การพัฒนาคลาวด์ด้วยโอเพ่นแสตค

OPENSTACK CLOUD DEVELOPMENT

ชญานนท์ วิวัฒนะวัฒนาการ

ญาณิศา ยิ้มสุวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2558

การพัฒนาคลาวด์ด้วยโอเพ่นแสตค

OPENSTACK CLOUD DEVELOPMENT

ชญานนท์ วิวัฒนะวัฒนาการ

ญาณิศา ยิ้มสุวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2558

ปริญญานิพนธ์ปีการศึกษา 2558

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาคลาวด์ด้วยโอเพ่นแสตค

OPENSTACK CLOUD DEVELOPMENT

ผู้จัดทำ

1. นายชญานนท์ วิวัฒนะวัฒนาการ รหัสนักศึกษา 55010214
2. นางสาวญาณิศา ยิ้มสุวรรณ รหัสนักศึกษา 55010302

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา

การพัฒนาคลาวด์ด้วยโอเพ่นแสตค

นายชญานนท์ วิวัฒนะวัฒนาการ 55010214
นางสาวญาณิศา ยิ้มสุวรรณ 55010302
ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2558

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้มีวัตถุประสงค์ในการนำเทคโนโลยีการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) เข้าประยุกต์ใช้ เข้ามาช่วยปรับปรุงระบบพีซี (PC: Personal Computer) ของผู้ใช้งาน แทนที่จะต้องซื้อฮาร์ดแวร์ (Hardware) ที่มีประสิทธิภาพสูงมาใช้งานเป็นพีซีแต่ละเครื่องสำหรับผู้ใช้งานแต่ละคน ซึ่งผู้ใช้งานแต่ละคนต่างก็ไม่ได้ใช้งานประสิทธิภาพของฮาร์ดแวร์เหล่านั้นอย่างเต็มที่ตลอดเวลา เทคโนโลยีเวอร์ชวลเดสก์ทอปอินฟราสตรัคเจอร์ (The Virtual Desktop Infrastructure Technology) จะทำการยุบรวมอิมเมจ (Image) ของคอมพิวเตอร์ในองค์กรทั้งหมดมาอยู่บนเวอร์ชวลไลเซชันอินฟราสตรัคเจอร์ (Virtualization Infrastructure)

สถาปัตยกรรม Cloud นั้นต้องการที่จะมุ่งเน้นถึงกลุ่มของทรัพยากรที่ให้ผู้ใช้งานได้ใช้โดยทั่วไป โดยทรัพยากรเหล่านี้ได้รวมถึงการคำนวณการใช้ทรัพยากรในปริมาณต่างๆกัน, สตอเรจ (Storage) สำหรับเก็บทรัพยากร และทรัพยากรด้านเน็ตเวิร์ค (Network) โดยการนำมาใช้ร่วมกันนั่นเอง และทำการแจกจ่ายออกสู่อินเตอร์เน็ต (Internet) โดยบริการคลาวด์ (Cloud) นั้นสามารถพัฒนาปรับปรุงการใช้งานเซิร์ฟเวอร์ (Server) บนเวอร์ชวลไลเซชั่น (Virtualization) โดยผู้ใช้บริการนั้นสามารถที่จะเริ่มหรือปิดเวอร์ชวลอินสแตนซ์ (Virtual Instance) ได้ตามต้องการ

ในส่วนของการพัฒนาได้เลือกใช้ OpenStack ในการพัฒนาเนื่องจากเป็น Open source project โดยที่ OpenStack เป็นระบบปฏิบัติการแบบกลุ่มเมฆที่คอยควบคุมกลุ่ม compute node ขนาดใหญ่, storage และทรัพยากรเน็ตเวิร์ค (Network) จะใช้การจัดการทั้งหมดผ่านทางแดชบอร์ด (Dashboard) ที่จะให้ผู้ดูแลเป็นคนควบคุมในขณะที่เพิ่มศักยภาพของผู้ใช้งานในการให้ใช้ทรัพยากรผ่านทางเว็บ อินเตอร์เฟส (Web interface) และเรายังทดลองใช้ Docker containers เพื่อเพิ่มประสิทธิภาพการทำงานของ OpenStack อีกด้วย

OpenStack Cloud Development

Mr. Chayanon Viwattanawattanakarn 55010214

Ms. Yanisa Yimsuwan 55010302

Dr. Voravat Limpoka Advisor

Academic Year 2015

ABSTRACT

The purpose of this report is to present the cloud computing technology in the way of use its benefit, which is to improve user's PC (PC: Personal Computer) system and usage. Instead of purchasing expensive systems and equipment on the business, we can reduce the costs by using the cloud computing resources from service providers.

All of the images will be merged with the Virtual Desktop Infrastructure Technology into Virtualization Infrastructure.

Cloud computing architectures tend to focus on a common set of resources that are virtualized and exposed to a user on an on-demand basis. These resources include compute resources of varying capability, persistent storage resources, and configurable networking resources to tie them together in addition to conditionally exposing these resources to the internet. . With a cloud, users can start and shut down virtual instances as needed.

