รายงานสหกิจศึกษาฉบับสมบูรณ์

การพัฒนา Kubernetes CronJob เพื่อเปลี่ยนราคา Spot price ในการใช้งาน AWS EC2 Spot Instance

The Development of Kubernetes CronJob for Changing AWS EC2 Spot Instance's Spot price

นายกมนณพ อรุณรัตน์

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

รายงานสหกิจศึกษาฉบับสมบูรณ์

การพัฒนา Kubernetes CronJob เพื่อเปลี่ยนราคา Spot price ในการใช้งาน AWS EC2 Spot Instance

The Development of Kubernetes CronJob for Changing AWS EC2 Spot Instance's Spot price

นายกมนณพ อรุณรัตน์

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

ชื่อโครงงานสหกิจศึกษา การพัฒนา Kubernetes CronJob เพื่อเปลี่ยนราคา Spot price ในการใช้งาน AWS EC2 Spot Instance

ชื่อ-สกุล นักศึกษา นายกมนณพ อรุณรัตน์

คณะ วิศวกรรมศาสตร์ **ภาควิชา** วิศวกรรมคอมพิวเตอร์

ชื่อ-สกุล อาจารย์นิเทศ อาจารย์บัณฑิต พัสยา

อาจารย์จิระศักดิ์ สิทธิกร

ชื่อ-สกุล ผู้นิเทศงาน นายกิตติพันธ์ ขันตีไตรรัตน์

ชื่อสถานประกอบการ บริษัท ออพซ์ตา (ประเทศไทย) จำกัด

บทคัดย่อ

ปัจจุบันมีการใช้ cloud computing เพื่อจุดประสงค์ด้านเทคโนโลยีสารสนเทศมากขึ้น เนื่องด้วยข้อดีกว่าการใช้งานเครื่องแม่ข่ายหลายประการ โดย cloud computing ดังกล่าวถูกนำมาใช้ในการทำงานหลากหลายรูปแบบ หนึ่งในนั้นคือการนำ cloud computing หลาย instance มาทำงานร่วมกันในรูปแบบของ Kubernetes cluster ซึ่งจะช่วยกระจายภาระงานให้ทุก instance ช่วยกันประมวลผลเพื่อรองรับผู้ใช้งานจำนวนมากหรือประมวลผลงานที่มีขนาดใหญ่ และเป็นที่แน่นอนว่าการใช้งาน cloud computing จำนวนมาก ย่อมต้องเสียค่าใช้จ่ายจำนวนมากเช่นเดียวกัน ทางเลือกหนึ่งในการลดค่าใช้จ่ายคือการใช้ AWS EC2 Spot Instance แต่นั่นยังไม่เพียงพอ เพราะราคา Spot price ของ AWS EC2 Spot Instance มีโอกาสเกิดการเปลี่ยนแปลงได้ตลอดเวลา หากตั้งราคา Spot price ที่ต้องการใช้งานไว้น้อยเกินไป ก็อาจไม่ผ่านราคาฐานของ Spot price ได้ หรือหากตั้งราคา Spot price ที่ต้องการไว้สูงเกินไปก็จะเป็นการสิ้นเปลืองโดยใช่เหตุ

โครงงานสหกิจศึกษาฉบับนี้จึงถูกคิดค้นขึ้นเพื่อลดค่าใช้จ่ายในการใช้งาน AWS EC2 Spot Instance ให้ได้มากที่สุด โดยการปรับเปลี่ยนราคา Spot price ที่ต้องการใช้งานให้มากกว่าราคาฐานของ Spot price ในปริมาณที่น้อยที่สุดที่ผู้ใช้พอใจ และยังลดภาระในการติดตามราคา Spot price เอง รวมถึงลดขั้นตอนการเปลี่ยนแปลงราคา Spot price ที่ต้องการซึ่งเป็นขั้นตอนที่ยุ่งยากและเกิดความผิดพลาดได้ง่าย ด้วยข้อดีเหล่านี้ จะทำให้การใช้งาน cloud computing แบบ Kubernetes cluster มีเสถียรภาพมากขึ้น ควบคู่ไปกับการลดค่าใช้จ่ายอีกด้วย

คำสำคัญ : Cloud computing, AWS EC2 Spot Instance, Kubernetes, Kubernetes CronJob, Docker, Golang

Co-operative Title: The Development of Kubernetes CronJob for Changing AWS EC2 Spot Instance's Spot price

Student Intern Name: Mr. Kamonnop Arunrat

Faculty: Engineering **Department:** Computer Engineering

Advisor Name: Mr. Bundit Pasaya

Mr. Jirasak Sittigorn

Mentor Name: Mr. Kittipun Khantitrirat

Company: Opsta (Thailand) Co.,Ltd.

ABSTRACT

Nowadays, the amount of people using cloud computing to serve the information technology purpose are increasing since there is much more advantages than using own's server machine. Cloud computing is being used in many ways. One of them is using many cloud computing instances as Kubernetes cluster to distribute workload among those instances. But also noted that using many cloud computing instances cost a great value of budget. One way to reduce the expense is to use AWS EC2 Spot Instance but that is not the best way since the Spot price can be changed anytime. If the Spot price of cluster is set too low, the AWS EC2 Spot Instance would cannot be used, or if the Spot price is set too high, it would be such a waste of budget.

This co-operative project was invented to reduce the cost of AWS EC2 Spot Instance usage by adjusting the desired Spot price to satisfy both base Spot price and user's budget. Also, this project could help reducing effort to monitor the changes in Spot price and reduce the steps to change Spot price of cluster which is complicate and may easily lead to error. With these advantages, this project could help using Kubernetes cluster much more stable along with reduce the expense in using AWS EC2 Spot Instance.

Keywords : Cloud computing, AWS EC2 Spot Instance, Kubernetes, Kubernetes CronJob, Docker, Golang

กิตติกรรมประกาศ

โครงงานสหกิจศึกษาฉบับนี้สามารถสำเร็จลุล่วงไปด้วยดี ด้วยคำแนะนำและคำปรึกษาจาก คุณจิรายุส นิ่มแสง ผู้ก่อตั้งและ CEO บริษัทออพซ์ตา (ประเทศไทย) จำกัด ,คุณกิตติพันธ์ ขันติไตรรัตน์ ผู้นิเทศงานสหกิจศึกษา และ อาจารย์บัณฑิต พัสยา ที่ให้ความรู้และแนวทางในการทำโครงงานชิ้นนี้ อันเป็นแนวทางชี้นำให้เกิดปัญญาและชี้แนะจุดบกพร่องของโครงงานสหกิจศึกษาให้สมบูรณ์มากขึ้น ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์จากท่านอาจารย์และขอบพระคุณเป็นอย่างสูง

ขอขอบคุณอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ผู้ซึ่งให้ความรู้ตลอดช่วงการศึกษา ที่เป็นส่วนช่วยในการทำให้วิทยานิพนธ์นี้สำเร็จมาได้ด้วยดี

สุดท้ายนี้ ข้าพเจ้าขอขอบคุณบุพการี และญาติของข้าพเจ้า ที่คอยให้กำลังใจในการทำโครงงานสหกิจศึกษาฉบับนี้

กมนณพ อรุณรัตน์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ในปัจจุบัน ผู้ใช้งานระบบเทคโนโลยีสารสนเทศมีจำนวนมากขึ้นและมีแนวโน้มการใช้งานเทคโนโลยีสารสนเทศมากขึ้นเช่นเดียวกัน ทำให้ผู้ให้บริการด้านเทคโนโลยีสารสนเทศต้องรองรับการใช้งานที่มากขึ้น ด้วยข้อเท็จจริงที่การให้บริการแอปพลิเคชันด้านเทคโนโลยีสารสนเทศต้องใช้คอมพิวเตอร์ระดับเครื่องแม่ข่าย เมื่อผู้ใช้บริการเยอะขึ้น จึงจำเป็นต้องมีการลงทุนด้าน infrastructure เพิ่มขึ้นเป็นปกติ

ด้วยเหตุผลหลายอย่างที่ทำให้ผู้ให้บริการแอปพลิเคชันไม่สามารถลงทุนด้าน infrastructure ได้ อาทิ ไม่มีพื้นที่สำหรับเก็บคอมพิวเตอร์แม่ข่าย, งบประมาณในการบำรุงรักษาไม่เพียงพอ, บุคลากรขาดความรู้ความสามารถในการติดตั้งและดูแลระบบแม่ข่าย, อัตราความต้องการเครื่องแม่ข่ายมีมากกว่าความสามารถในการจัดหาเครื่องแม่ข่าย, ฯลฯ เหตุผลเหล่านี้ทำให้ผู้ให้บริการแอปพลิเคชันบางส่วนหันมาพึ่งพาเทคโนโลยี cloud computing ในลักษณะของ infrastructure-as-a-service (IaaS) ซึ่งมีข้อดีทดแทนเหตุผลข้างต้น อาทิ ไม่ต้องติดตั้งและดูแลรักษาเอง, ไม่ต้องการพื้นที่ในการเก็บเครื่องแม่ข่าย, สามารถเพิ่ม-ลดจำนวนของเครื่องแม่ข่ายได้ตามต้องการตลอด 24 ชั่วโมง เป็นต้น เมื่อสามารถใช้งานเครื่องแม่ข่ายได้จำนวนมากแล้ว สิ่งต่อไปที่ผู้ให้บริการแอปพลิเคชันจะทำเพื่อใช้ความสามารถของเครื่องแม่ข่ายจำนวนมากเหล่านี้ออกมาคือ การทำคลัสเตอร์คอมพิวเตอร์ ด้วยจุดประสงค์เพื่อให้เครื่องแม่ข่ายแบ่งภาระกันประมวลผลการใช้งานของผู้ใช้จำนวนมาก หรือเพื่อให้แอปพลิเคชันมีคุณสมบัติ high availability เป็นต้น

อนึ่ง การใช้งาน cloud computing ก็ยังต้องเสียค่าใช้จ่ายอยู่เช่นกัน โดยปกติมักคิดราคาเป็นรายชั่วโมง (0.0528 ดอลล่าร์สหรัฐอเมริกาต่อชั่วโมงสำหรับ AWS EC2 instance ขนาด 2vCPU, 4.0GiB memory[1]) ด้วยเหตุนี้ AWS จึงเปิดให้บริการ AWS EC2 แบบ Spot Instance เพิ่มขึ้นด้วย ซึ่งบริการดังกล่าวจะเป็นการนำทรัพยากรที่เหลือของเครื่องแม่ข่ายที่กำลังให้บริการ AWS EC2 มาเปิดให้ประมูลราคาในการใช้งานโดยที่ทาง AWS อ้างว่า สามารถลดราคาของการใช้งานจาก instance EC2 ปกติได้สูงสุดถึง 90% (AWS EC2 แบบ Spot Instance ขนาด 2vCPU, 4.0GiB มีอัตราค่าบริการเพียง 0.0158 ดอลล่าร์สหรัฐอเมริกาต่อชั่วโมงซึ่งมีราคาถูกกว่าการใช้บริการ AWS

[1] AWS EC2 instance ประเภท t2.medium ในภูมิภาคเอเชียแปซิฟิก(สิงคโปร์) ข้อมูล ณ วันที่ 5 ธันวาคม 2561

EC2 แบบปกติประมาณ 76% [2]) แต่ต้องแลกกับการที่ AWS จะสามารถปิด หรือทำลาย Spot Instance ที่กำลังใช้งานอยู่หากเครื่องแม่ข่ายที่ให้บริการ AWS EC2 ต้องการทรัพยากรกลับคืน หรือมีการเปลี่ยนแปลงราคาประมูลของ AWS EC2 Spot Instance ที่สูงขึ้นเกินกว่าราคาที่ผู้ใช้ประมูลไว้

เพื่อป้องกันปัญหาที่อาจเกิดขึ้นจากการปิดหรือทำลาย AWS EC2 Spot Instance ที่กำลังใช้งานอยู่ จึงเป็นที่มาของการพัฒนา Kubernetes CronJob สำหรับเปลี่ยนแปลงราคาประมูล Spot Instance ให้มีค่ามากกว่าราคาประมูลขั้นต่ำที่ AWS กำหนดไว้โดยอัตโนมัติ

1.2 วัตถุประสงค์ของการวิจัย

เพื่อศึกษาขั้นตอนวิธีที่เหมาะสม และพัฒนา Kubernetes CronJob สำหรับเปลี่ยนแปลงราคาประมูลของ AWS EC2 Spot Instance ที่กำลังใช้งานอยู่โดยอัตโนมัติ

1.3 ขอบเขตของการวิจัย

ดำเนินการศึกษาขั้นตอนวิธีที่เหมาะสม และทำการพัฒนา CronJob ด้วยภาษา Go เวอร์ชัน 1.11.2 และดำเนินการทดลองบน AWS EC2 Spot Instance ชนิด t2.medium (2vCPU, 4.0GiB memory) จำนวน 8 instances , AMI หมายเลข ami-0856dbf77f7ffb494, Kubernetes เวอร์ชัน 1.10.6

1.4 วิธีดำเนินการวิจัย

ขั้นตอนวิธีดำเนินการวิจัย สามารถสรุปออกมาได้เป็น Action Plan ได้ดังนี้

[2] AWS EC2 Spot Instance ประเภท t2.medium ในภูมิภาคเอเชียแปซิฟิก(สิงคโปร์) ข้อมูล ณ วันที่ 5 ธันวาคม 2561

