที่มีผลต่อการพิจารณาการขยายรองรับการใช้งานนั่นคือค่าการทำงานของหน่วยประมวลผลและความสามารถของการใช้งานเครือข่ายข้อมูล (Bandwidth) เพื่อที่จะรองรับการทำงานของแอพพลิเคชั่นต่างๆ ได้อย่างต่อเนื่องและมีประสิทธิภาพ หากปัจจัยที่ได้กล่าวมานั้นไม่เพียงพอต่อการใช้งานก็จะดำเนินการจัดซื้อจัดจ้างอุปกรณ์เข้ามาเสริมใหม่เพื่อเตรียมความพร้อมให้บริการอยู่เสมอ ต่อมาในยุคของการเทคโนโลยีเครื่องคอมพิวเตอร์เสมือนซึ่งได้มีการใช้งานเทคโนโลยีคลาวด์เข้าด้วยนั้นยังคงมีการวัดประสิทธิภาพของหน่วยประมวลผลเช่นเดิม แต่จะมีการกำหนดช่วงของค่าที่เหมาะสมในการดำเนินการขยายความสามารถ คือ มีการตั้งค่าเส้นขีดบน (Upper Scale) และเส้นขีดล่าง (Lower Scale) [6] หากค่าของหน่วยประมวลผลเกินเส้นขีดบนเมื่อไรก็จะทำการขยายเครื่องคอมพิวเตอร์เพื่อรองรับการใช้บริการและหากค่าของหน่วยประมวลผลน้อยกว่าเส้นขีดล่างเมื่อไรก็จะทำการลดจำนวนของเครื่องคอมพิวเตอร์ลงเพื่อรักษาค่าใช้จ่ายต่าง ๆ ที่ได้เกิดขึ้นจากผู้ให้บริการคลาวด์

2.4.2 อัลกอริทึมในการขยายเครื่องคอมพิวเตอร์แบบอัตโนมัติ

อัลกอริทึมการขยายความสามารถของเครื่องคอมพิวเตอร์ขณะทำงานอยู่ [7] ได้นำตัวแปรในการตัดสินใจขยายความสามารถ เช่น ระยะเวลาของ SLA ลักษณะงาน และค่าประสิทธิภาพของหน่วยประมวลผลมาพิจารณา เป็นต้น เนื่องจากงานวิจัยดังกล่าวเกี่ยวข้องกับการขยายความสามารถ

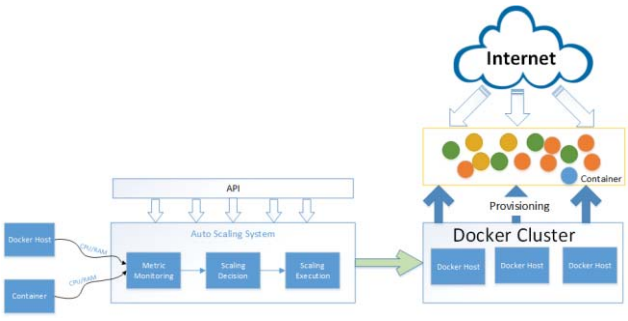
ของเครื่องคอมพิวเตอร์ตามลักษณะงานที่เข้ามาเป็นลำดับขั้น (Workflow) จึงต้องคำนึงถึงลักษณะงานและจัดลำดับความสำคัญของงานในการขยายความสามารถ ซึ่งอัลกอริทึมดังกล่าวได้ทดสอบใช้งานบน Amazon AWS EC2 พบว่าสามารถทำงานได้ตามที่ผู้วิจัยได้กำหนด สามารถที่จะเพิ่มและลดขนาดความสามารถของเครื่องคอมพิวเตอร์ที่ใช้ประมวลผลได้ตามระยะเวลาที่ต้องการ อันเนื่องมาจากข้อตกลง SLA แล้วค่าใช้จ่ายที่ได้เกิดขึ้นบนการใช้บริการคลาวด์