In our project, we had decided to use OpenStack since it was released as an open source. OpenStack is a cloud computing platform which manages the lifecycle of compute instances in an OpenStack environment. Including manages storage, networking by using Dashboard to interact with underlying OpenStack services via the web interface. We also experienced Docker Containers to as a way of managing multiple containers on a single machine. However used behind Nova makes it much more powerful since it's then possible to manage several hosts, which in turn manage hundreds of containers. The current Docker project aims for full OpenStack compatibility.

กิตติกรรมประกาศ

คณะผู้จัดทำขอขอบคุณอาจารย์ที่ปรึกษาดร.วรวัฒน์ ลิ้มโภคาที่คอยให้ความสนใจสอบถามถึงความคืบหน้าของงานให้คำแนะนำและให้ความช่วยเหลือในเรื่องต่างๆอีกทั้งยังให้แนวคิดและประสบการณ์ที่ดีแก่คณะผู้จัดทำนอกจากนี้ยังขอขอบคุณคุณศิลาณี จรัสวชิรกุล และคุณชานนท์ ทรัพย์สำราญ ที่ช่วยเหลือให้คำแนะนำที่ดีกับคณะผู้จัดทำมาโดยตลอด

ขอขอบพระคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสถาบันการศึกษาในอดีต ที่ให้โอกาสดีๆทางการศึกษาแก่ข้าพเจ้ามาโดยตลอดสำหรับคุณงามความดีอันใดที่เกิดจากรายงานเล่มนี้คณะผู้จัดทำขอมอบให้บิดามารดาซึ่งเป็นที่รักและเคารพยิ่งตลอดจนครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้และประสบการณ์ที่ดีแก่คณะผู้จัดทำ

ชญานนท์ วิวัฒนะวัฒนาการ

ญาณิศา ยิ้มสุวรรณ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

การเปลี่ยนแปลงของโครงสร้างธุรกิจในยุคปัจจุบันที่มีการแข่งขันสูงขึ้น เรื่องความรวดเร็วในการทำธุรกิจถือเป็นปัจจัยหลักในการเอาชนะคู่แข่งขันได้ ระบบไอทีจึงเป็นสิ่งสำคัญที่ต้องสามารถตอบสนองการเติบโตและการเปลี่ยนแปลงอย่างรวดเร็วของโมเดลธุรกิจ จึงได้เกิดการพัฒนาระบบ Cloud Computing เป็นระบบการบริหารจัดการทรัพยากรไอทีแบบใหม่ ที่มีความยืดหยุ่นสูง เพื่อรองรับกับธุรกิจยุคใหม่ที่พร้อมจะเปลี่ยนแปลงในทุกวินาที ระบบการจัดการทรัพยากรทางด้านไอทีแบบ Cloud Computing ส่งผลให้ธุรกิจสามารถเพิ่ม ลด และปรับเปลี่ยนระบบไอทีได้ทันทีที่ต้องการตามนโยบายการดำเนินธุรกิจ โดยไม่จำเป็นต้องรอการสั่งซื้ออุปกรณ์เพิ่มเติม เพราะโครงสร้างของระบบถูกออกแบบให้ทำงานอยู่บนระบบเสมือน (Virtualization) ทำให้สามารถเพิ่ม ลด และปรับเปลี่ยนโครงสร้างระบบไอทีที่รองรับการทำงานของ Software และ Application ที่จำเป็นต่อธุรกิจได้ทันที และปัจจุบันได้มี Open source ที่พัฒนาโดย NASA และ Rackspace ซึ่งก็คือ OpenStack เพื่อที่จะใช้พัฒนา Cloud ได้สะดวกและมีประสิทธิภาพยิ่งขึ้น

1.2 วัตถุประสงค์ของการศึกษา

วัตถุประสงค์ของรายงานฉบับนี้จัดทำเพื่อศึกษาและทดลองการทำงานของ Neutron โดยเลือกใช้ในรูปแบบ GRE tunnels เพื่อแบ่ง Tenant ออกจากกัน โดยการทดลองและศึกษาค้นคว้านี้มีประสงค์เพื่อที่จะทำให้ทราบและเข้าใจถึงโครงสร้างหลักการทำงานการใช้งานรวมถึงประโยชน์ที่จะได้รับ

1.3 ขอบเขตของโครงงาน

ศึกษาการใช้งานและติดตั้งระบบ Cloud โดยใช้ OpenStack และวิเคราะห์ข้อดีและเสียของ Network ต่างๆ ที่ Neutron สามารถแบ่งออกได้ และทำการทดลองใช้ Storage ของ OpenStack อีกด้วย

1.4 วิธีการดำเนินการ

1) ทำการศึกษาข้อมูลเบื้องต้นเพื่อทำความเข้าใจในเรื่อง Cloud Computing, Virtualization และ OpenStack
2) ทำการศึกษาเรื่อง Network (Neutron) และ Storage (Cinder, Swift) ของ OpenStack
3) ทำการศึกษา Architecture ของ Neutron
4) ทำการศึกษาเรื่อง GlusterFS ในเบื้องต้น
5) ทำการศึกษาการใช้และทดลองสร้าง Controller, Compute, Storage
6) ทำการทดลองใช้งานและทดลองสร้าง Volume 3 แบบประกอบด้วย Striped, Distributed และ Replicated