ตารางที่ 1.1 Action plan ของการวิจัย

หัวข้องาน / Assignments	เดือนที่ 1 1st Month	เดือนที่ 2 2nd Mont	เดือนที่ 3 3rd Mont	เดือนที่ 4 4th Mont	เดือนที่ 5 5th Mont	เดือนที่ 6 5th Mont
1. รวบรวมและวิเคราะห์ requirements	↔					
2. อ่านข้อมูล API ของระบบที่เกี่ยวข้อง (AWS EC2 และ Kubernetes)	↔	↔				
3. ทดลองใช้งาน API ที่เกี่ยวข้อง	↔	↔				
4. เขียนสคริปท์เพื่อเรียกใช้งาน API		↔	↔			
5. ทดสอบการทำงานของสคริปท์		↔	↔			
6. นำสคริปท์ไปทำ Docker image				↔		
7. ทดสอบ Docker image				↔		
8. นำ Docker image ไปทำ CronJob deployment template					↔	
9. ทดสอบ deploy script ลง บนKubernetes cluster					↔	↔
10. สรุปผล นำเสนอ และเขียนเอกสาร						↔

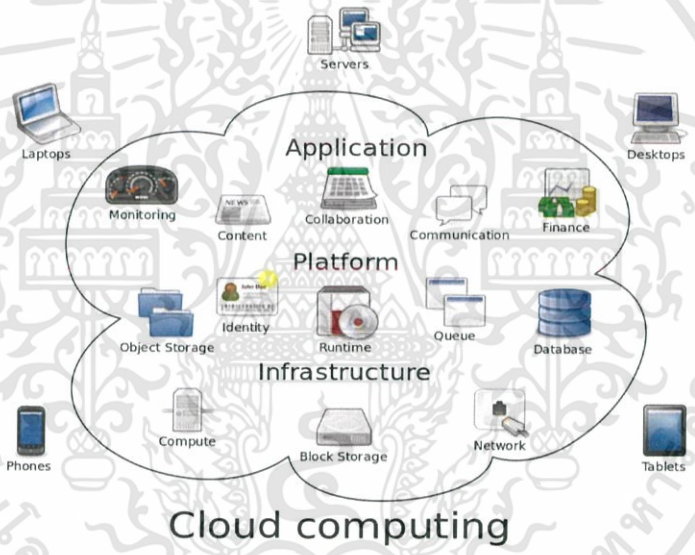
1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ศึกษาโครงสร้างภาษา และไวยากรณ์การเขียนโปรแกรมด้วยภาษา Go
2. ได้ศึกษาคุณลักษณะและใช้งานผลิตภัณฑ์ AWS สำหรับการใช้งาน cloud computing ในลักษณะ Infrastructure-as-a-service
3. ได้ศึกษาวิธีการสร้าง Docker image จากแอปพลิเคชันที่เขียนขึ้นใหม่
4. ได้ศึกษาการเขียน Kubernetes CronJob Kubemanifest file สำหรับการ deploy CronJob ลงบน Kubernetes cluster
5. ช่วยลดค่าใช้จ่ายที่เกิดจากการใช้งาน AWS EC2 ซึ่งใช้เป็น Kubernetes cluster

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

Cloud computing เป็นเทคโนโลยีชนิดหนึ่งที่เปิดให้ผู้เข้าใช้งานสามารถเข้าถึงทรัพยากรด้านระบบการประมวลผลแบบใช้ร่วมกัน และบริการในระดับที่สูงกว่า (เช่น แอปพลิเคชัน) ซึ่งสามารถจัดเตรียมได้อย่างรวดเร็วและจัดการง่าย โดยมากมักเป็นการใช้งานผ่านระบบอินเทอร์เน็ต การนำ cloud computing มาใช้จะช่วยให้องค์กรสามารถเน้นการพัฒนาธุรกิจหลักขององค์กรแทนที่จะนำเวลาและทรัพยากรในส่วนดังกล่าวมาลงทุนด้านcomputer infrastructure



ภาพที่ 2.1 การประยุกต์ใช้ทั้งสามระดับของ cloud computing

ที่มา : it24hrs. 2018. Cloud Computing คืออะไร ? Cloud Computing ดีอย่างไร ?

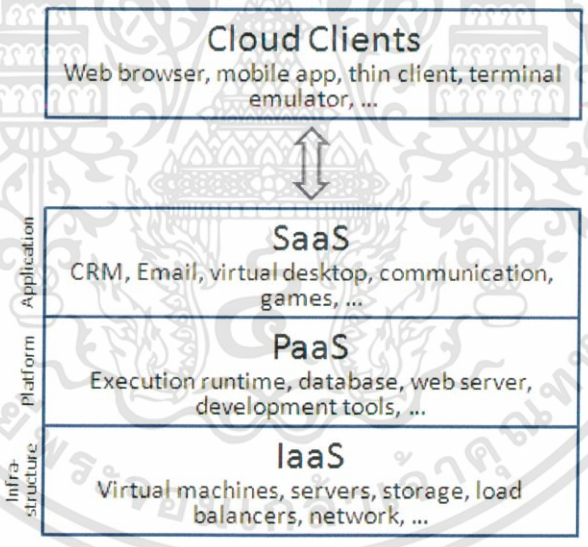
นอกจากนี้ cloud computing ยังถูกแบ่งออกเป็นสามระดับภายใต้แนวคิด “everything as a service (EaaS)” กล่าวคือ ผู้ให้บริการ cloud computing จะให้ “บริการ” ตามทรัพยากรต่าง ๆ ที่ผู้ใช้บริการต้องการ ได้แก่

Infrastructure as a service (IaaS) เป็นขั้นที่ผู้ให้บริการจะบริการ computer infrastructure เช่น หน่วยประมวลผล, หน่วยความจำ, หน่วยเก็บข้อมูล, ฯลฯ ในรูปแบบเสมือน รวมถึงอำนวยความสะดวกในการจัดการทรัพยากรดังกล่าว เช่น สร้าง แก้ไข เพิ่ม/ลดขนาด จัดการ ลบ ซึ่งจะลดความยุ่งยากในการดูแลรวมถึงลดงบประมาณที่ต้องใช้ในการจัดซื้ออุปกรณ์เพื่อ

วัตถุประสงค์ดังกล่าว ตัวอย่างผลิตภัณฑ์ cloud computing ในกลุ่มนี้ได้แก่ Google Cloud Platform, Amazon Web Service, Digital Ocean, Dropbox เป็นต้น

Platform as a service (PaaS) เป็นขั้นที่ผู้ให้บริการจะบริการสภาพแวดล้อมสำหรับการพัฒนา ทดสอบ และรันแอปพลิเคชันที่ผู้ใช้บริการพัฒนาขึ้นมาเอง โดยสิ่งที่เพิ่มมาจากระดับ infrastructure as a service คือซอฟต์แวร์ที่จำเป็นในการทำงานของแอปพลิเคชันดังกล่าว และชุดคำสั่งอำนวยความสะดวกในการจัดการแอปพลิเคชัน ตัวอย่างผลิตภัณฑ์ในกลุ่มนี้ได้แก่ Google App Engine, Amazon Lambda, Heroku เป็นต้น

Software as a service (SaaS) เป็นระดับของ cloud computing ที่สามารถเข้าถึงได้ง่ายที่สุด ด้วยลักษณะที่เป็นการให้บริการแอปพลิเคชันสำเร็จรูปผ่านระบบอินเทอร์เน็ตเพื่อวัตถุประสงค์ต่าง ๆ โดยที่สามารถเข้าถึงได้ง่ายผ่านอุปกรณ์หลายชนิด และสามารถเข้าถึงจากสถานที่ใดก็ได้ขอเพียงมีการเชื่อมต่ออินเทอร์เน็ตอยู่เท่านั้น ตัวอย่างผลิตภัณฑ์ในกลุ่มนี้ได้แก่ Gmail, Google Docs, Microsoft Office 365 เป็นต้น



ภาพที่ 2.2 แนวคิดทั้งสามระดับของ cloud computing

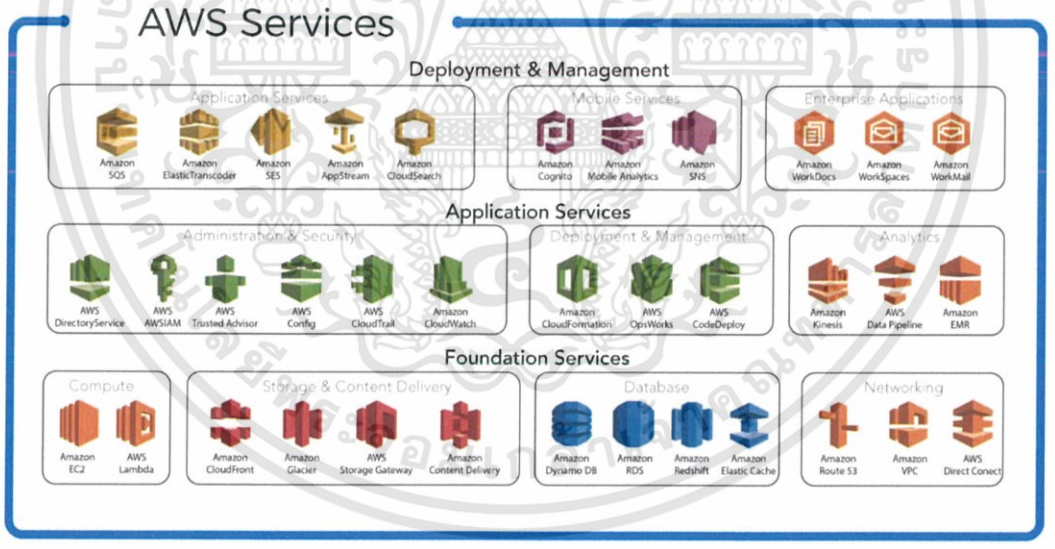
ที่มา : it24hrs. 2018. Cloud Computing คืออะไร ? Cloud Computing ดีอย่างไร ?

Amazon Web Services (AWS) เป็นบริษัทลูกของผู้ให้บริการ e-commerce ชื่อดัง "Amazon.com" ซึ่งให้บริการ cloud computing solutions ในระดับ infrastructure as a service และ platform as a service โดยให้บริการทั้งภาครัฐและเอกชน

AWS ให้บริการคลัสเตอร์คอมพิวเตอร์เสมือนผ่านระบบอินเทอร์เน็ต โดยที่ AWS สามารถจำลองส่วนประกอบของคอมพิวเตอร์จริงถึงระดับ CPU และ GPU, RAM, หน่วยบันทึกข้อมูล และมีตัวเลือกด้านระบบปฏิบัติการ, แอปพลิเคชันที่ติดตั้งเตรียมพร้อมให้ เช่น web server, ฐานข้อมูล เพื่อให้ลูกค้าเลือกใช้ให้ตรงกับความต้องการของตนเอง นอกจากนี้ AWS ยังเปิดช่องทางหน้ารวมการจัดการให้ลูกค้าเข้าถึงง่ายผ่านหน้าเว็บและมีหน้า console I/O เสมือน (คีย์บอร์ด, จอแสดงผล และเมาส์)

AWS มี server farm สำหรับให้บริการลูกค้าทั่วโลก ค่าบริการจะถูกคิดตามการใช้งานในแต่ละส่วนรวมกัน ได้แก่ ฮาร์ดแวร์ ระบบปฏิบัติการ ซอฟต์แวร์ ระบบเครือข่าย ระดับHigh Availability (availability, redundancy, security)

ณ ปี 2017 AWS มีผลิตภัณฑ์ที่ให้บริการกว่า 90 ผลิตภัณฑ์ ครอบคลุมเทคโนโลยีหลายด้าน เช่น การคำนวณ, หน่วยเก็บข้อมูล, ระบบเครือข่าย, ฐานข้อมูล, ระบบวิเคราะห์ข้อมูล, อุปกรณ์สำหรับการพัฒนาแอพลิชัน, อินเทอร์เน็ตของสรรพสิ่ง โดยบริการที่เป็นที่นิยมที่สุดก็คือ Amazon Elastic Compute Cloud (EC2) และ Amazon Simple Storage Service (S3)



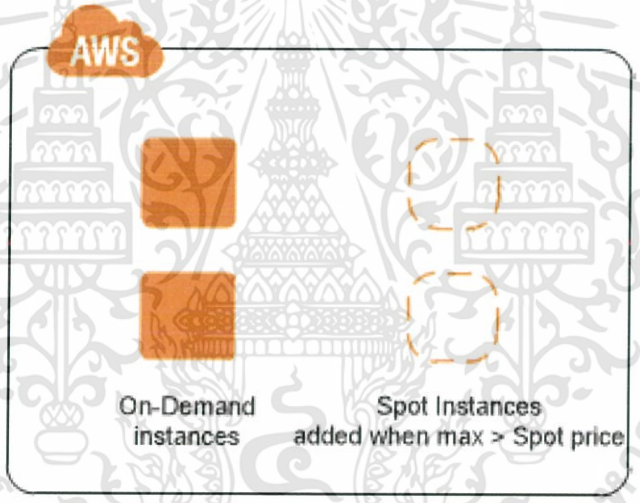
ภาพที่ 2.3 ผลิตภัณฑ์และบริการบางส่วนของ AWS

ที่มา : Westcon Comstor. 2018 Amazon Web Services

AWS EC2 Spot instance เป็นหนึ่งในแผนการให้บริการ Amazon Elastic Compute Cloud (EC2) ของ AWS โดยเป็นการนำทรัพยากรของเครื่องแม่ข่ายส่วนที่เหลือจากการให้บริการ