3. วิธีการดำเนินงาน

การออกแบบและพัฒนาระบบการขยายขนาดการให้บริการแบบอัตโนมัติในระบบด็อกเกอร์สามารถที่จะใช้งานระบบให้ทำงานร่วมกับการขยายขนาดของด็อกเกอร์คอนเทนเนอร์จำนวนหลายตัวที่อยู่ในระบบด็อกเกอร์โฮสต์เดียวกัน ผู้วิจัยได้แบ่งขั้นตอนในการดำเนินงานวิจัยออกเป็น 6 ขั้นตอน ดังต่อไปนี้ คือ

1) ออกแบบโครงสร้างของระบบด็อกเกอร์แสดงดังภาพที่ 2 ซึ่งประกอบด้วย

- Agent คือ ส่วนการเก็บข้อมูลทรัพยากรใช้งาน Metric
- Monitoring คือ ส่วนของการจัดเก็บค่าสถิติต่างๆ จากตัว Agent โดยออกแบบให้เก็บเป็นฐานข้อมูลแบบอนุกรมเวลา (Time Series Database)
- Scaling Decision คือ ส่วนของการตัดสินใจในการทำงานว่าควรจะดำเนินการขยายเพื่อรองรับการให้บริการหรือไม่ตามความเหมาะสมจากค่าที่ได้กำหนดไว้
- Scaling Execution คือ ส่วนของการดำเนินการสั่งงานให้มีการขยายเพื่อรองรับการให้บริการและมีการจัดสรรทรัพยากรได้อย่างเหมาะสม



ภาพที่ 2: โครงสร้างระบบรองรับการขยายตัวแบบอัตโนมัติในระบบด็อกเกอร์

2) ออกแบบส่วนในการเก็บข้อมูลการใช้งานทรัพยากรและการเฝ้าสังเกตข้อมูลการใช้งาน

ในส่วนของการเก็บข้อมูลทรัพยากรการใช้งานจะเป็นระบบที่มีชื่อว่า Heapster ซึ่งระบบ Heapster คือ ส่วนในการจัดการกับค่าของทรัพยากรต่างๆ โดยจะอ่านค่าจาก API ของด็อกเกอร์โฮสต์นั้นว่าแต่ละตัวมีการใช้งานทรัพยากรเป็นอย่างไร โดยดำเนินการอ่านค่าข้อมูลทรัพยากรการใช้งานทุก ๆ 1 นาที เมื่ออ่านค่าได้สำเร็จจะส่งค่าออกไปเก็บยังฐานข้อมูล ในที่นี้ได้เลือกใช้โปรแกรมจัดการระบบฐานข้อมูลแบบอนุกรมเวลา

ในส่วนของการเฝ้าสังเกตข้อมูลการใช้งานนั้นได้ออกแบบให้มีโปรแกรมที่ทำงานเป็นรอบเวลาโดยจะดึงข้อมูลเพื่อตรวจสอบทุก ๆ 1 นาที โดยจะทำการดึงข้อมูลออกจากฐานข้อมูลแบบอนุกรมเวลาที่ใช้ InfluxDB เป็นโปรแกรมจัดการฐานข้อมูล ซึ่งส่วนของการเก็บข้อมูลจะเขียนข้อมูลการใช้งานทรัพยากรลงในฐานข้อมูลนี้ โดยจะต้องพัฒนาส่วนในการดึงข้อมูลมาประมวลผลค่าสถิติและตัดสินใจว่าจะทำการขยายขนาดการให้บริการในช่วงเวลานั้นหรือไม่

3) ออกแบบส่วนการดำเนินการขยายหรือลดขนาดการให้บริการ เมื่อรับค่าข้อมูลจากการเฝ้าสังเกตแล้ว นำมาทำการคำนวณหาค่าของ CPU ที่ใช้งานในช่วงเวลานั้น ดังสมการที่ 3-1

$$cpu_i = \frac{cpu\_time_{(i,t)} - cpu\_time_{(i,t-1)}}{60 \times 10^9} \times core_i \qquad (3\text{-}1)$$