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้รับความรู้ความเข้าใจถึงโครงสร้างและการทำงานของการประมวลผลแบบกลุ่มเมฆ (Cloud Computing)
2) ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานการใช้งาน OpenStack
3) ได้รับความรู้ความเข้าใจถึงการทำงานของระบบ Network (Neutron)
4) ได้รับความรู้ความเข้าใจในการใช้การทำ storage ซึ่งเป็นหนึ่งใน Service ของ OpenStack
5) ช่วยให้เข้าใจถึงกระบวนการพัฒนาระบบตั้งแต่การวางแผนการวิเคราะห์การออกแบบการสร้างและการนำไปใช้งาน
6) ได้รับความรู้และทักษะในการใช้ลินุกซ์ (linux) เช่น CentOS 7
7) ได้รับความรู้ความเข้าใจในการใช้เครื่องมือจัดการระบบการประมวลผลแบบกลุ่มเมฆ (Openstack)
8) ได้รับความเข้าใจเกี่ยวกับการให้บริการคลาวด์เซิร์ฟเวอร์ (Cloud Server) ซึ่งเป็นการให้บริการด้านโครงสร้างพื้นฐาน Iaas (Infrastructure as a Service)

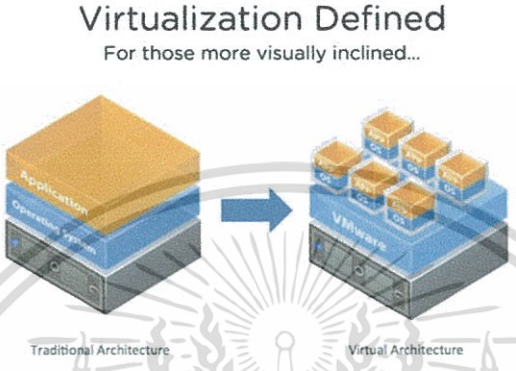
1.6 ส่วนประกอบของรายงาน

รายงานฉบับนี้ได้แบ่งเนื้อหาออกเป็น 4 บทด้วยกันคือ

1) บทที่ 1 บทนำกล่าวถึงความสำคัญและที่มาของโครงงานวัตถุประสงค์ของโครงงาน ขอบเขตของโครงงานวิธีการดำเนินการประโยชน์ที่คาดว่าจะได้รับและส่วนประกอบของรายงาน
2) บทที่ 2 ทฤษฎีที่เกี่ยวข้องกล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงงานประกอบด้วยอะไรบ้างให้บรรยายทฤษฎีทั้งหมดโดยละเอียด
3) บทที่ 3 การวิเคราะห์และออกแบบ โครงสร้างของระบบ
4) บทที่ 4 การทดลองกล่าวถึงรายละเอียดของโครงงานนี้ส่วนที่ได้ออกแบบและทดลองการทำงานของระบบ
5) บทที่ 5 บทสรุปและข้อแนะนำ จะกล่าวถึงรายละเอียดโดยสรุปของการทดลองที่ได้ศึกษา

บทที่ 2
ทฤษฎีพื้นฐานที่เกี่ยวข้อง

2.1 เทคโนโลยีการทำเวอร์ชวลไลเซชัน (Virtualization Technology)

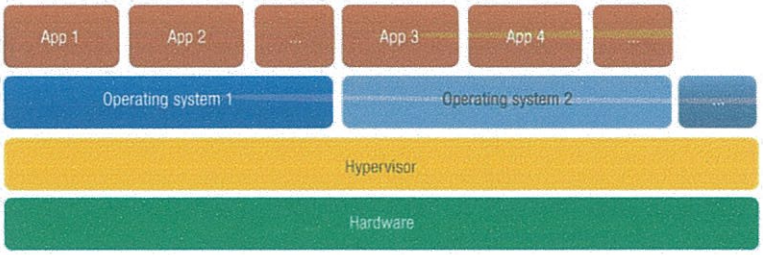


รูป 2.1โครงสร้าง Virtualization

Virtualization คือ การจำลองเครื่องเสมือนด้วยซอฟต์แวร์ ที่ทำให้คอมพิวเตอร์หนึ่งเครื่องสามารถทำงานเป็นเครื่องเสมือนหลายๆ ระบบได้ โดยแต่ละระบบมีทรัพยากรหน่วยความจำ, ฮาร์ดดิสก์ และอุปกรณ์เนตเวิร์กเสมือนที่เป็นอิสระต่อกัน เครื่องเสมือนแต่ละเครื่องจึงสามารถมีระบบปฏิบัติการและซอฟต์แวร์เป็นของตนเองโดยอิสระ