AWS EC2 มาลดราคาและให้บริการแก่ลูกค้าที่ต้องการ โดยทาง AWS กล่าวว่าลูกค้าสามารถใช้บริการ AWS EC2 ได้มากกว่าเดิมสิบเท่าตัวภายใต้งบประมาณเท่ากัน แต่ต้องแลกกับการที่เมื่อ AWS EC2 ต้องการทรัพยากรส่วนที่ให้บริการ AWS EC2 Spot Instance คืน instance ที่กำลังใช้บริการ AWS EC2 Spot Instance อยู่จะต้องปิดตัวลง ดังนั้น การใช้งาน AWS EC2 Spot Instance จึงเหมาะกับแอปพลิเคชันที่มี fault-tolerant

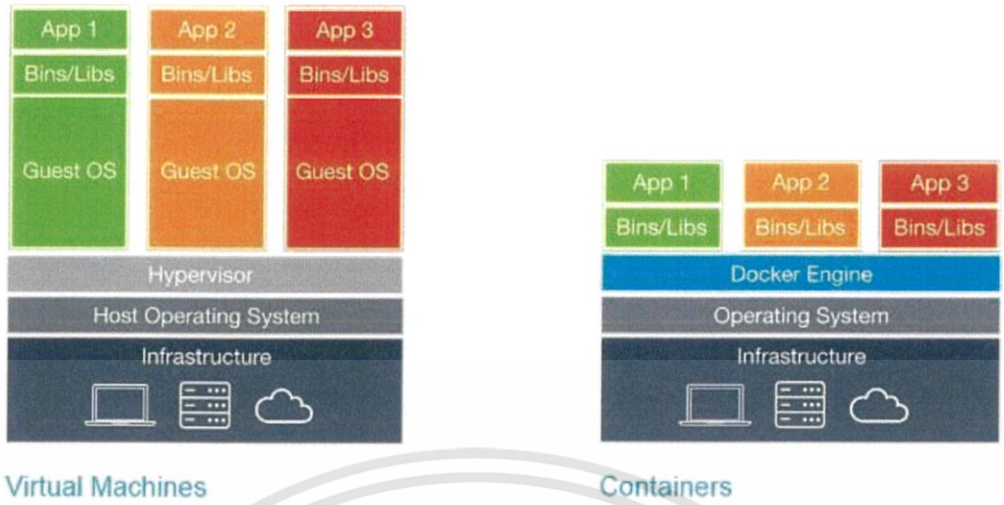
การใช้งาน AWS EC2 Spot Instance ผู้ใช้จะต้องสร้าง Spot request เพื่อแสดงความต้องการในการใช้งาน AWS EC2 Spot Instance โดยภายใน request จะมีราคาสูงสุดในการใช้เครื่อง EC2 ต่อชั่วโมงที่ผู้ใช้พึงใจที่จะจ่าย หากราคาที่ผู้ใช้ระบุภายใน request มีค่าสูงกว่าราคาต่ำสุดที่ AWS ระบุไว้ ผู้ใช้จะได้สิทธิ์ในการใช้งานไป แต่ถ้าราคา Spot price สูงกว่า ผู้ใช้จะไม่มีสิทธิ์ในการใช้เครื่อง รวมถึงเครื่องที่เคยมีสิทธิ์ในการใช้งานจะปิดตัวลง



ภาพที่ 2.4 แนวคิดของ AWS EC2 Spot Instance

ที่มา : AWS team. 2018. Spot Instance

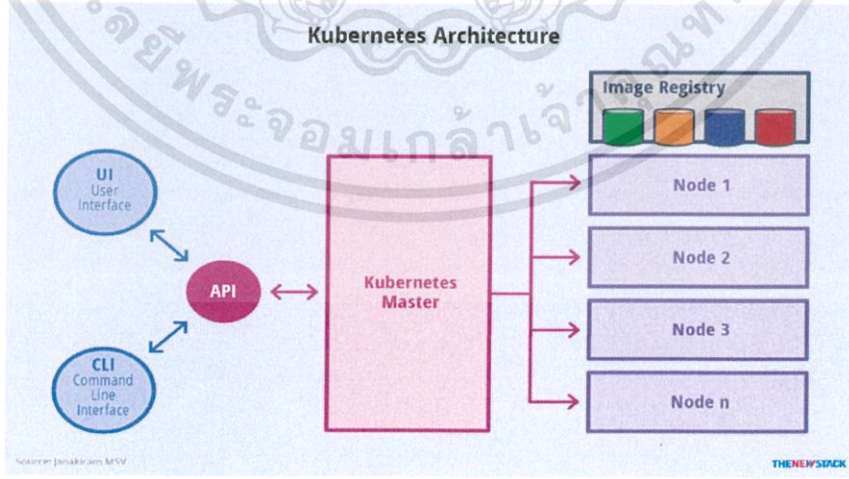
Docker เป็นโปรแกรมที่ช่วยจำลองสภาพแวดล้อมที่แอปพลิเคชันต้องการในการทำงานโดยไม่เกี่ยวข้องกับแอปพลิเคชันอื่น ๆ ที่รันอยู่บนระบบเดียวกัน (เรียกว่า container) ข้อดีของการใช้ Docker คือผู้ใช้สามารถนำแอปพลิเคชันของ ไปรันในต่างเครื่องได้ โดยที่ยังสามารถทำงานได้เหมือนเดิม ข้อแตกต่างระหว่างการใช้ container และ virtual machine คือ container จะจำลองเฉพาะไลบรารีและส่วนอื่น ๆเฉพาะที่จำเป็นในการทำงานของแอปพลิเคชันเท่านั้น ในขณะที่ virtual machine จะต้องจำลองถึงระดับระบบปฏิบัติการ ซึ่งเป็นการสิ้นเปลืองทรัพยากรของระบบโดยใช่เหตุ



ภาพที่ 2.5 แสดงความแตกต่างของการใช้ virtual machine กับ container

ที่มา : JerryGoyal. 2017. Docker, what is it and what is the purpose.

Kubernetes (k8s) เป็นโปรแกรมที่ช่วยจัดการการ container หรือที่เรียกว่า container orchestration ซึ่งจะช่วยจัดการการ deploy container เดียวกันลงบน computer cluster รวมถึงจัดการการเชื่อมต่อระหว่าง container, การสร้าง load balancer โดยผู้ใช้สามารถระบุ specs ที่ต้องการในรูปแบบ yaml เพื่อให้ Kubernetes นำไปสร้างระบบตามที่ผู้ใช้ต้องการ การกระทำลักษณะนี้เรียกว่า Infrastructure as a code นอกจากนี้ Kubernetes ยังมีความสามารถในการทำ self-healing ในกรณีที่แอปพลิเคชันเกิดการ crash ตัว Kubernetes จะทำการ restart หรือ deploy แอปพลิเคชันให้ใหม่



ภาพที่ 2.6 สถาปัตยกรรมของ Kubernetes cluster

ที่มา : Janakiram MSV. 2016. KUBERNETES: AN OVERVIEW.

2.2 งานวิจัยที่เกี่ยวข้อง

ในการพัฒนา Kubernetes CronJob นี้ เป็นการนำความสามารถของแอปพลิเคชัน Kops (Kubernetes Operations) ส่วนที่เกี่ยวข้องกับ AWS EC2 Spot Instance มาใช้

Kops เป็นแอปพลิเคชันที่ช่วยในการสร้าง, แก้ไข, ทำลาย, อัพเกรด และบำรุงรักษา Kubernetes cluster ในระดับ production-grade สำหรับการใช้งานบน cloud computing platform (AWS, GCE) โดยในขั้นตอนการสร้าง Kubernetes cluster นั้น หลังจากที่ผู้ใช้ระบุคุณสมบัติของ cluster ที่ต้องการแล้ว Kops จะสร้าง/ติดตั้งระบบพื้นฐานที่จำเป็นสำหรับการใช้งาน cluster เช่น AWS EC2 instance, VPC network, AWS S3 bucket สำหรับเก็บ cluster configurations เป็นต้น รวมถึงติดตั้ง Kubernetes ให้โดยอัตโนมัติ ซึ่งขั้นตอนที่ผู้ใช้ระบุคุณสมบัติเพื่อสร้าง cluster ผู้ใช้สามารถเลือกให้ node ของ cluster ใช้ AWS EC2 Spot Instance และกำหนดราคา Spot price ที่ต้องการได้ หรือหลังจากติดตั้งแล้ว ก็สามารถแก้ไข Spot price ได้เช่นกัน

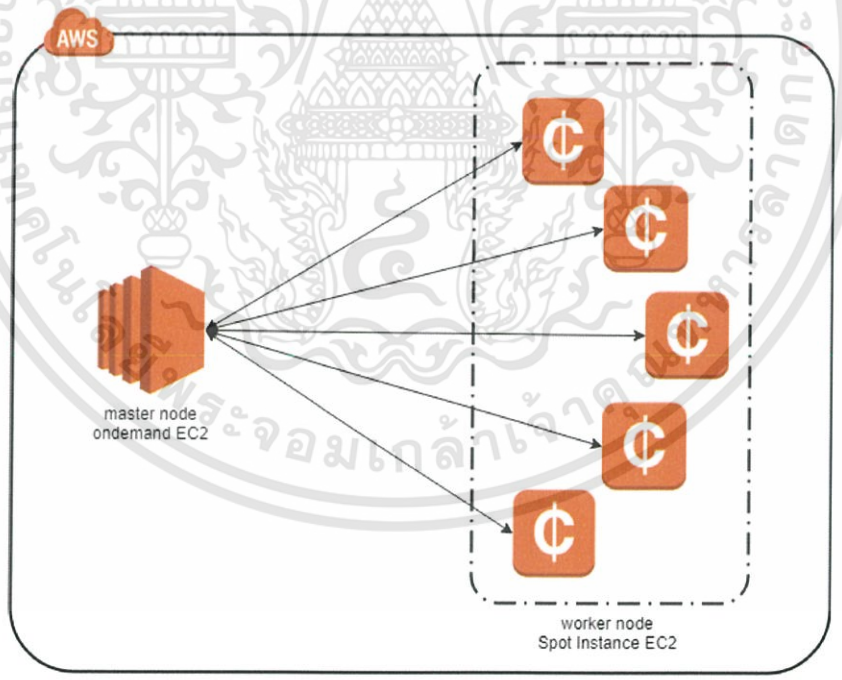
แต่อย่างไรก็ดี Kops ไม่มีความสามารถในการเปลี่ยนราคา Spot price สำหรับ node ใน cluster ได้โดยอัตโนมัติ จึงเป็นหน้าที่ของผู้ใช้เองที่จะต้องติดตามราคา Spot price และเปลี่ยนแปลงการตั้งค่า Spot price ของ cluster เพื่อให้ AWS EC2 Spot instance ใช้งานได้อย่างต่อเนื่อง

บทที่ 3
วิธีดำเนินการวิจัย

3.1 สถาปัตยกรรมของระบบ

การทำงานของ Kubernetes CronJob สำหรับเปลี่ยนราคาการประมูลการใช้งาน AWS Spot Instance เป็นโปรแกรมที่ทำงานตามกำหนดการที่ผู้ใช้กำหนดไว้ (cron job) ซึ่งทำงานอยู่บน Kubernetes cluster บน cloud infrastructure ของ Amazon Web Service (AWS)

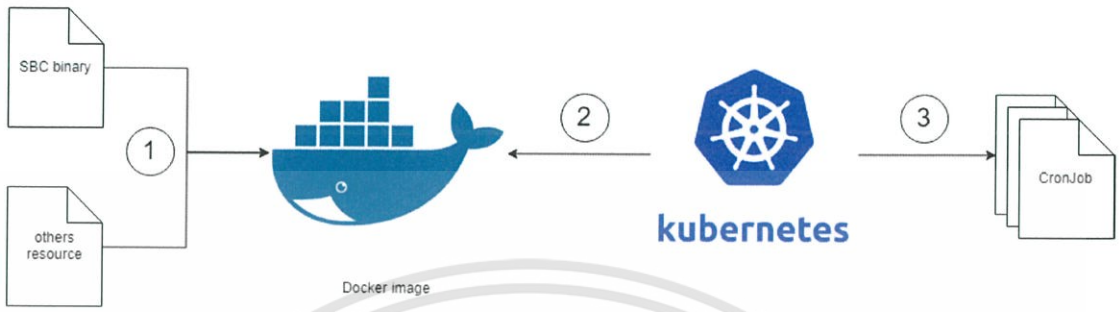
Kubernetes cluster ประกอบด้วย node 2 ชนิด คือ master node และ worker node ซึ่ง master node ทำหน้าที่ควบคุมการทำงานของ container ที่ทำงานอยู่บน worker node รวมถึงจัดการข้อมูลด้านอื่น ๆ ที่เกี่ยวข้องกับ cluster อาทิ internal DNS, node metrics, role-based access control, ฯลฯ. Node ทั้งสองชนิดทำงานอยู่บน AWS Elastic Compute Cloud (EC2) โดยที่ master node ทำงานอยู่บน EC2 instance ชนิด on demand แต่ worker node ทำงานอยู่บน EC2 instance ชนิด Spot Instance เพื่อลดค่าใช้จ่าย



ภาพที่ 3.1 ภาพรวมของ Kubernetes cluster บน AWS

โดย Kubernetes CronJob นี้ถูกพัฒนาขึ้นด้วยภาษา Golang เนื่องจากสามารถทำงานกับ API ของ AWS ได้ อีกทั้งยังมีสมรรถนะในการทำงานสูงใกล้เคียงกับภาษาซี หลังจากการพัฒนาเสร็จสิ้นแล้ว CronJob จะถูกคอมไพล์เป็นไบนารีไฟล์และนำไปสร้างเป็น Docker container image เพื่อ

เป็นแม่แบบในการสร้าง cron job container บน Kubernetes cluster ตามกำหนดเวลา โดย cron job container ดังกล่าวจะทำงานบน master node