โดยที่ cpu_i คือ ค่าของการใช้งาน CPU ของ pod_i

$core_i$ คือ จำนวน CPU ที่ pod_i ใช้งานมีหน่วยเป็นมิลลิคอร์(millicore)

$cpu\_time_{(i,t)}$ คือ ค่าของการใช้งาน CPU ณ เวลา t หน่วยเป็นไมโครวินาที(microsecond)

$cpu\_time_{(i,t-1)}$ คือ ค่าของการใช้งาน CPU ณ เวลา t-1 หน่วยเป็นไมโครวินาที(microsecond)

จากนั้นทำการคำนวณหาค่า CPU เฉลี่ยของทุก Pods ที่ให้บริการในช่วงเวลานั้น เพื่อนำไปในการจำนวน Pods ดังสมการที่ 3-2

$$cpu_{avg} = \frac{\sum_{i=1}^{n_{current}} cpu_i}{n_{current}} \qquad (3\text{-}2)$$

โดยที่ cpu_{avg} คือ ค่าเฉลี่ยของการใช้งานหน่วยประมวลผลของ Pods นั้น

$n_{currect}$ คือ จำนวน Pods ที่ให้บริการอยู่ในปัจจุบัน

cpu_i คือ ค่าของการใช้งาน CPU ของแต่ละ Pods

หลังจากนั้นนำค่าที่ได้ไปคำนวณหาจำนวน Pods ดังสมการที่ 3-3

$$n_{new} = \left\lceil \frac{cpu_{avg}}{cpu_{target}} \right\rceil \times n_{current} - n_{current} \quad (3\text{-}3)$$

โดยที่ n_{new} คือจำนวนของ Pods ใหม่ที่จะดำเนินการขยายหรือลดขนาด

$n_{currect}$ คือ จำนวนของ Pods ในปัจจุบันที่ให้บริการ

cpu_{avg} คือ ค่าเฉลี่ยของการใช้งานหน่วยประมวลผลของ Pods นั้น

cpu_{target} คือค่าของหน่วยประมวลผลที่ผู้ใช้งานได้ใส่ค่าเข้ามาว่าจะต้องอยู่ในเกณฑ์ที่เท่าไร

4) ทดสอบระบบ จะเป็นการทดสอบการทำงานของระบบโดยการจำลองการใช้งานการบริการเพื่อสังเกตพฤติกรรมในการขยายขนาดการให้บริการโดยอัตโนมัติโดยพิจารณาว่าระบบสามารถทำงานได้ถูกต้องตามที่ได้วางแผนการทดสอบได้หรือไม่ โดยทำการจำลองเหตุการณ์ออกเป็น 3 เหตุการณ์ ดังนี้ เหตุการณ์ที่ 1 ไม่มีการใช้บริการ จำนวนผู้ใช้เท่ากับ 0 เหตุการณ์ที่ 2 มีการใช้บริการ จำนวนผู้ใช้บริการเท่ากับ 1600 หน่วงเวลาการใช้บริการ 5 นาที และเหตุการณ์ที่ 3 หยุดการใช้บริการ จำนวนผู้ใช้บริการเท่ากับ 0

5) ประเมินประสิทธิภาพ จะเป็นการประเมินประสิทธิภาพของการรองรับการให้บริการด้วยการขยายขนาดการให้บริการได้สัมพันธ์กับปริมาณการใช้บริการ โดยจำลองผู้ใช้งานเริ่มต้นที่จำนวน 800 ผู้ใช้งานต่อวินาที ทำการเรียกใช้บริการเว็บไซต์เป็นช่วงเวลาต่อเนื่อง และพิจารณาว่าการเรียกใช้งานเว็บไซต์นั้นสามารถแสดงผลได้อย่างถูกต้องไม่มีข้อผิดพลาด