2.1.1 ประเภทของ Hypervisor

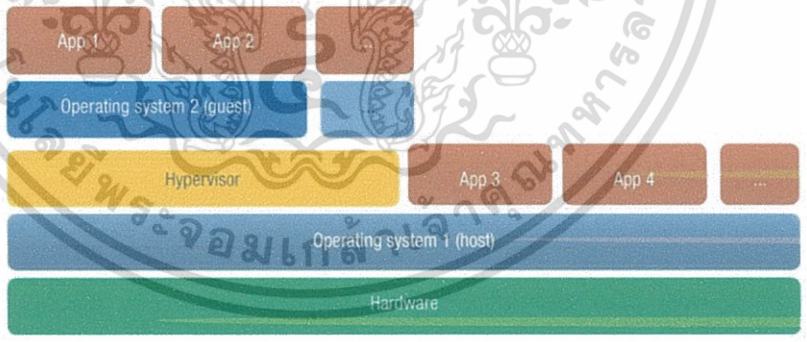
การสร้างระบบ Virtualization ได้กำหนดชื่อเรียกตัว software ที่ทำหน้าที่ virtual ว่า Hypervisor หรือVirtual Machine Manager (VMM) ได้มีการแบ่ง hypervisor ออกเป็น 2 ประเภทคือ



รูป 2.2Type 1 Hypervisor

"Type 1 Hypervisor" สถาปัตยกรรมเนทีฟ (Native บางทีเรียก Bare-Metal หรือ Enterprise Architecture) เป็นสถาปัตยกรรมที่ Hypervisor ถูกติดตั้งโดยตรงลงบนฮาร์ดแวร์ เสมือนหนึ่งเป็นระบบปฏิบัติการของเครื่องไปโดยปริยาย ด้วยโครงสร้างแบบนี้ทำให้ทำงานได้ประสิทธิภาพสูงกว่าสถาปัตยกรรมแบบโฮสต์ (Host) อย่างมิอาจเทียบเคียง

จุดด้อยของสถาปัตยกรรมแบบ Native ก็คือ การรองรับของฮาร์ดแวร์ เครื่องที่จะนำมาใช้งานบนโครงสร้างนี้ต้องผ่านการรับรองจากผู้ผลิต Hypervisor เสียก่อน ถึงจะมั่นใจว่าสามารถทำงานร่วมกันได้เป็นอย่างดี แม้จะรองรับบนฮาร์ดแวร์เฉพาะบางรุ่น แต่ด้วยความร้อนแรงของโลกเสมือน ทำให้เครื่องเซิร์ฟเวอร์รุ่นใหม่ๆ ล้วนตบเท้าดาหน้าเข้ามาเป็นสาวกโดยพร้อมเพรียงกัน จะหาเครื่องเซิร์ฟเวอร์รุ่นที่ทำงานกับ Native Hypervisor ไม่ใช่เรื่องยากเกินไปนักในยุคปัจจุบัน

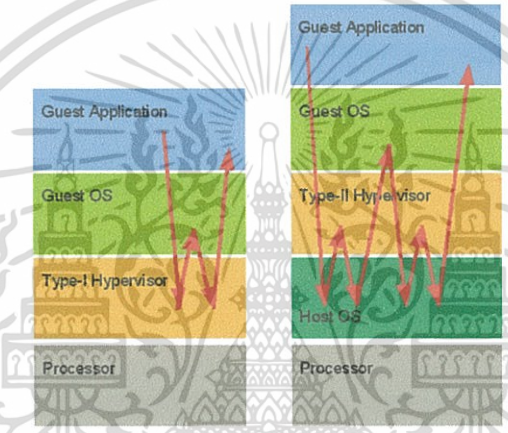


รูป 2.3Type 2 Hypervisor

"Type 2 Hypervisor"ในสถาปัตยกรรมแบบนี้ นอกจากที่เราจะต้องมีตัวเครื่อง คือ ฮาร์ดแวร์ หรือที่เรียกว่า "โฮสต์ (Host)" แล้ว เรายังต้องมี "ระบบปฏิบัติการโฮสต์ (Host Operating System)" อยู่ด้วยHypervisor เลเยอร์ของไฮเปอร์ไวเซอร์(Virtualization Layer) จะถูกติดตั้งลงบนระบบปฏิบัติการ Host ซึ่งถือได้ว่า Hypervisor เป็นแอพพลิเคชั่นตัวหนึ่งบน Hots จากนั้นเราจึงจะสามารถสร้างจักรกลเสมือนขึ้นมาได้ระบบปฏิบัติการที่ถูกติดตั้งลงบนจักรกลเสมือนจะถูกเรียกว่า

"ระบบปฏิบัติการเกสต์ (Guest Operation System)" โดยที่ระบบปฏิบัติการ Host และ Guest Operating System จะเป็นตัวเดียวกันหรือต่างกันก็ตามสะดวก สถาปัตยกรรมแบบ Host นิยมใช้สำหรับการทดสอบระบบ หรือการเขียนโปรแกรม ซึ่งส่วนใหญ่มักนิยมติดตั้งลงบนเครื่องพีซี หรือโน๊ตบุ๊คและใช้งานส่วนตัว