ภาพที่ 3.2 กระบวนการสร้าง Docker image และการนำไปใช้

3.2 การออกแบบ

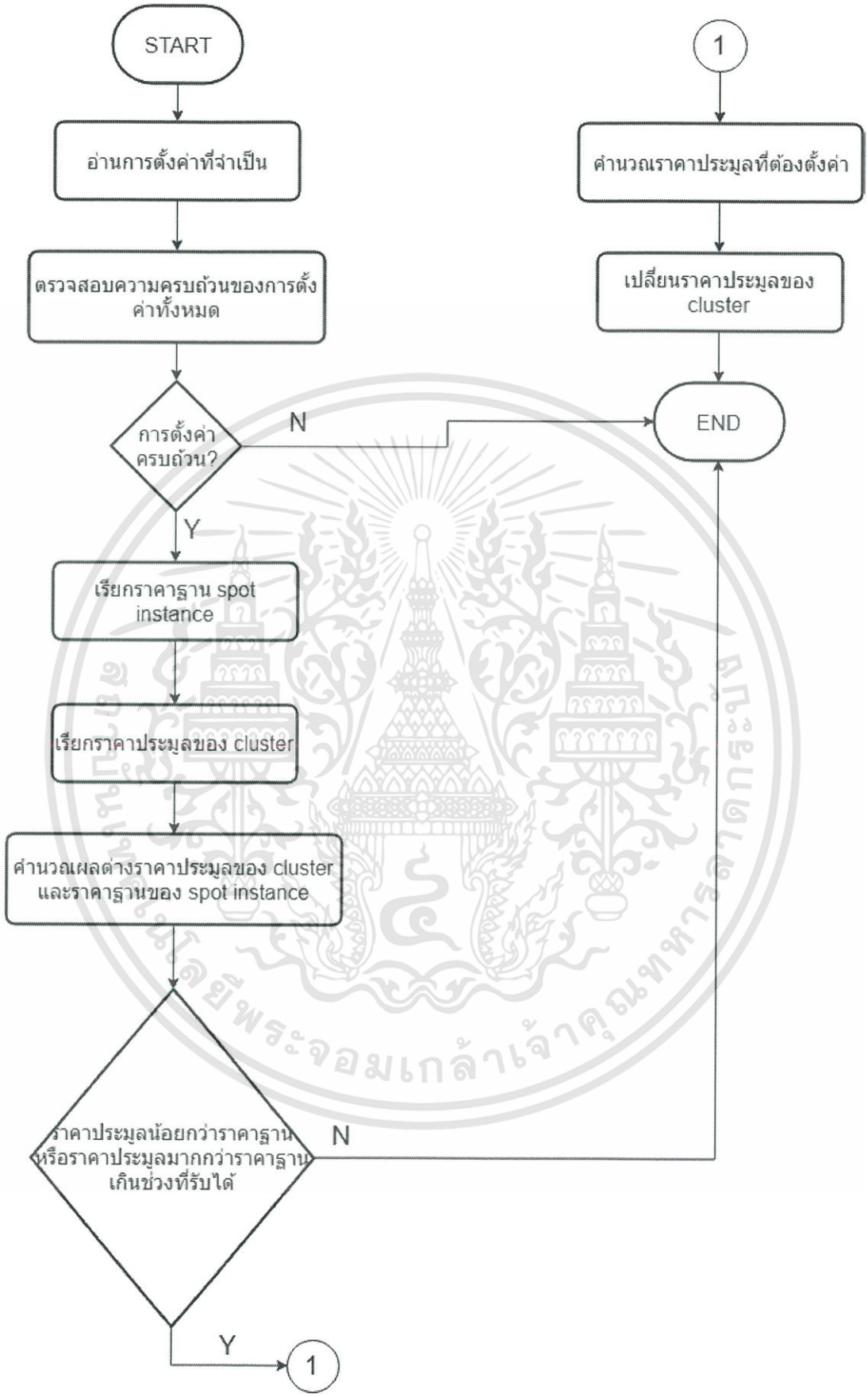
3.2.1 Use case diagram

เนื่องจาก Kubernetes CronJob ถูกเรียกขึ้นมาทำงานโดยอัตโนมัติตามกำหนดการณ์ของ cron job ดังนั้น ผู้ใช้จึงมีหน้าที่เพียงแค่ตั้งค่าการใช้งานตามที่ Kubernetes CronJob ต้องการ รวมถึงตั้งค่าส่วนต่างของราคาประมูลกับราคาฐานของ Spot Instance (threshold) ให้อยู่ในช่วงที่ผู้ใช้ยอมรับได้ และเปลี่ยนกำหนดการในการเรียก Kubernetes CronJob ขึ้นมาทำงานเท่านั้น

ภาพที่ 3.3 Use case diagram ของผู้ใช้ที่มีต่อ Kubernetes CronJob

3.2.2 Flowchart ขั้นตอนการทำงานของโปรแกรม

ในการทำงานของ Kubernetes CronJob จะประกอบด้วยขั้นตอนในการทำงานตามแผนภาพด้านล่าง



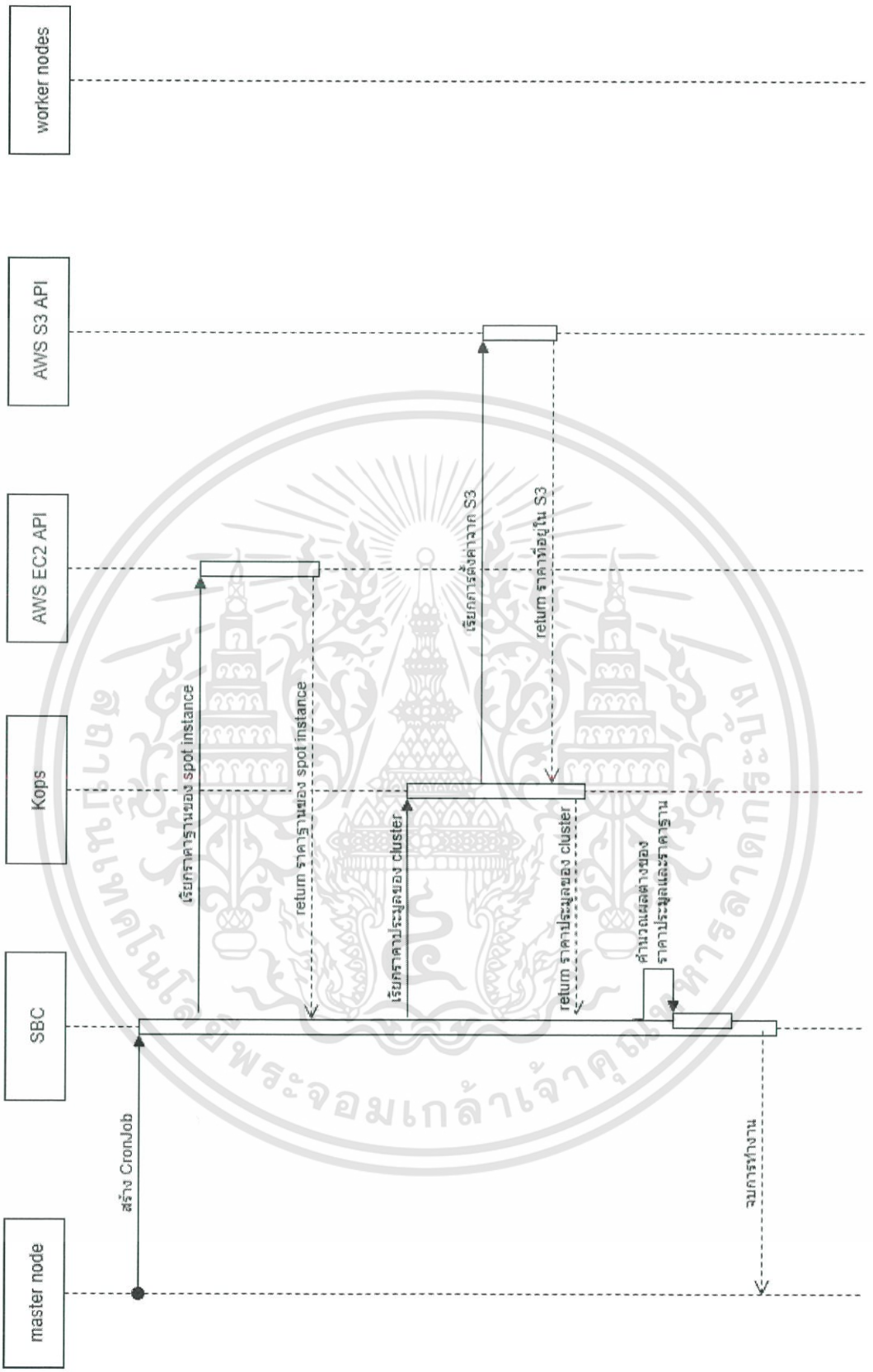
ภาพที่ 3.4 แผนภาพขั้นตอนการทำงานของ Kubernetes CronJob

3.2.3 Sequential diagram

เนื่องจาก Kubernetes cluster ที่ Kubernetes CronJob ทำงานอยู่ ถูกสร้างขึ้นจากโปรแกรมอื่นอีกหนึ่งโปรแกรม คือ Kops (Kubernetes Operation) จึงมีความจำเป็นที่ต้องทำการแก้ไข cluster ผ่านแอปพลิเคชัน Kops

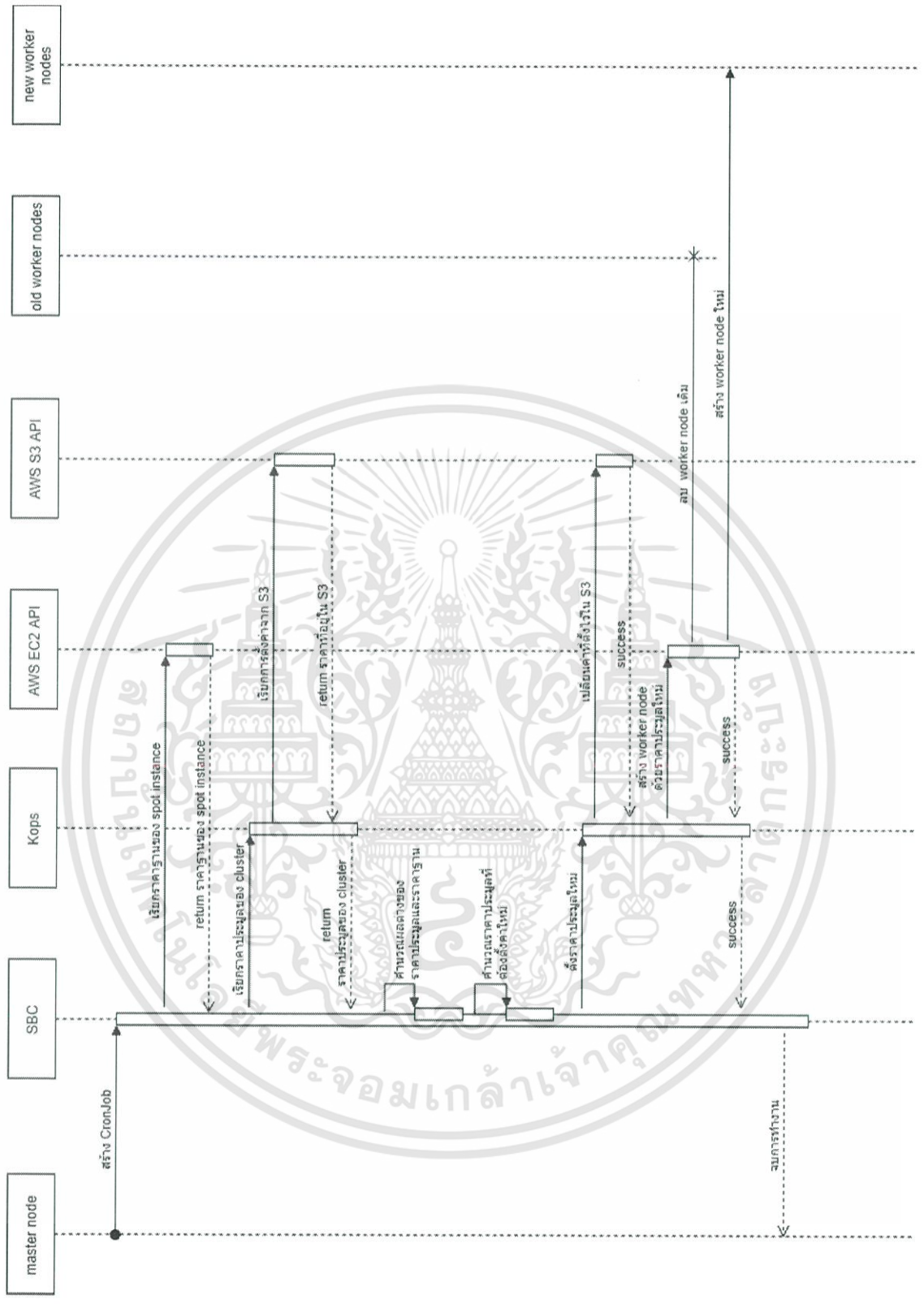
ขั้นตอนในการทำงานของ Kubernetes CronJob ร่วมกับ Kops และ AWS ถูกแบ่งออกได้เป็นสามกรณี ดังนี้