4. ผลการดำเนินงาน

4.1 ผลการทดสอบส่วนการดำเนินการขยายขนาดการให้บริการ

ทดสอบการหาค่าจำนวนการขยายขนาดการให้บริการโดยอาศัยค่าที่ได้จากการคำนวณการใช้งานหน่วยประมวลผลของ Pods การทดสอบในกระบวนการนี้ได้ตั้งค่าเอาไว้ว่าเมื่อมีการใช้งานหน่วยประมวลผลของด็อกเกอร์คอนเทนเนอร์และ Pods มากกว่าร้อยละ 30 ให้ดำเนินการสั่งการขยายขนาดการให้บริการ แต่ในกระบวนการทดสอบนี้ติดปัญหาที่เครื่องคอมพิวเตอร์ที่ได้นำมาทำเป็นระบบคลัสเตอร์นั้นมีทรัพยากรไม่เพียงพอต่อการสร้างด็อกเกอร์คอนเทนเนอร์ที่มีจำนวนมากกว่า 40 คอนเทนเนอร์ จึงจำเป็นต้องจำกัดการขยายขนาดที่มากที่สุด คือ 40 คอนเทนเนอร์เพื่อให้โปรแกรมยังคงสามารถทำงานต่อได้

ตารางที่ 1: การทดสอบการทำงานการปรับขนาดการให้บริการ

นาที	CPU	จำนวน	เหตุการณ์	จำนวนผู้ใช้บริการ
0	0	1	ไม่มีการใช้บริการ	0
1	100	4	จำลองการใช้บริการ	1600
2	ช่วงหน่วงเวลาการขยายขนาด	4		1600
3		4		1600
4		4		1600
5	42.256	6		1600
6	ช่วงหน่วงเวลาการขยายขนาด	6		1600
7		6		1600
8		6		1600
9	29.098	6	จำลองการใช้บริการ	1600
10	29.039	6		1600
11	27.163	6		0
12	0	1	หยุดการจำลองการใช้บริการ	0
13	ช่วงหน่วงเวลาการลดขนาด	1		0
14		1		0
15		1		0
16		1		0

ตารางที่ 1 แสดงผลการทำงานเมื่อกำหนดให้มีการขยายขนาดการให้บริการอัตโนมัติเมื่อกำหนดให้มีการให้บริการเว็บไซต์ด้วยโปรแกรม Nginx บนด็อกเกอร์คอนเทนเนอร์หรือ Pods ที่มีจำนวนเท่ากับ 1 และเมื่อมีการใช้งานหน่วยประมวลผลมากกว่าร้อยละ 40 ให้ทำการขยายขนาดการให้บริการอัตโนมัติและเมื่อการใช้งานหน่วยประมวลผลน้อยกว่าร้อยละ 30 ให้มีการลดขนาดการให้บริการโดยเป็นไปตามค่าที่ได้จากการคำนวณ

จากการทดสอบในนาทีที่ 1 ได้มีการจำลองการเรียกใช้บริการจำนวนมากด้วยการจำลองให้มีผู้ใช้งานจำนวน 1600 หน่วยและทำการเรียกการใช้งานต่อเนื่องเป็นเวลา 5 นาที ผลลัพธ์ คือ ระบบได้มีการสั่งให้เพิ่มจำนวนของด็อกเกอร์คอนเทนเนอร์ขึ้น