ข้อดีของสถาปัตยกรรม Host คือ สามารถติดตั้งได้ง่าย, ไม่สนใจฮาร์ดแวร์ด้านล่าง, มีเพียงระบบปฏิบัติการบนเครื่อง Host ที่รองรับก็เพียงพอ ปัจจุบันทั้งวินโดวส์, ลินุกซ์ รวมถึงแม็คโอเอส (Mac OS) ก็มี Hypervisor แบบนี้ให้ใช้งานกัน

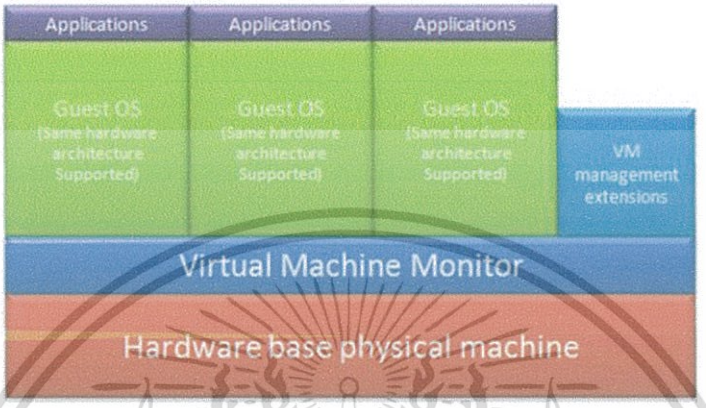


รูป 2.4 Type 1: Native หรือ Bare Metal

สำหรับข้อด้อย คือ จักรกลเสมือนจะทำงานได้ช้า เพราะถ้าดูจากโครงสร้างแล้ว กว่าแอพพลิเคชั่นบนจักรกลเสมือนจะเข้าถึงฮาร์ดแวร์จริงๆ จะต้องผ่านระบบปฏิบัติการตัวใหญ่ถึง 2 ตัว คือ ระบบปฏิบัติการ Host และระบบปฏิบัติการของ Guest เอง แล้วยังต้องนับ Hypervisor เข้าไปอีก ผู้ที่เคยใช้งานจักรกลเสมือนบนโครงสร้างแบบนี้ ถึงกับข้องใจในความล่าช้า แต่โครงสร้างนี้ถูกออกแบบมาสำหรับการใช้งานเพื่อทดสอบหรือพัฒนาโปรแกรม รวมถึงงานกาฝากขนาดเล็กเท่านั้น เช่น เราอาจจะมีโน้ตบุ๊กที่ติดตั้ง Windows 7 ไว้ แล้วต้องการทดสอบการทำงานของ Windows Server 2008 R2 เราก็เพียงแต่สร้างจักรกลเสมือนขึ้นมาเพื่อติดตั้ง Windows Server 2008 R2 เท่านั้น ทำให้เราไม่จำเป็นต้องหาเครื่องใหม่มาใช้ในการทดสอบ เป็นต้น

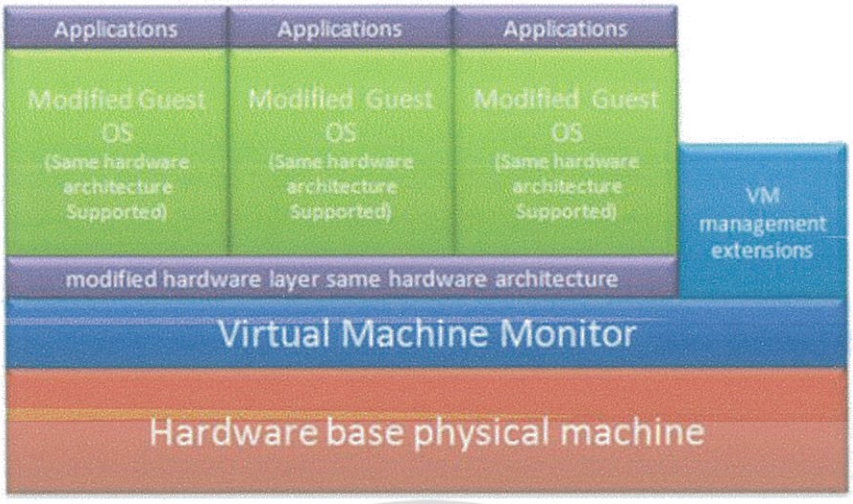
2.1.2 สถาปัตยกรรมของการทำเวชวลไลเซชัน

มีทางเลือกมากมายในการ สนับสนุนเทคโนโลยีการทำงานของเวชวลไลเซชัน แต่มีเทคนิคอยู่ 2 รูปแบบที่เป็นผู้นำทางด้านเทคโนโลยีนี้อยู่ก็คือเทคนิคแบบ Full virtualization และ Para-virtualization