3.2.3.1 กรณีราคา Spot price ของ cluster ที่ถูกตั้งค่าไว้ มีค่ามากกว่าราคาฐานของ spot instance เป็นมูลค่าน้อยกว่าหรือเท่ากับช่วงของผลต่างที่ผู้ใช้ยอมรับได้ (Spot price ของ cluster - ราคาฐาน <= threshold) กรณีนี้ Kubernetes CronJob จะไม่ดำเนินการเปลี่ยนแปลงการตั้งค่าราคา Spot price ของ cluster ของ worker node เนื่องจากยังอยู่ในช่วงผลต่างที่ผู้ใช้ยอมรับได้อยู่



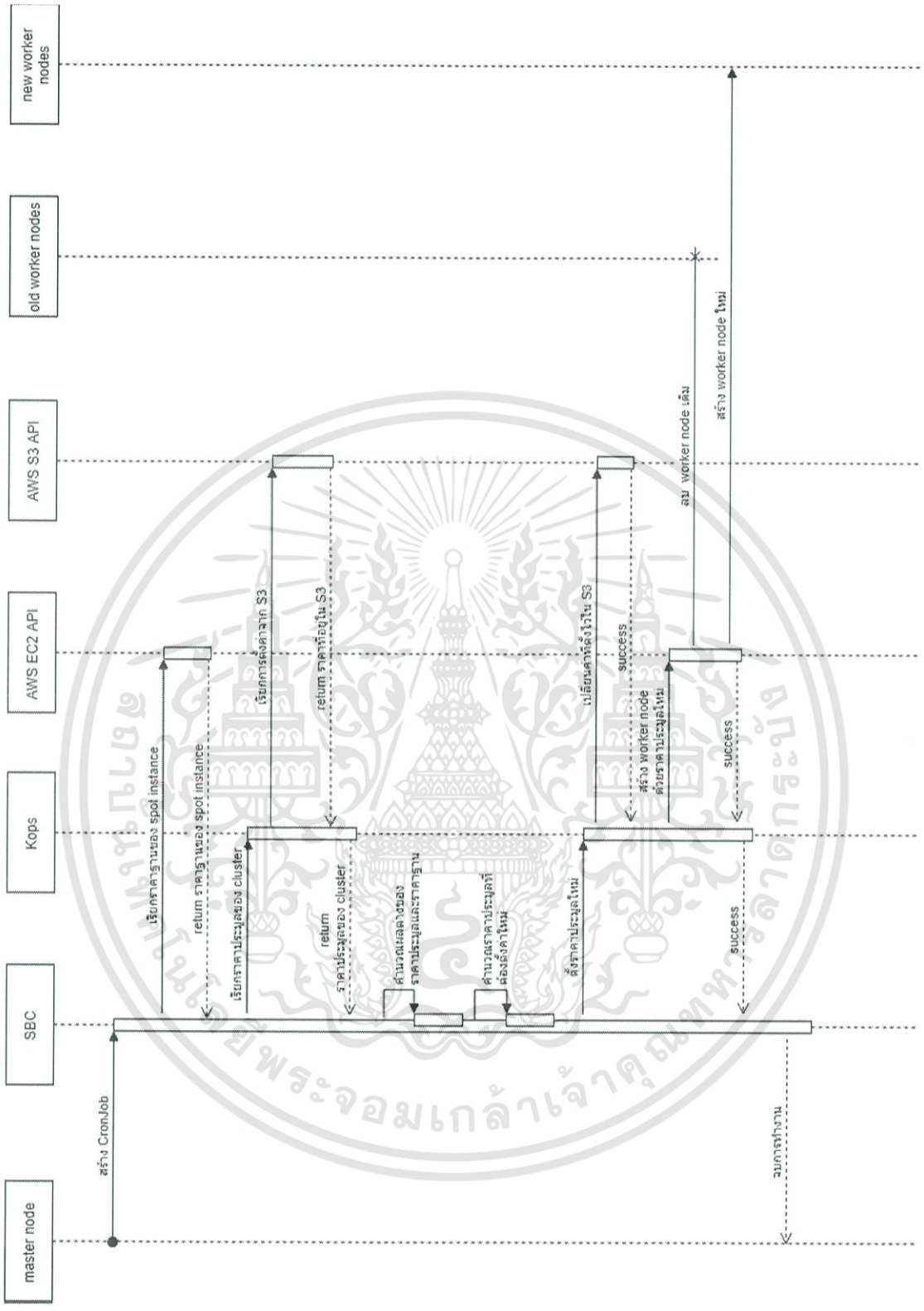
ภาพที่ 3.5 แผนภาพลำดับการทำงานร่วมกับระบบอื่นของสถานการณ์ราคา Spot price ของ cluster มีค่ามากกว่าราคาฐานของ spot instance เป็นมูลค่าน้อยกว่าหรือเท่ากับช่วงของผลต่างที่ผู้ใช้ยอมรับได้

3.2.3.2 กรณีราคา Spot price ของ cluster ที่ถูกตั้งค่าไว้ มีค่ามากกว่าราคาฐาน ของ spot instance เป็นมูลค่ามากกว่าช่วงของผลต่างที่ผู้ใช้ยอมรับได้ (Spot price ของ cluster - ราคาฐาน > threshold) ในกรณีนี้ ถึงแม้จะสามารถใช้งาน spot instance ได้ แต่ราคาของ spot instance ที่ถูกตั้งค่าไว้กลับสูงกว่าราคาฐานเกินช่วงที่ผู้ใช้ยอมรับ อันเป็นการสิ้นเปลืองงบประมาณ โดยใช่เหตุ Kubernetes CronJob จะทำการปรับราคา Spot price ของ cluster ให้เป็นราคาฐาน รวมกับช่วงของผลต่างที่ผู้ใช้ยอมรับได้ (Spot price ของ cluster = ราคาฐาน + threshold)



ภาพที่ 3.6 แผนภาพลำดับการทำงานร่วมกับระบบอื่นของสถานการณ์ราคา Spot price ของ cluster มีค่ามากกว่าราคาฐานของ spot instance เป็นมูลค่ามากกว่าช่วงของผลต่างที่ผู้ใช้ยอมรับได้

3.2.3.3 กรณีราคา Spot price ของ cluster ที่ถูกตั้งค่าไว้ มีค่าน้อยกว่าราคาฐานของ spot instance (Spot price ของ cluster < ราคาฐาน) ในกรณีนี้จะไม่สามารถใช้งาน spot instance ได้ เนื่องจากผู้ใช้ประมูล spot instance ด้วยราคาที่ต่ำกว่าราคาฐาน Kubernetes CronJob จะทำการปรับราคา Spot price ของ cluster ให้เป็นราคาฐานรวมกับช่วงของผลต่างที่ผู้ใช้ยอมรับได้ (Spot price ของ cluster = ราคาฐาน + threshold)



ภาพที่ 3.7 แผนภาพลำดับการทำงานร่วมกับระบบอื่นของสถานการณ์ราคา Spot price ของ cluster มีค่าน้อยกว่าราคาฐานของ spot instance

3.3 การตั้งค่าอื่น ๆ ที่ต้องใช้ในการทำงานของโปรแกรม

ด้านล่างเป็นตารางรวมการตั้งค่าที่ต้องใช้ในการทำงานของ SBC ซึ่งจะถูกตั้งค่าให้เรียกใช้จาก environment variable ด้วยเหตุผลด้านความปลอดภัย โดยผู้ใช้สามารถกำหนดค่า environment variable ดังกล่าวผ่านทาง resource ของ Kubernetes สองชนิด ได้แก่ Configmap ซึ่งเก็บข้อมูลในรูปแบบ key-value สำหรับเก็บค่าทั่วไป และ Secret ซึ่งเป็น Configmap ที่ถูก encode ด้วย base64 ทำให้ยากต่อการอ่าน สำหรับเก็บข้อมูลที่มีความอ่อนไหวเช่น access key ID และ access key secret

ตารางที่ 3.1 แสดงชื่อของ environment variable ที่ต้องใช้ในการทำงาน และคำอธิบาย environment variable

ชื่อของ environment variable	คำอธิบาย
SBC_INSTANCE_GROUP_NAME	กลุ่มของ worker nodes ที่ต้องการกำหนดให้ SBC ทำงาน สามารถดูได้จากการใช้ Kops
SBC_INSTANCE_TYPE	ชนิดของ EC2 instance ที่ใช้รัน worker nodes เช่น t3.medium
SBC_PRODUCT_DESCRIPTION	Product family ของ worker nodes โดยปกติจะเป็น Linux/UNIX
SBC_AVAILABILITY_ZONE	Availability zone ที่ worker nodes ทำงานอยู่
SBC_PRICE_THRESHOLD	ส่วนต่างของราคาประมูลกับราคาฐานของ Spot Instance ซึ่งอยู่ในช่วงที่ผู้ใช้ยอมรับได้
AWS_REGION	AWS region ที่ worker nodes ทำงานอยู่
AWS_ZONE	Availability zone ที่ worker nodes ทำงานอยู่
AWS_ACCESS_KEY_ID	Access key ID ของ AWS ของ user ที่มีสิทธิ์ในการทำงานเพียงพอสำหรับ Kops
AWS_SECRET_ACCESS_KEY	Access key secret ของ AWS ของ user ที่มี access key ID ข้างต้น
KOPS_STATE_STORE	ที่อยู่ของ AWS S3 bucket ซึ่งเก็บ configuration ของ cluster
S3_BUCKET_NAME	ชื่อของ AWS S3 bucket ซึ่งเก็บ configuration ของ cluster
KOPS_CLUSTER_NAME	ชื่อของ cluster ที่ SBC จะถูกนำไปรัน

3.4 ข้อคำนึงอื่น ๆ ในการออกแบบ

การทำงานของ Kubernetes CronJob มีความจำเป็นอย่างยิ่งที่จะต้องให้ Kubernetes CronJob ทำงานบน master node เนื่องจากขั้นตอนการเปลี่ยนแปลงราคา Spot price ของ cluster จะต้องมีการ terminate node ทิ้ง และสร้างขึ้นใหม่ หากให้ Kubernetes CronJob ทำงาน

บน worker node จะทำให้การเปลี่ยนราคาประมูลไม่สำเร็จ เนื่องจาก Kubernetes CronJob ถูกขัดการทำงานเพราะ node ที่รัน Kubernetes CronJob ถูก terminate ทิ้ง

3.5 สภาพแวดล้อมที่ใช้ทดสอบการทำงาน

เพื่อให้ Kubernetes CronJob สามารถทำงานได้ตรงตามวัตถุประสงค์ จึงต้องมีการทดสอบการทำงานกับ Kubernetes cluster โดยมีสภาพแวดล้อมในการทดสอบการทำงานดังนี้

ตารางที่ 3.2 ตารางแสดงสภาพแวดล้อมที่ใช้ทดสอบการทำงานของ Kubernetes CronJob

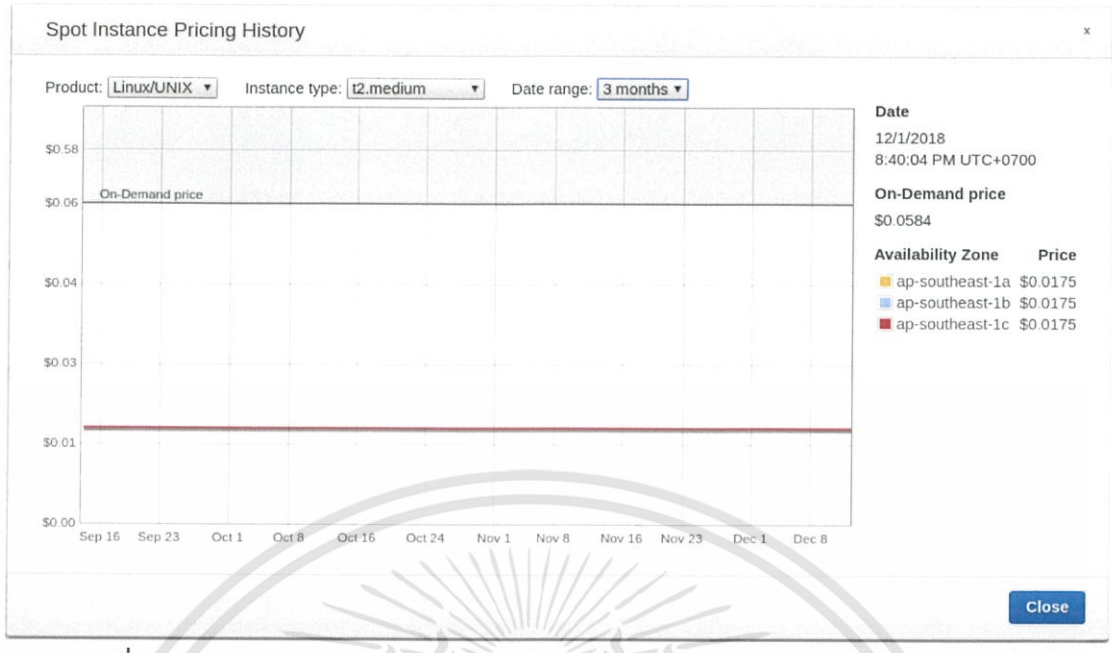
จำนวน node ที่ใช้ AWS EC2 Spot Instance	8 nodes
ชนิดของ AWS EC2 Spot Instance	t2.medium
Kubernetes cluster version	1.10.6
CronJob interval	5 นาที
Spot price ณ เวลาที่ทดสอบ	0.01750 USD/ชั่วโมง
Spot price threshold	0.00002 USD

3.6 Scenario ที่ใช้ทดสอบการทำงาน

Scenario ที่ใช้ทดสอบการทำงาน จะสอดคล้องกับเงื่อนไขในการทำงานของแอปพลิเคชันข้างต้นจำนวน 3 scenarios ได้แก่