เป็น 4 คอนเทนเนอร์และหลังจากนั้นจะมีช่วงเวลาในการหน่วงเวลาเพื่อรอการสร้างด็อกเกอร์คอนเทนเนอร์และความพร้อมในการเก็บข้อมูลการใช้งานทรัพยากร ในนาทีที่ 10 ได้มีการตรวจสอบใหม่อีกครั้งพบว่ามีการใช้งานทั้ง 4 ด็อกเกอร์คอนเทนเนอร์มีการใช้งานเฉลี่ยแล้วร้อยละ 42.256 จำเป็นต้องขยายขนาดการให้บริการเพิ่มขึ้นไปเป็น 6 คอนเทนเนอร์ ในนาทีที่ 12 ได้มีการตรวจสอบการใช้งานทรัพยากรอีกครั้งพบว่ามีการใช้งานหน่วยประมวลของด็อกเกอร์คอนเทนเนอร์ทั้ง 6 คอนเทนเนอร์เฉลี่ยอยู่ที่ร้อยละ 0 ประกอบมีการสิ้นสุดการเรียกใช้บริการจึงทำให้ไม่มีการใช้งานหน่วยประมวลผล เมื่อระบบตรวจสอบแล้วได้ผลว่าลดขนาดจำนวนการให้บริการลงเหลือ 1 คอนเทนเนอร์ และมีช่วงหน่วงเวลาการลดขนาดอยู่ที่ 4 นาที และหลังจากนาทีที่ 16 เป็นต้นไปนั้นไม่มีการใช้งานจึงคงจำนวนด็อกเกอร์คอนเทนเนอร์อยู่ที่ 1 คอนเทนเนอร์เช่นเดิม

4.2 ทดสอบความพร้อมของการให้บริการ

การทดสอบความสามารถในการให้บริการเว็บไซต์ด้วยโปรแกรม Nginx บนด็อกเกอร์คอนเทนเนอร์เป็นการทดสอบที่มีจุดประสงค์คือระบบการให้บริการต่างๆ ภายในระบบคลัสเตอร์เดียวกันจะต้องมีความพร้อมให้บริการตลอดเวลาและมีความราบรื่นในการเพิ่มหรือขยายขนาดการให้บริการซึ่งจะต้องเกิดข้อผิดพลาดให้น้อยที่สุดมี การทดสอบใช้เครื่องมือในการจำลองผู้ใช้งานด้วยการใช้โปรแกรม Apache Jmeter

การทดสอบได้จำลองผู้ใช้งานให้เพิ่มขึ้นและลดลงตามช่วงเวลาเพื่อทดสอบความพร้อมในการให้บริการของด็อกเกอร์คอนเทนเนอร์โดยกำหนดให้มีการขยายขนาดการให้บริการเมื่อมีระดับการใช้งานหน่วยประมวลผลมากกว่าร้อยละ 30 ของการใช้งาน ซึ่งการให้บริการนี้จะให้บริการเว็บไซต์และทดสอบด้วยการใช้โปรแกรมจำลองผู้ใช้งานเริ่มต้นที่จำนวน 800 ผู้ใช้งานต่อวินาทีโดยให้เรียกใช้บริการเว็บไซต์เป็นช่วงเวลาต่อเนื่องไป โดยจะมีการวัดผลการให้บริการคือ การตอบสนองสำเร็จหมายความว่าการเรียกใช้งานเว็บไซต์สามารถแสดงผลได้อย่างถูกต้องไม่มีข้อผิดพลาด (มีการแสดงผลการตอบสนองของรหัส HTTP เป็น 200 หมายความว่าสำเร็จ) เกิดขึ้นและการตอบสนองล้มเหลวหมายความว่าการเรียกใช้งานเว็บไซต์นั้นให้การแสดงผลที่ผิดพลาด เช่น การบริการมีความไม่พร้อมให้บริการ (มีการแสดงผลการตอบสนองของรหัส HTTP 503 หมายความว่าไม่พร้อมให้บริการ) มีผลที่ได้ดังต่อไปนี้