รูป 2.5 Full Virtualization

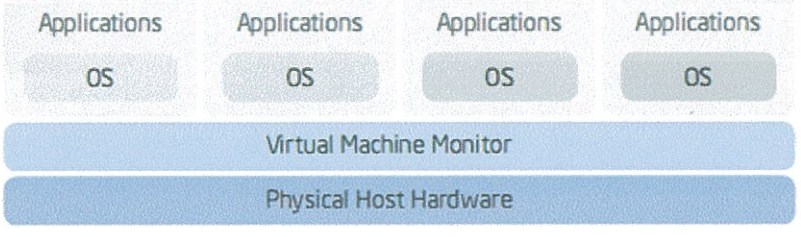
การทำ Full virtualization สำหรับการทำเวชวลไลเซชันในรูปแบบนี้ ถูกออก แบบเพื่อเตรียมการทำให้เป็นรูปแบบเสมือนทั้งหมดของฮาร์ดแวร์ และสร้างระบบเสมือนที่สมบูรณ์ ในที่นี้จะทำให้เราสามารถที่นำ ระบบปฏิบัติการอื่นๆ มาติดตั้งและสามารถที่จะทำงานอยู่บนเครื่องคอมพิเตอร์เดียวกันได้ ซึ่งเราจะเรียกว่าระบบปฏิบัติที่มาติดตั้งเพิ่มเติมนี้ว่า ระบบปฏิบัติการเยือน (Guest Operating System: GOS) โดยที่ระบบปฏิบัติการเยือนสามารถที่จะทำงานได้โดยไม่ต้องมีการแก้ไขเปลี่ยน แปลงสิ่งใดๆ กับคำสั่งที่ถูกร้องขอจากระบบปฏิบัติการเยือนนั้นๆ หรือในตัวโปรแกรมของมันเอง เพราะฉะนั้น ระบบปฏิบัติการเยือนหรือโปรแกรม จะไม่ทราบถึงสภาพแวดล้อมจำลองเสมือนจริงที่เกิดขึ้น จึงทำให้ระบบปฏิบัติการเยือนและโปรแกรมของมันทำงานอยู่บน เวชวลแมชชีน ในขณะที่ในความจริงแล้วจะต้องทำงานบนสภาวะแวดล้อมของระบบจริงๆ (Physical system) วิธีการนี้ทำให้เกิดประโยชน์ เพราะว่ามันได้แยกการเชื่อมต่อของซอฟต์แวร์และระบบปฏิบัติการเยือน ออกจากฮาร์ดแวร์อย่างสมบูรณ์ ดังนั้นผลลัพธ์ของวิธีการแบบ Full virtualization ก็คือสามารถให้มีเส้นทางการเคลื่อนย้ายของตัวซอฟต์แวร์ และ ภาระงานต่างๆ (workloads) ระหว่างระบบปฏิบัติการที่มีคุณสมบัติที่แตกต่างกัน ตัวอย่างของซอฟต์แวร์เวชวลไลเซชัน ที่ใช้เทคนิค Full virtualization ก็คือ Microsoft Virtual Server, และ VMware ESX Server



รูป 2.6 Paravirtualization

การทำ Para-virtualization เป็น อีกวิธีการหนึ่งในการทำเวชวลไลเซชัน โดยนำเสนอให้แต่ละ เวชวลแมชชีน คือรูปแบบเสมือนของฮาร์ดแวร์ที่ถูกนำเสนอเช่นเดียวกันกับแบบ Full virtualization แต่มีสิ่งที่ไม่เหมือนกันก็คือในเทคนิคแบบนี้จะสามารถระบุไปถึงภายในกายภาพของฮาร์ดแวร์ (Physical Hardware) โดยเทคนิค Para-virtualization ต้องการที่จะมีการเปลี่ยนแปลงแก้ไขคำร้องขอของระบบปฏิบัติการเยือน ที่กำลังทำงานอยู่บนเวชวลแมชชีน ผลลัพธ์ของมันก็คือระบบปฏิบัติการเยือน จะรับรู้ได้ว่ามันกำลังทำงานอยู่บนซอฟต์แวร์เวชวลแมชชีนนั่นเอง มีการยอมรับว่าประสิทธิภาพที่ได้จะใกล้เคียงกับประสิทธิภาพตามธรรมชาติของ ระบบปฏิบัติการเยือน วิธีการของ Para- virtualization ยังคงดำเนินการพัฒนาและยังมีข้อจำกัดอยู่ เช่นการเกิดแคชของข้อมูลของระบบปฏิบัติการเยือน (Guest Operating System Cache Data) และการเชื่อมต่อกันที่ยังไม่มีความน่าเชื่อถือเพียงพอ (Unauthenticated Connections)

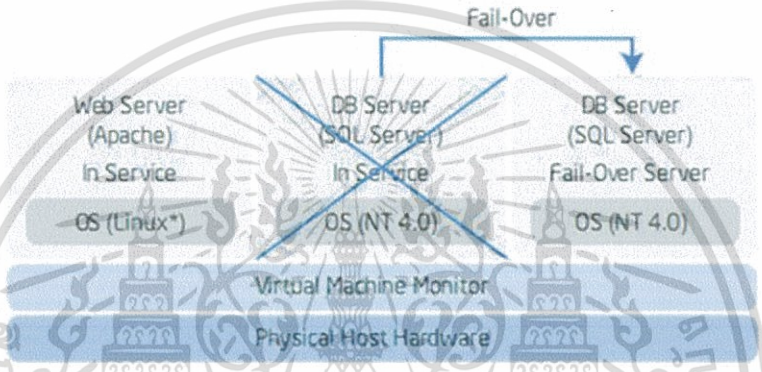
2.1.3 การใช้งานเทคโนโลยีเวอร์ชวลไลเซชัน (Usage Virtualization Technology)