1. Spot price ของ cluster - ราคาฐาน <= threshold
2. Spot price ของ cluster - ราคาฐาน > threshold
3. Spot price ของ cluster < ราคาฐาน

เนื่องจากชนิดของ AWS EC2 Spot Instance ที่ใช้ทดสอบการทำงาน (t2.medium) เป็นชนิดที่ราคาไม่ผันผวนมาก ในการทดสอบ scenario ที่ 2 และ 3 จึงมีความจำเป็นจะต้องเปลี่ยนแปลงราคา Spot price ของ cluster เองก่อน โดยสำหรับ scenario ที่ 2 จะตั้งราคา Spot price ไว้ที่ 0.01758 USD/ชั่วโมง และสำหรับ scenario ที่ 3 จะตั้งราคา Spot price ไว้ที่ 0.017 USD/ชั่วโมง



ภาพที่ 3.8 แสดงราคา Spot price สำหรับ AWS EC2 Spot Instance ย้อนหลัง 3 เดือน

บทที่ 4
ผลการวิจัย

4.1 ผลการทดลองการทำงานกับ scenario ต่าง ๆ

4.1.1 ผลการทดลองกับ scenario Spot price ของ cluster - ราคาฐาน <= threshold

Scenario นี้เป็นเหตุการณ์ที่เกิดขึ้นเป็นปกติของ cluster หากราคา Spot price ของ AWS ไม่มีการเปลี่ยนแปลง โดยภายใน Kubernetes cluster จะทำการเก็บผลการรัน CronJob ย้อนหลัง 30 ครั้ง ได้ผลและ log การทำงานดังนี้

NAME	READY	STATUS	RESTARTS	AGE
spot-bid-changer-1544603100-lvtmn	0/1	Completed	0	2h
spot-bid-changer-1544603400-vpf9w	0/1	Completed	0	2h
spot-bid-changer-1544603700-swjvf	0/1	Completed	0	2h
spot-bid-changer-1544604000-8knth	0/1	Completed	0	2h
spot-bid-changer-1544604300-k8g56	0/1	Completed	0	2h
spot-bid-changer-1544604600-m2s6v	0/1	Completed	0	2h
spot-bid-changer-1544604900-2wxpl	0/1	Completed	0	1h
spot-bid-changer-1544605200-lj77v	0/1	Completed	0	1h
spot-bid-changer-1544605500-l4jwj	0/1	Completed	0	1h
spot-bid-changer-1544605800-hqhsq	0/1	Completed	0	1h
spot-bid-changer-1544606100-s6ths	0/1	Completed	0	1h
spot-bid-changer-1544606400-4lxw2	0/1	Completed	0	1h
spot-bid-changer-1544606700-42q6b	0/1	Completed	0	1h
spot-bid-changer-1544607000-k2zjw	0/1	Completed	0	1h
spot-bid-changer-1544607300-zdbzp	0/1	Completed	0	1h
spot-bid-changer-1544607600-d4gzp	0/1	Completed	0	1h
spot-bid-changer-1544607900-wgdgf	0/1	Completed	0	1h
spot-bid-changer-1544608200-mrxlj	0/1	Completed	0	1h
spot-bid-changer-1544608500-tt6cw	0/1	Completed	0	55m
spot-bid-changer-1544608800-7rvxp	0/1	Completed	0	50m
spot-bid-changer-1544609100-5h9nx	0/1	Completed	0	46m
spot-bid-changer-1544609400-9j4bl	0/1	Completed	0	41m
spot-bid-changer-1544609700-58nl2	0/1	Completed	0	36m
spot-bid-changer-1544610000-8fh8p	0/1	Completed	0	31m
spot-bid-changer-1544610300-xmxrs	0/1	Completed	0	26m
spot-bid-changer-1544610600-xf2fk	0/1	Completed	0	21m
spot-bid-changer-1544610900-dgf54	0/1	Completed	0	16m
spot-bid-changer-1544611200-f8dnz	0/1	Completed	0	11m
spot-bid-changer-1544611500-8f9kx	0/1	Completed	0	6m
spot-bid-changer-1544611800-2c8x4	0/1	Completed	0	1m

ภาพที่ 4.1 แสดงผลการทำงานของ Kubernetes CronJob ย้อนหลัง 30 ครั้ง

```
+ creating session
+ success
+ creating new EC2 session
+ latest spot instance price: 0.017500
+ current spot price for this instance group: 0.017520
+ no need to update spot price
```

ภาพที่ 4.2 แสดง log การทำงานของ Kubernetes CronJob แต่ละครั้งใน scenario แรก

4.1.2 Spot price ของ cluster - ราคาฐาน > threshold

ใน scenario นี้ได้ทำการทดลองโดยเปลี่ยนราคา Spot price ของ cluster เป็น 0.01758 USD/ชั่วโมง โดย CronJob สามารถทำงานได้อย่างราบรื่น ใช้เวลาในการทำงานรวม 48 นาที 6 วินาที (log ของการทำงานสามารถพิจารณารายละเอียดที่ภาคผนวก ง)

4.1.3 Spot price < ราคาฐาน

ใน scenario นี้ได้ทำการทดลองโดยเปลี่ยนราคา Spot price ของ cluster เป็น 0.0170 USD/ชั่วโมง โดย CronJob สามารถทำงานได้อย่างราบรื่น ใช้เวลาในการทำงานรวม 48 นาที 16 วินาที (log ของการทำงานสามารถพิจารณารายละเอียดที่ภาคผนวก จ)

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลจากการทดลอง

จากการทดลองใช้งาน ทั้งการใช้งานจริง และการสร้างสถานการณ์เพื่อให้ตรงกับเงื่อนไขการทำงานของ Kubernetes CronJob พบว่า Kubernetes CronJob สามารถทำงานได้ผลลัพธ์ตรงตามวัตถุประสงค์ที่วางไว้ คือสามารถปรับราคา Spot price ของ cluster ให้อยู่ในช่วงที่ผู้ใช้พอใจ และผ่านราคา Spot price ขั้นต่ำของ AWS ได้

5.2 ข้อดี-ข้อเสียของงานวิจัย

ประโยชน์โดยตรงของ Kubernetes CronJob นี้คือช่วยลดภาระในการติดตามราคา Spot price จาก AWS และช่วยลดความยุ่งยากในการแก้ไขราคา Spot price ของ cluster ด้วยตนเอง ซึ่งเกิดความผิดพลาดในบางครั้งและก่อให้เกิดความเสียหายต่อ cluster Kubernetes cluster และแอปพลิเคชันที่กำลังรันอยู่บน cluster

ข้อเสียของ Kubernetes CronJob นี้คือ ยังต้องพึ่งพาไบนารีของ Kops ที่ถูกคอมไพล์แล้ว ทำให้ขนาดของ Docker image มีขนาดใหญ่กว่าที่ควรจะเป็น อีกทั้งยังต้องตามอัพเดต Docker image ทุกครั้งที่ Kops ออก release ใหม่

5.3 ข้อเสนอแนะสำหรับการต่อยอด

เนื่องจาก Kubernetes CronJob นี้ ถูกออกแบบมาเพื่อทำงานบน Kubernetes cluster เท่านั้น หากต้องการนำไปใช้งานกับ AWS EC2 Spot Instance ที่ไม่ได้ใช้งานเป็น Kubernetes cluster ก็สามารถทำได้ โดยแก้ไขให้ใช้เพียงแค่ AWS API อย่างเดียว และต้องนำไปรันบนเครื่องที่ไม่ใช่ AWS EC2 Spot Instance

นอกจากนี้สามารถเพิ่มความสามารถในการ auto update Docker image ได้ โดยสร้าง CI/CD flow สำหรับติดตาม release ของ Kops และนำไปสร้าง Docker image ใหม่ให้โดยอัตโนมัติ

รวมถึงการลดขนาดของ Docker image สามารถทำได้โดยเลือกใช้ source code เพียงบางส่วนของ Kops ที่เกี่ยวข้องกับขั้นตอนการเปลี่ยนราคา Spot price แล้วนำมาคอมไพล์เป็นไบนารีเดียวรวมกับ CronJob นี้เลย

เอกสารอ้างอิง

it24hrs. 2015. **Cloud Computing คืออะไร ? Cloud Computing ดีอย่างไร ?.** [online]
Available : https://www.it24hrs.com/2015/cloud-computing-and-cloud-definition/

AWS team. 2018. **What is Cloud Computing?.** [online]
Available : https://aws.amazon.com/what-is-cloud-computing

AWS team. 2018. **Spot Instances.** [online]
Available : https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

AWS team. 2018. **Amazon EC2 Spot Instances Pricing.** [online]
Available : https://aws.amazon.com/ec2/spot/pricing/

AWS team. 2018. **AWS SDK for Go.** [online]
Available : https://docs.aws.amazon.com/sdk-for-go/api/

Docker team. 2018. **What is a Container.** [online]
Available : https://www.docker.com/resources/what-container

Docker team. 2018. **Dockerfile reference.** [online]
Available : https://docs.docker.com/engine/reference/builder/

Golang team. 2018. **Documentation.** [online]
Available : https://golang.org/doc/

Janakiram MSV. 2016. **KUBERNETES: AN OVERVIEW.** [online]
Available : https://thenewstack.io/kubernetes-an-overview/

JerryGoyal. 2017. **Docker, what is it and what is the purpose.** [online]
Available : https://stackoverflow.com/questions/28089344/docker-what-is-it-and-what-is-the-purpose

Kubernetes team. 2018. **Concepts.** [online]
Available : https://kubernetes.io/docs/concepts/

Kubernetes team. 2018. **CronJob v1beta1 batch.** [online]
Available : https://v1-10.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#cronjob-v1beta1-batch

Westcon Comstor. 2018. **Amazon Web Service.** [online]
Available : https://www.westconcomstor.com/au/en/vendors/wc-vendors/amazon-web-services.html

ภาคผนวก

ภาคผนวก ก

Source code ของ CronJob

```
package main

import (
    "bytes"
    "errors"
    "fmt"
    "io"
    "io/ioutil"
    "os"
    "os/exec"
    "strconv"
    "strings"
    "time"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
    "gopkg.in/yaml.v2"
)

type Config struct {
    Spec struct {
        MaxPrice string `yaml:"maxPrice"`
    } `yaml:"spec"`
}

const KopsBin = "/kops"

func execTail(bin *exec.Cmd) {
    var stdoutBuf, stderrBuf bytes.Buffer
    stdoutIn, _ := bin.StdoutPipe()
    stderrIn, _ := bin.StderrPipe()

    var errStdout, errStderr error
    stdout := io.MultiWriter(os.Stdout, &stdoutBuf)
    stderr := io.MultiWriter(os.Stderr, &stderrBuf)

    err := bin.Start()
    if err != nil {
        panic(err)
```

```
    }

    go func() {
        _, errStdout = io.Copy(stdout, stdoutIn)
    }()

    go func() {
        _, errStderr = io.Copy(stderr, stderrIn)
    }()

    err = bin.Wait()
    if err != nil {
        panic(err)
    }

    if errStdout != nil || errStderr != nil {
        fmt.Println("failed to capture")
    }

    fmt.Println()

}

func getSpotPrice(sess *session.Session, instanceTypes string,
productDesc string, az string) float64 {
    tme := time.Now()

    fmt.Println(" + creating new EC2 session")
    svc := ec2.New(sess)

    input := &ec2.DescribeSpotPriceHistoryInput{
        InstanceTypes: []*string{
            aws.String(instanceTypes),
        },
        ProductDescriptions: []*string{
            aws.String(productDesc),
        },
        AvailabilityZone: aws.String(az),
        StartTime:        &tme,
    }

    result, err := svc.DescribeSpotPriceHistory(input)
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            default:
                panic(aerr.Error())
            }
```

```
        } else {
            panic(err)
        }
    }

    latestPrice, err :=
strconv.ParseFloat(*result.SpotPriceHistory[0].SpotPrice, 64)
    if err != nil {
        panic(err)
    }

    return latestPrice
}

func getCurrentConfig(igName string, fileName string) float64 {
    f, err := os.Create(fileName)
    if err != nil {
        panic(err)
    }

    kopsGetIg := exec.Command(KopsBin, "get", "ig", igName, "-o",
"yaml")

    defer f.Close()

    kopsGetIg.Stdout = f
    err = kopsGetIg.Run()
    if err != nil {
        fmt.Println(err)
        panic(err)
    }

    var specConfig Config
    source, err := ioutil.ReadFile(fileName)
    if err != nil {
        panic(err)
    }
    err = yaml.Unmarshal(source, &specConfig)
    if err != nil {
        panic(err)
    }

    fmt.Println(" + current spot price for this instance group: " +
specConfig.Spec.MaxPrice)
    currentPrice, err := strconv.ParseFloat(specConfig.Spec.MaxPrice,
64)
    if err != nil {
        panic(err)
```

```
    }

    return currentPrice
}

func updateMaxPrice(newMaxPrice float64, fileName string) {
    fileRead, err := ioutil.ReadFile(fileName)
    if err != nil {
        panic(err)
    }

    lines := strings.Split(string(fileRead), "\n")

    for i, line := range lines {
        if strings.Contains(line, "maxPrice") {
            lines[i] = "  maxPrice: \"" + fmt.Sprintf("%.6f",
newMaxPrice) + "\""
        }
    }
    output := strings.Join(lines, "\n")
    err = ioutil.WriteFile(fileName, []byte(output), 0644)
    if err != nil {
        panic(err)
    }
}

func main() {

    igName := os.Getenv("SBC_INSTANCE_GROUP_NAME")
    if igName == "" {
        panic(errors.New("No instance group name specified"))
    }

    instanceType := os.Getenv("SBC_INSTANCE_TYPE")
    if instanceType == "" {
        panic(errors.New("No instance type specified"))
    }

    productDesc := os.Getenv("SBC_PRODUCT_DESCRIPTION")
    if productDesc == "" {
        panic(errors.New("No product description specified"))
    }

    az := os.Getenv("SBC_AVAILABILITY_ZONE")
    if az == "" {
        panic(errors.New("No availability zone specified"))
    }
```

```
    priceThresholdStr := os.Getenv("SBC_PRICE_THRESHOLD")
    if priceThresholdStr == "" {
        panic(errors.New("No price threshold specified"))
    }

    priceThreshold, err := strconv.ParseFloat(priceThresholdStr, 64)
    if err != nil {
        panic(err)
    }

    envChecklist := []string{
        "AWS_ACCESS_KEY_ID",
        "AWS_REGION",
        "AWS_SECRET_ACCESS_KEY",
        "AWS_ZONES",
        "KOPS_CLUSTER_NAME",
        "KOPS_STATE_STORE",
        "S3_BUCKET_NAME",
    }

    for _, env := range envChecklist {
        envString, isSet := os.LookupEnv(env)
        if !isSet || envString == "" {
            panic(errors.New("No environment variable " + env + "
specified"))
        }
    }

    fmt.Println(" + creating session")

    sess, err := session.NewSession()
    if err != nil {
        panic(err)
    }
    fmt.Println(" + success")

    latestPrice := getSpotPrice(sess, instanceType, productDesc, az)
    fmt.Println(" + latest spot instance price: " + fmt.Sprintf("%f",
latestPrice))

    fileName := "ig-config"

    currentPrice := getCurrentConfig(igName, fileName)

    var newPrice float64

    if (latestPrice > currentPrice) || ((currentPrice - latestPrice)
> priceThreshold) {
```

```
        newPrice = latestPrice + priceThreshold
    } else {
        fmt.Println(" + no need to update spot price")
        return
    }

    fmt.Println(" + changing spot price to " + fmt.Sprintf("%f",
newPrice))
    updateMaxPrice(newPrice, fileName)

    kopsReplace := exec.Command(KopsBin, "replace", "-f", fileName)
    execTail(kopsReplace)

    kopsUpdate := exec.Command(KopsBin, "update", "cluster", "--yes")
    execTail(kopsUpdate)

    kopsRoll := exec.Command(KopsBin, "rolling-update", "cluster", "-
-instance-group", igName, "--yes")
    execTail(kopsRoll)

    return
}
```