ตารางที่ 2: ผลการทดสอบการให้บริการ

เวลา (นาที)	จำนวนผู้ใช้งาน	ค่าเฉลี่ยการใช้ CPU	จำนวน Pods ปัจจุบัน	จำนวน Pods ที่ต้องขยาย/ลดขนาด (Target)	สำเร็จ (%)
1	800	0	1	1	100
2	800	43.727	1	2	100
5	800	55.653	2	4	100
8	1200	36.264	4	5	100
11	1200	28.54	5	5	100
12	1200	24.1	5	5	100
13	100	17.28	5	3	100
17	100	7.078	3	1	99.99
21	100	20.25	1	1	100
22	100	19.48	1	1	100
23	100	17.603	1	1	100
24	1500	100	1	4	100
27	1500	42.518	4	6	100
30	1500	32.339	6	7	100
33	1	0.27	7	1	100
37	1	0.27	1	1	100

จากตารางที่ 2 ได้แสดงผลของการทดสอบความพร้อมของการให้บริการในการใช้กลไกการขยายขนาดการให้บริการอัตโนมัติในระบบด็อกเกอร์โดยแสดงผลตามเวลา (นาที) จำนวนผู้ใช้งานจะเป็นค่าที่ถูกจำลองขึ้นมาจากโปรแกรมจำลองผู้ใช้งาน การใช้งานหน่วยประมวลผลจะเป็นค่าเฉลี่ยของหน่วยประมวลผลของด็อกเกอร์คอนเทนเนอร์ที่ทำงาน ในการให้บริการนั้น จำนวน Pods ปัจจุบัน คือ จำนวนของด็อกเกอร์คอนเทนเนอร์ที่ทำงานในการให้บริการอยู่ในช่วงเวลานั้น จำนวน Pods ที่ต้องขยายหรือลดขนาดคือจำนวนที่ได้จากการตัดสินใจในการขยายหรือลดขนาดของด็อกเกอร์คอนเทนเนอร์ ค่าสำเร็จคืออัตราส่วนของจำนวนการเรียกใช้บริการและได้ผลสำเร็จต่อจำนวนการเรียกใช้บริการทั้งหมด ค่าล้มเหลว คือ อัตราส่วนของจำนวนการเรียกใช้บริการและได้ผลล้มเหลวต่อ

จำนวนการเรียกใช้บริการทั้งหมดค่าที่ได้จากตารางที่ 2 สามารถอธิบายได้ดังต่อไปนี้ ในช่วงที่มีการเพิ่มขนาด

การให้บริการต่อผู้ใช้บริการที่เพิ่มขึ้นเพื่อให้สามารถรองรับการทำงานได้มีประสิทธิภาพในนาทีที่ 2, 5, 8 และ 24 จะพบว่ามีค่าเฉลี่ยของการใช้งานหน่วยประมวลผลที่เพิ่มขึ้นและมีการขยายขนาดของด็อกเกอร์คอนเทนเนอร์เพิ่มขึ้นตามจำนวนที่ได้จากการคำนวณและเมื่อทดสอบความสามารถในการให้บริการพบว่ามีอัตราสำเร็จร้อยละ 100 นั่นหมายความว่าเมื่อขยายขนาดการให้บริการจะไม่มีผลกระทบต่อการให้บริการที่ได้มีการให้บริการอยู่ก่อนหน้านี้และในช่วงที่มีการลดขนาดการให้บริการในนาทีที่ 13, 17 และ 33 ในนาทีที่ 17 จะพบว่าอัตราความพร้อมในการให้บริการมีค่าอยู่ที่ร้อยละ 99.99 สาเหตุที่เป็นเช่นนี้เพราะในการสร้างการให้บริการ (Service) ใน Kubernetes ได้มีการกำหนดให้มีลักษณะเป็นการกระจายงาน (Load balancing)ให้กับด็อกเกอร์คอนเทนเนอร์ที่อยู่ภายใต้ Replication Controller และในการลดจำนวนของด็อกเกอร์คอนเทนเนอร์ภายในการให้บริการนี้มีโอกาสที่จะพบช่วงที่กำลังลบด็อกเกอร์คอนเทนเนอร์ออกไปจึงทำให้ตัวกระจายงานนั้นจ่ายงานให้กับด็อกเกอร์คอนเทนเนอร์ที่กำลังโดนลบแต่เมื่อจ่ายงานให้แล้วไม่มีการให้บริการจึงได้ตอบสนองเป็นไม่พร้อมให้บริการ (รหัส HTTP คือ 503) ดังนั้นย่อมมีโอกาสที่จะเกิดเหตุการณ์เช่นนี้จึงทำให้การตอบสนองความพร้อมในการให้บริการนั้นมีความผิดพลาดอยู่ร้อยละ 0.01 ในช่วงเวลาดังกล่าว ในการทดสอบความพร้อมในการให้บริการเมื่อใช้กลไกการขยายขนาดการให้บริการอัตโนมัติในระบบด็อกเกอร์นั้นมีความสามารถที่จะให้บริการได้โดยมีความผิดพลาดเฉพาะระหว่างการลดขนาดของด็อกเกอร์คอนเทนเนอร์เพียงเล็กน้อยเท่านั้น