รูป 2.7 การทำงานของเทคโนโลยีเวอชวลไลเซชัน

2.1.3.1 Consolidation

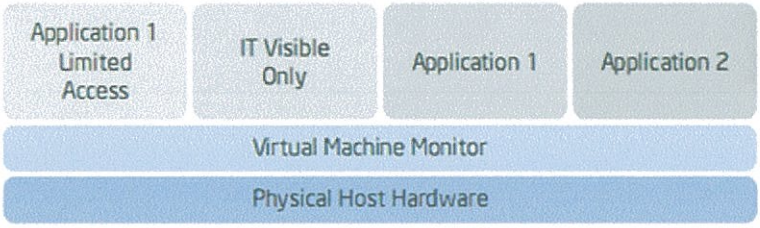
การรวบรวมทรัพยากรต่างๆ เช่น Storage และ Server เข้ามาอยู่จุดเดียว ทำการเพิ่มอัตราการใช้งานฮาร์ดแวร์ที่มีอยู่อย่างจำกัด ให้เป็นไปอย่างมีประสิทธิภาพ และคุ้มค่ามากที่สุด เนื่องจากฮาร์ดแวร์บางตัวอาจจะใช้งานไม่มากนัก ขณะที่บางตัวใช้งานมากจนเกินไป, ช่วยประหยัดพลังงานค่าไฟ ประหยัดพื้นที่ ประหยัดค่าใช้จ่ายต่างๆ การดูแลบริหารจัดการและบำรุงรักษาระบบทำได้ง่ายขึ้น เนื่องจากมีฮาร์ดแวร์เพียงชุดเดียวเท่านั้น สามารถติดตั้งได้หลายระบบปฏิบัติการและแอพพลิเคชั่นบนสภาพแวดล้อมที่แตกต่างกัน เนื่องจากบางแอพพลิเคชัน (Legacy Applications) จำเป็นต้องทำงานบนระบบปฏิบัติการที่จำเพาะ เป็นการยืดอายุการใช้งานแอพพลิเคชั่นเก่า



รูป 2.8 การทำงานของเทคโนโลยีเวอชวลไลเซชัน กรณีเฟลล์โอเวอร์

2.1.3.2 Reliability

ความน่าเชื่อถือในการทำงาน, เพิ่มความสะดวกและความคล่องตัวทางธุรกิจ ระบบเสมือนสามารถจัดเตรียมเพื่อรองรับหรือปรับขนาดได้ภายในไม่กี่นาทีเพื่อรองรับการติดตั้งแอพพลิเคชั่นใหม่ ภาระงานที่เพิ่มขึ้นหรือกรณีที่เกิดความผิดพลาดขึ้นในระบบ ทำการ Backup และ Recovery สามารถทำงานได้อย่างรวดเร็วขึ้น ภายใต้ Virtual Machine เดียวกัน และรองรับการทำงานหลากหลายอย่าง อาทิ System Migration, Backup และ Recovery



รูป 2.9 การทำงานของเทคโนโลยีเวอชวลไลเซชัน ในด้าน Reliability

2.1.3.3 Security

การรักษาความปลอดภัยที่ดีขึ้น สามารถกำหนดระดับความปลอดภัยให้แต่ละระบบเสมือนให้มีความแตกต่างกันได้ การโจมตีในรูปแบบดิจิตอล (Virus, Hacker) จะถูกแยกออกจากกัน สำหรับแต่ละระบบ และการทำงานที่ล้มเหลวส่วนใหญ่มาจากซอร์ฟแวร์ จึงมั่นใจได้ว่าความผิดพลาดที่เกิดขึ้นจะไม่ส่งผลกระทบต่อระบบเสมือนอื่นๆ

2.2 Cloud computing

2.2.1 แนวคิด

Cloud Computing เป็นแนวคิดด้านบริการที่เชื่อมโยงกันโดยคอมพิวเตอร์ต่างๆ ที่ทำงานร่วมกัน ซึ่งอาจตั้งอยู่ในห้องเดียวกันหรือคนละที่ก็ได้ โดยระบบจะทำงานประสานกันแบบรวมศูนย์ คือผู้ใช้ไม่จำเป็นต้องสนใจเลยว่าระบบนั้นมีการทำงานอย่างไร และประกอบไปด้วยทรัพยากร (resource) อะไรบ้าง แต่ผู้ใช้แค่ระบุความต้องการ (requirement) ไปยังซอฟต์แวร์ของระบบ Cloud Computing (ซอฟต์แวร์จะร้องขอให้ระบบจัดสรรทรัพยากรและบริการให้ตรงกับความต้องการผู้ใช้ โดยระบบสามารถเพิ่มและลดจำนวนของทรัพยากรรวมถึงเสนอบริการให้พอเหมาะกับความต้องการของผู้ใช้ได้ตลอดเวลา) และหลังจากนั้นบริการ (service) ก็จะให้ผลลัพธ์แก่ผู้ใช้ ส่วนบริการจะไปจัดการกับทรัพยากรอย่างไรนั้นผู้ใช้ไม่จำเป็นต้องสนใจ ดังนั้นจึงสรุปได้ว่า ผู้ใช้นั้นจะมองเห็นเพียงบริการซึ่งทำหน้าที่เสมือนซอฟต์แวร์ที่ทำงานตามความต้องการของผู้ใช้ โดยที่ผู้ใช้ไม่จำเป็นต้องทราบถึงทรัพยากรที่แท้จริงว่ามีอะไรบ้างและถูกจัดการเช่นไร หรือถูกเก็บอยู่ที่ไหน