ภาคผนวก ข

Source code ของ CronJob Dockerfile

```
FROM golang:1.10.4-alpine3.8
ADD spot-bid-changer /
ADD kops /
CMD ["/spot-bid-changer"]
```

ภาคผนวก ค

Source code ของ Kubernetes CronJob manifest file

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: spot-bid-changer
  namespace: cronjob
spec:
  schedule: "*/5 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  concurrencyPolicy: "Forbid"
  startingDeadlineSeconds: 5400
  jobTemplate:
    spec:
      template:
        spec:
          imagePullSecrets:
            - name: dockersecret
          containers:
            - name: spot-bid-changer
              image: [OMITTED]/[OMITTED]-base:spot-bid-changer
              imagePullPolicy: Always
              env:
                - name: POD_NAME
                  valueFrom:
                    fieldRef:
                      fieldPath: metadata.name
                - name: POD_NAMESPACE
                  valueFrom:
                    fieldRef:
                      fieldPath: metadata.namespace
                - name: SBC_INSTANCE_GROUP_NAME
                  valueFrom:
                    configMapKeyRef:
                      name: sbc-configmap
                      key: SBC_INSTANCE_GROUP_NAME
                - name: SBC_INSTANCE_TYPE
                  valueFrom:
                    configMapKeyRef:
                      name: sbc-configmap
                      key: SBC_INSTANCE_TYPE
                - name: SBC_PRODUCT_DESCRIPTION
                  valueFrom:
                    configMapKeyRef:
```

```
            name: sbc-configmap
            key: SBC_PRODUCT_DESCRIPTION
        - name: SBC_AVAILABILITY_ZONE
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: SBC_AVAILABILITY_ZONE
        - name: SBC_PRICE_THRESHOLD
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: SBC_PRICE_THRESHOLD
        - name: AWS_REGION
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: AWS_REGION
        - name: AWS_ZONES
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: AWS_ZONES
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: sbc-secret
              key: AWS_ACCESS_KEY_ID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: sbc-secret
              key: AWS_SECRET_ACCESS_KEY
        - name: KOPS_STATE_STORE
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: KOPS_STATE_STORE
        - name: S3_BUCKET_NAME
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: S3_BUCKET_NAME
        - name: KOPS_CLUSTER_NAME
          valueFrom:
            configMapKeyRef:
              name: sbc-configmap
              key: KOPS_CLUSTER_NAME
    tolerations:
```

```
      - operator: "Exists"
        key: "node-role.kubernetes.io/master"
        effect: "NoSchedule"
    nodeSelector:
      kubernetes.io/role: "master"
    restartPolicy: OnFailure
```

ภาคผนวก ง

Log จากการทดสอบ scenario ที่ 2

```
 + creating session
 + success
 + creating new EC2 session
 + latest spot instance price: 0.017500
 + current spot price for this instance group: 0.017580
 + changing spot price to 0.017520


*********************************************************************

A new kubernetes version is available: 1.10.11
Upgrading is recommended (try kops upgrade cluster)

More information:
https://github.com/kubernetes/kops/blob/master/permalinks/upgrade_k8s
.md#1.10.11

*********************************************************************

I1203 01:30:14.243844      26 apply_cluster.go:510] Gossip DNS:
skipping DNS validation
I1203 01:30:14.508855      26 executor.go:103] Tasks: 0 done / 102
total; 36 can run
I1203 01:30:15.588402      26 executor.go:103] Tasks: 36 done / 102
total; 31 can run
I1203 01:30:16.790605      26 executor.go:103] Tasks: 67 done / 102
total; 27 can run
I1203 01:30:18.776292      26 executor.go:103] Tasks: 94 done / 102
total; 5 can run
I1203 01:30:19.321150      26 executor.go:103] Tasks: 99 done / 102
total; 3 can run
I1203 01:30:19.523284      26 executor.go:103] Tasks: 102 done / 102
total; 0 can run
I1203 01:30:19.523558      26 kubectl.go:134] error running kubectl
config view --output json
I1203 01:30:19.523574      26 kubectl.go:135]
I1203 01:30:19.523580      26 kubectl.go:136]
W1203 01:30:19.523598      26 update_cluster.go:278] error reading
kubecfg: error getting config from kubectl: error running kubectl:
exec: "kubectl": executable file not found in $PATH
I1203 01:30:19.536436      26 update_cluster.go:290] Exporting
kubecfg for cluster
kops has set your kubectl context to XXXXX-staging.k8s.local

Cluster changes have been applied to the cloud.
```

```
Changes may require instances to restart: kops rolling-update cluster


NAME   STATUS       NEEDUPDATE   READY  MIN    MAX    NODES
nodes  NeedsUpdate  8            0      8      8      8
I1203 01:30:22.210617      34 instancegroups.go:165] Draining the
node: "ip-10-101-73-25.ap-southeast-1.compute.internal".
node/ip-10-101-73-25.ap-southeast-1.compute.internal cordoned
node/ip-10-101-73-25.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-69xv4,
telegraf-ds-wfrt6, weave-net-tqpld
pod/tiller-deploy-84c4454dcb-cvgzh evicted
pod/kube-dns-autoscaler-787d59df8f-vnjws evicted
pod/[OMITTED]-7ff6b66884-x9zvl evicted
I1203 01:30:31.555562      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 01:32:01.555838      34 instancegroups.go:185] deleting node
"ip-10-101-73-25.ap-southeast-1.compute.internal" from kubernetes
I1203 01:32:01.564439      34 instancegroups.go:299] Stopping
instance "i-0070fb830655814de", node "ip-10-101-73-25.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 01:32:02.112125      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 01:36:02.112350      34 instancegroups.go:209] Validating the
cluster.
I1203 01:36:02.886222      34 instancegroups.go:276] Cluster
validated.
I1203 01:36:02.886266      34 instancegroups.go:165] Draining the
node: "ip-10-101-72-183.ap-southeast-1.compute.internal".
node/ip-10-101-72-183.ap-southeast-1.compute.internal cordoned
node/ip-10-101-72-183.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-dnk6w,
telegraf-ds-ntk84, weave-net-65g59
pod/[OMITTED]-7ff6b66884-1t955 evicted
I1203 01:36:20.970146      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 01:37:50.970374      34 instancegroups.go:185] deleting node
"ip-10-101-72-183.ap-southeast-1.compute.internal" from kubernetes
I1203 01:37:50.976984      34 instancegroups.go:299] Stopping
instance "i-0114fa481ec630f3d", node "ip-10-101-72-183.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 01:37:51.358603      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 01:41:51.358802      34 instancegroups.go:209] Validating the
cluster.
I1203 01:41:52.089702      34 instancegroups.go:276] Cluster
validated.
I1203 01:41:52.089728      34 instancegroups.go:165] Draining the
node: "ip-10-101-74-92.ap-southeast-1.compute.internal".
```

```
node/ip-10-101-74-92.ap-southeast-1.compute.internal cordoned
node/ip-10-101-74-92.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-827h4,
telegraf-ds-gk7vd, weave-net-chxqn
pod/tiller-deploy-84c4454dcb-qccjs evicted
pod/[OMITTED]-7f66fff84b-8kr6c evicted
pod/[OMITTED]-6789544959-868sd evicted
pod/ingress-nginx-controller-7589785f76-l4rz5 evicted
I1203 01:42:13.448663      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 01:43:43.448874      34 instancegroups.go:185] deleting node
"ip-10-101-74-92.ap-southeast-1.compute.internal" from kubernetes
I1203 01:43:43.455459      34 instancegroups.go:299] Stopping
instance "i-073aa9126bf78ce56", node "ip-10-101-74-92.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 01:43:43.878159      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 01:47:43.878366      34 instancegroups.go:209] Validating the
cluster.
I1203 01:47:44.870980      34 instancegroups.go:276] Cluster
validated.
I1203 01:47:44.871005      34 instancegroups.go:165] Draining the
node: "ip-10-101-72-230.ap-southeast-1.compute.internal".
node/ip-10-101-72-230.ap-southeast-1.compute.internal cordoned
node/ip-10-101-72-230.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-bwcjc,
telegraf-ds-rs7fc, weave-net-7l6hr
pod/[OMITTED]-7b7d8cc5d5-wnbvb evicted
I1203 01:47:49.960911      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 01:49:19.961120      34 instancegroups.go:185] deleting node
"ip-10-101-72-230.ap-southeast-1.compute.internal" from kubernetes
I1203 01:49:19.969064      34 instancegroups.go:299] Stopping
instance "i-07d5e3aacbd4a96bd", node "ip-10-101-72-230.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 01:49:20.344005      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 01:53:20.344198      34 instancegroups.go:209] Validating the
cluster.
I1203 01:53:21.087970      34 instancegroups.go:276] Cluster
validated.
I1203 01:53:21.087997      34 instancegroups.go:165] Draining the
node: "ip-10-101-74-156.ap-southeast-1.compute.internal".
node/ip-10-101-74-156.ap-southeast-1.compute.internal cordoned
node/ip-10-101-74-156.ap-southeast-1.compute.internal cordoned
WARNING: Deleting pods with local storage: kubernetes-dashboard-
7b9c7bc8c9-z2vlx; Ignoring DaemonSet-managed pods: fluentd-logger-
q5dwq, telegraf-ds-sbn48, weave-net-lh7q2
pod/kube-dns-autoscaler-787d59df8f-2f62g evicted
pod/default-http-backend-74c4b5b5b9-4bc9k evicted
```

```
pod/kubernetes-dashboard-7b9c7bc8c9-z2vlx evicted
I1203 01:53:41.517384      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
pod/[OMITTED]-8cb89d4b7-24bvz evicted
pod/[OMITTED]-85f6c885-f8jxt evicted
I1203 01:55:11.517608      34 instancegroups.go:185] deleting node
"ip-10-101-74-156.ap-southeast-1.compute.internal" from kubernetes
I1203 01:55:11.524122      34 instancegroups.go:299] Stopping
instance "i-09bf5236f3a947c8c", node "ip-10-101-74-156.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 01:55:11.910324      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 01:59:11.910653      34 instancegroups.go:209] Validating the
cluster.
I1203 01:59:12.566120      34 instancegroups.go:276] Cluster
validated.
I1203 01:59:12.566148      34 instancegroups.go:165] Draining the
node: "ip-10-101-75-14.ap-southeast-1.compute.internal".
node/ip-10-101-75-14.ap-southeast-1.compute.internal cordoned
node/ip-10-101-75-14.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-fzqgw,
telegraf-ds-jl74b, weave-net-n4xzr
pod/metrics-server-7996d5b6cb-z8qmd evicted
pod/trigger-controller-d57798bd5-th7gq evicted
pod/kube-dns-756bfc7fdf-hgl4r evicted
I1203 02:00:31.694638      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 02:02:01.694816      34 instancegroups.go:185] deleting node
"ip-10-101-75-14.ap-southeast-1.compute.internal" from kubernetes
I1203 02:02:01.702031      34 instancegroups.go:299] Stopping
instance "i-0c3a856bf47157a59", node "ip-10-101-75-14.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 02:02:02.070022      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 02:06:02.070235      34 instancegroups.go:209] Validating the
cluster.
I1203 02:06:03.106803      34 instancegroups.go:276] Cluster
validated.
I1203 02:06:03.106832      34 instancegroups.go:165] Draining the
node: "ip-10-101-75-138.ap-southeast-1.compute.internal".
node/ip-10-101-75-138.ap-southeast-1.compute.internal cordoned
node/ip-10-101-75-138.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-zfgqn,
telegraf-ds-thhpn, weave-net-sgss7
pod/[OMITTED]-68f7cbb4ff-s29wz evicted
pod/[OMITTED]-656dd47f5f-2xcsh evicted
pod/[OMITTED]-7ff6b66884-ndmrc evicted
I1203 02:06:16.465322      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
pod/[OMITTED]-5fb647ccf5-pfqnh evicted
```