5. สรุป

งานวิจัยนี้ผู้วิจัยได้ตรวจจับการใช้งานทรัพยากรหน่วยประมวลผลของด็อกเกอร์คอนเทนเนอร์และนำค่าการใช้งานหน่วยประมวลผลมาเข้าสู่ระบบการเฝ้าระวังจากด็อกเกอร์คอนเทนเนอร์หลาย ๆ ตัวในระบบการทำงานแบบกลุ่มและตรวจสอบการใช้งานหากมีการใช้งานทรัพยากรมากกว่าค่าที่ผู้ใช้ได้ระบุในระบบแล้วส่วนของการขยายขนาดการให้บริการก็จะดำเนินการต่อไปโดยอาศัยค่าที่ได้จากการเฝ้าระวังมาคำนวณเพื่อตัดสินใจว่าจะขยายหรือลดขนาดการให้บริการเป็นจำนวนเท่าไรในระบบด็อกเกอร์

ผลการทดสอบการทำงานของระบบพร้อมกับทดสอบความพร้อมของการให้บริการเมื่อมีการขยายหรือลดขนาดแบบอัตโนมัติผลที่ได้ คือ ระบบมีการลดและขยายขนาดเป็นไปตามการใช้งานหน่วยประมวลผลและยังมีความพร้อมในการให้บริการของส่วนการให้บริการภายในด็อกเกอร์โดยที่มีความผิดพลาดเพียงเล็กน้อยเท่านั้น และในส่วนของกระบวนการเฝ้าสังเกตข้อมูลที่เพิ่มขึ้นมานั้น การใช้งาน CPU จะขึ้นอยู่กับจำนวน Pods หากมีการใช้งานของ Pods มากก็ส่งผลทำให้การใช้งาน CPU มากขึ้นเช่นกัน สำหรับงานวิจัยนี้มีการใช้งานจำนวน Pods ไม่มาก ทำให้การใช้งาน CPU น้อยและไม่มีนัยสำคัญ

เอกสารอ้างอิง

[1] Nikravesh, A.Y., S.A. Ajila, and L. Chung-Horng. Measuring Prediction Sensitivity of a Cloud Auto-scaling System. in Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International. 2014.

[2] Joy, A.M. Performance comparison between Linux containers and virtual machines. in Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in. 2015.

[3] Kubernetes. What is Kubernetes? Available online at https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[4] Key Concepts of Kubernetes. Available online at http://blog.arungupta.me/key-concepts-kubernetes/

[5] influxdata. Available online at https://www.influxdata.com/

[6] Han, R., et al., Lightweight Resource Scaling for Cloud Applications, in Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). 2012, IEEE Computer Society. p. 644-651.

[7] Heinze, T., et al., Auto-scaling techniques for elastic data stream processing, in Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. 2014, ACM: Mumbai, India. p. 318-321.