```
I1203 02:07:46.465540      34 instancegroups.go:185] deleting node
"ip-10-101-75-138.ap-southeast-1.compute.internal" from kubernetes
I1203 02:07:46.487020      34 instancegroups.go:299] Stopping
instance "i-0cf82f53b42aedc7c", node "ip-10-101-75-138.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 02:07:47.054310      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 02:11:47.054485      34 instancegroups.go:209] Validating the
cluster.
I1203 02:11:48.023554      34 instancegroups.go:276] Cluster
validated.
I1203 02:11:48.023584      34 instancegroups.go:165] Draining the
node: "ip-10-101-73-201.ap-southeast-1.compute.internal".
node/ip-10-101-73-201.ap-southeast-1.compute.internal cordoned
node/ip-10-101-73-201.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-gdsnd,
telegraf-ds-nvrzs, weave-net-bvb7c
pod/[OMITTED]-6454487c5d-nxr97 evicted
pod/kube-dns-756bfc7fdf-99rcc evicted
I1203 02:12:49.092153      34 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 02:14:19.092411      34 instancegroups.go:185] deleting node
"ip-10-101-73-201.ap-southeast-1.compute.internal" from kubernetes
I1203 02:14:19.097714      34 instancegroups.go:299] Stopping
instance "i-0fa4ddf9cbd395f0f", node "ip-10-101-73-201.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 02:14:19.486475      34 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 02:18:19.486684      34 instancegroups.go:209] Validating the
cluster.
I1203 02:18:20.419878      34 instancegroups.go:276] Cluster
validated.
I1203 02:18:20.419914      34 rollingupdate.go:184] Rolling update
completed for cluster "XXXXX-staging.k8s.local"!
```

ภาคผนวก จ

Log จากการทดสอบ scenario ที่ 3

```
 + creating session
 + success
 + creating new EC2 session
 + latest spot instance price: 0.017500
 + current spot price for this instance group: 0.017
 + changing spot price to 0.017520


*********************************************************************

A new kubernetes version is available: 1.10.11
Upgrading is recommended (try kops upgrade cluster)

More information:
https://github.com/kubernetes/kops/blob/master/permalinks/upgrade_k8s
.md#1.10.11

*********************************************************************

I1203 07:30:11.269184      25 apply_cluster.go:510] Gossip DNS:
skipping DNS validation
I1203 07:30:11.554582      25 executor.go:103] Tasks: 0 done / 102
total; 36 can run
I1203 07:30:12.584104      25 executor.go:103] Tasks: 36 done / 102
total; 31 can run
I1203 07:30:13.765711      25 executor.go:103] Tasks: 67 done / 102
total; 27 can run
I1203 07:30:15.629045      25 executor.go:103] Tasks: 94 done / 102
total; 5 can run
I1203 07:30:16.165518      25 executor.go:103] Tasks: 99 done / 102
total; 3 can run
I1203 07:30:16.370593      25 executor.go:103] Tasks: 102 done / 102
total; 0 can run
I1203 07:30:16.370884      25 kubectl.go:134] error running kubectl
config view --output json
I1203 07:30:16.370962      25 kubectl.go:135]
I1203 07:30:16.370969      25 kubectl.go:136]
W1203 07:30:16.370986      25 update_cluster.go:278] error reading
kubecfg: error getting config from kubectl: error running kubectl:
exec: "kubectl": executable file not found in $PATH
I1203 07:30:16.381437      25 update_cluster.go:290] Exporting
kubecfg for cluster
kops has set your kubectl context to XXXXX-staging.k8s.local

Cluster changes have been applied to the cloud.
```

```
Changes may require instances to restart: kops rolling-update cluster


NAME    STATUS          NEEDUPDATE      READY   MIN     MAX     NODES
nodes   NeedsUpdate     8               0       8       8       8
I1203 07:30:19.021616      33 instancegroups.go:165] Draining the
node: "ip-10-101-75-54.ap-southeast-1.compute.internal".
node/ip-10-101-75-54.ap-southeast-1.compute.internal cordoned
node/ip-10-101-75-54.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-skz2q,
telegraf-ds-k99w6, weave-net-bmctn
pod/metrics-server-7996d5b6cb-8964d evicted
pod/trigger-controller-d57798bd5-dlb2h evicted
pod/kube-dns-756bfc7fdf-htgjv evicted
I1203 07:30:51.336256      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 07:32:21.336512      33 instancegroups.go:185] deleting node
"ip-10-101-75-54.ap-southeast-1.compute.internal" from kubernetes
I1203 07:32:21.350741      33 instancegroups.go:299] Stopping
instance "i-0035a8bc7590ecefd", node "ip-10-101-75-54.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 07:32:21.772616      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 07:36:21.772834      33 instancegroups.go:209] Validating the
cluster.
I1203 07:36:22.798318      33 instancegroups.go:276] Cluster
validated.
I1203 07:36:22.798346      33 instancegroups.go:165] Draining the
node: "ip-10-101-73-69.ap-southeast-1.compute.internal".
node/ip-10-101-73-69.ap-southeast-1.compute.internal cordoned
node/ip-10-101-73-69.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-vpwlv,
telegraf-ds-sxxw9, weave-net-jfgfh
I1203 07:36:26.876021      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
pod/[OMITTED]-7b7d8cc5d5-vr2wg evicted
I1203 07:37:56.876241      33 instancegroups.go:185] deleting node
"ip-10-101-73-69.ap-southeast-1.compute.internal" from kubernetes
I1203 07:37:56.883893      33 instancegroups.go:299] Stopping
instance "i-0163482e6df8409cb", node "ip-10-101-73-69.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 07:37:57.243423      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 07:41:57.243646      33 instancegroups.go:209] Validating the
cluster.
I1203 07:41:58.318300      33 instancegroups.go:276] Cluster
validated.
I1203 07:41:58.318320      33 instancegroups.go:165] Draining the
node: "ip-10-101-72-185.ap-southeast-1.compute.internal".
node/ip-10-101-72-185.ap-southeast-1.compute.internal cordoned
node/ip-10-101-72-185.ap-southeast-1.compute.internal cordoned
```

```
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-xddkl,
telegraf-ds-wzqdd, weave-net-c9kdd
pod/tiller-deploy-84c4454dcb-42xd8 evicted
pod/ingress-nginx-controller-7589785f76-9c2c6 evicted
pod/[OMITTED]-656dd47f5f-skbnx evicted
pod/kube-dns-756bfc7fdf-c767w evicted
I1203 07:43:22.552497      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 07:44:52.552727      33 instancegroups.go:185] deleting node
"ip-10-101-72-185.ap-southeast-1.compute.internal" from kubernetes
I1203 07:44:52.561680      33 instancegroups.go:299] Stopping
instance "i-02cb452390aea73b3", node "ip-10-101-72-185.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 07:44:53.015700      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 07:48:53.022974      33 instancegroups.go:209] Validating the
cluster.
I1203 07:48:53.896121      33 instancegroups.go:276] Cluster
validated.
I1203 07:48:53.896148      33 instancegroups.go:165] Draining the
node: "ip-10-101-74-52.ap-southeast-1.compute.internal".
node/ip-10-101-74-52.ap-southeast-1.compute.internal cordoned
node/ip-10-101-74-52.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-dmfhq,
telegraf-ds-m66kh, weave-net-54qll; Deleting pods with local storage:
kubernetes-dashboard-7b9c7bc8c9-wgffq
pod/kube-dns-autoscaler-787d59df8f-8x8ft evicted
pod/trigger-controller-d57798bd5-qq8nd evicted
pod/[OMITTED]-68f7cbb4ff-mpgr7 evicted
pod/kubernetes-dashboard-7b9c7bc8c9-wgffq evicted
I1203 07:49:14.988468      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
pod/default-http-backend-74c4b5b5b9-mb9ff evicted
I1203 07:50:44.988674      33 instancegroups.go:185] deleting node
"ip-10-101-74-52.ap-southeast-1.compute.internal" from kubernetes
I1203 07:50:44.996039      33 instancegroups.go:299] Stopping
instance "i-05f4af775750cf107", node "ip-10-101-74-52.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 07:50:45.411233      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 07:54:45.411463      33 instancegroups.go:209] Validating the
cluster.
I1203 07:54:46.411839      33 instancegroups.go:276] Cluster
validated.
I1203 07:54:46.411865      33 instancegroups.go:165] Draining the
node: "ip-10-101-74-72.ap-southeast-1.compute.internal".
node/ip-10-101-74-72.ap-southeast-1.compute.internal cordoned
node/ip-10-101-74-72.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-fpmgh,
telegraf-ds-9wm26, weave-net-g2pzk
pod/[OMITTED]-6454487c5d-kcmnk evicted
```

```
pod/kube-dns-756bfc7fdf-4l5z9 evicted
I1203 07:55:37.704752      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 07:57:07.704938      33 instancegroups.go:185] deleting node
"ip-10-101-74-72.ap-southeast-1.compute.internal" from kubernetes
I1203 07:57:07.711251      33 instancegroups.go:299] Stopping
instance "i-09185c8dd45cea62d", node "ip-10-101-74-72.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 07:57:08.485604      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 08:01:08.485815      33 instancegroups.go:209] Validating the
cluster.
I1203 08:01:09.524558      33 instancegroups.go:276] Cluster
validated.
I1203 08:01:09.524583      33 instancegroups.go:165] Draining the
node: "ip-10-101-73-37.ap-southeast-1.compute.internal".
node/ip-10-101-73-37.ap-southeast-1.compute.internal cordoned
node/ip-10-101-73-37.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-2hszt,
telegraf-ds-khtxd, weave-net-5lkdr
I1203 08:01:20.619852      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
pod/[OMITTED]-7ff6b66884-wmpwv evicted
I1203 08:02:50.619990      33 instancegroups.go:185] deleting node
"ip-10-101-73-37.ap-southeast-1.compute.internal" from kubernetes
I1203 08:02:50.627456      33 instancegroups.go:299] Stopping
instance "i-096b43e1babc43f02", node "ip-10-101-73-37.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 08:02:51.150677      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 08:06:51.150860      33 instancegroups.go:209] Validating the
cluster.
I1203 08:06:53.091886      33 instancegroups.go:276] Cluster
validated.
I1203 08:06:53.091913      33 instancegroups.go:165] Draining the
node: "ip-10-101-74-157.ap-southeast-1.compute.internal".
node/ip-10-101-74-157.ap-southeast-1.compute.internal cordoned
node/ip-10-101-74-157.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-mnlkt,
telegraf-ds-vk8q6, weave-net-8dqcx
pod/[OMITTED]-5fb647ccf5-s5b47 evicted
pod/[OMITTED]-7f66fff84b-pc8f2 evicted
pod/[OMITTED]-6789544959-q99rs evicted
pod/[OMITTED]-85f6c885-mb76g evicted
I1203 08:07:05.328527      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 08:08:35.328746      33 instancegroups.go:185] deleting node
"ip-10-101-74-157.ap-southeast-1.compute.internal" from kubernetes
I1203 08:08:35.336949      33 instancegroups.go:299] Stopping
instance "i-0a136165913790144", node "ip-10-101-74-157.ap-southeast-
```

```
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 08:08:36.424970      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 08:12:36.425177      33 instancegroups.go:209] Validating the
cluster.
I1203 08:12:37.462860      33 instancegroups.go:276] Cluster
validated.
I1203 08:12:37.462887      33 instancegroups.go:165] Draining the
node: "ip-10-101-75-250.ap-southeast-1.compute.internal".
node/ip-10-101-75-250.ap-southeast-1.compute.internal cordoned
node/ip-10-101-75-250.ap-southeast-1.compute.internal cordoned
WARNING: Ignoring DaemonSet-managed pods: fluentd-logger-zdm29,
telegraf-ds-rxqzw, weave-net-ktht2
pod/metrics-server-7996d5b6cb-ng2n4 evicted
pod/[OMITTED]-656dd47f5f-rp7zp evicted
pod/[OMITTED]-59d7789dfb-gpldx evicted
pod/[OMITTED]-6789544959-wcbl5 evicted
I1203 08:12:56.753200      33 instancegroups.go:358] Waiting for
1m30s for pods to stabilize after draining.
I1203 08:14:26.753494      33 instancegroups.go:185] deleting node
"ip-10-101-75-250.ap-southeast-1.compute.internal" from kubernetes
I1203 08:14:26.761243      33 instancegroups.go:299] Stopping
instance "i-0eaf1ae638629eb0b", node "ip-10-101-75-250.ap-southeast-
1.compute.internal", in group "nodes.XXXXX-staging.k8s.local" (this
may take a while).
I1203 08:14:27.161033      33 instancegroups.go:198] waiting for 4m0s
after terminating instance
I1203 08:18:27.161456      33 instancegroups.go:209] Validating the
cluster.
I1203 08:18:27.892117      33 instancegroups.go:276] Cluster
validated.
I1203 08:18:27.892155      33 rollingupdate.go:184] Rolling update
completed for cluster "XXXXX-staging.k8s.local"!
```