2.2.2 โครงสร้างของระบบ Cloud computing

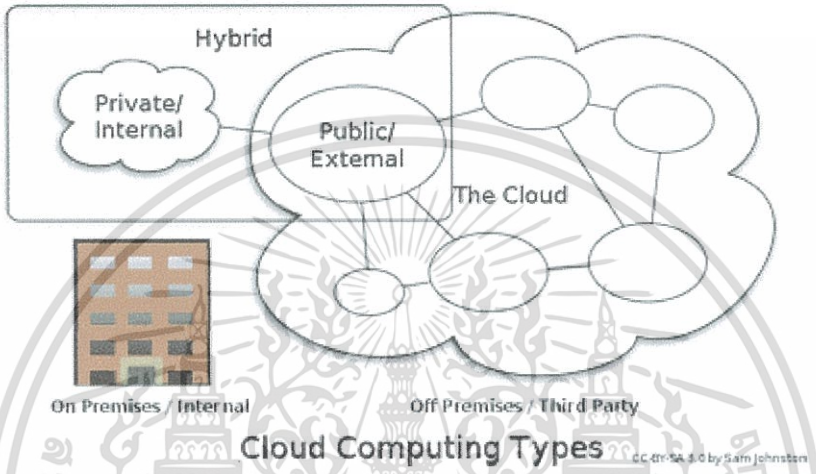
การประมวลผลแบบกลุ่มเมฆจะมีโครงสร้างดังนี้

1) กลุ่มเมฆของเซอร์ฟเวอร์ (cloud server) ซึ่งเป็นเซอร์ฟเวอร์จำนวนมหาศาลนับหมื่นนับแสนเครื่องที่ตั้งอยู่ในที่เดียวกัน กลุ่มเมฆนี้ต่อเชื่อมเข้าหากันด้วยเครือข่าย เป็นระบบกริด ในระบบนี้จะใช้ซอฟต์แวร์เวอร์ชัวไลเซชันในการทำงาน
2) ส่วนติดต่อกับผู้ใช้ (User interaction interface) ทำหน้าที่รับคำขอบริการจากผู้ใช้ในรูปแบบเว็บโปรโตคอล
3) ส่วนจัดเก็บรายการบริการ (Services Catalog) เก็บและบริหารรายการของบริการผู้ใช้สามารถค้นดูบริการที่มีจากที่นี่
4) ส่วนบริหารงาน (system management) ทำหน้าที่กำหนดทรัพยากรที่เหมาะสมเมื่อผู้ใช้เรียกใช้บริการ เมื่อมีการขอใช้บริการ ข้อมูลการขอ request จะถูกส่งผ่านให้ส่วนนี้

5) ส่วนจัดหาทรัพยากร (provisioning services) จากนั้นส่วนบริหารงานจะติดต่อกันส่วนนี้ เพื่อจองทรัพยากรจากกลุ่มเมฆ

6) ส่วนตรวจสอบข้อมูลการใช้งาน (Monitoring and Metering) เพื่อใช้ในการเก็บค่าบริการหรือเก็บข้อมูลสถิติเพื่อปรับปรุงระบบต่อไป

2.2.3 ชนิดของระบบ Cloud computing



รูป 2.10 รูปแบบของ Cloud Computing

1) Public cloud หรือเรียกว่า External cloud หมายถึงระบบ cloud computing ที่มีทรัพยากรเป็นสาธารณะ สามารถบริการผ่านทาง Internet, Web Application หรือ Web service ให้บริการการแชร์ทรัพยากรและ Utilities ขั้นพื้นฐาน

2) Private cloud หรือ Internal cloud เป็นคำศัพท์ใหม่ที่ผู้ให้บริการเพิ่งใช้ในการอธิบายระบบ cloud computing ใน private network โดยผู้ให้บริการอ้างว่าระบบนี้เป็นการรักษาสิทธิประโยชน์ของระบบ cloud และ เป็นศูนย์กลางของการรักษาความปลอดภัยให้กับข้อมูล และมีความแน่นอนที่เชื่อถือได้

3) Hybrid cloud จะประกอบขึ้นด้วยผู้ให้บริการแบบ Internal และ external เห็นได้จากรัฐวิสาหกิจส่วนมาก

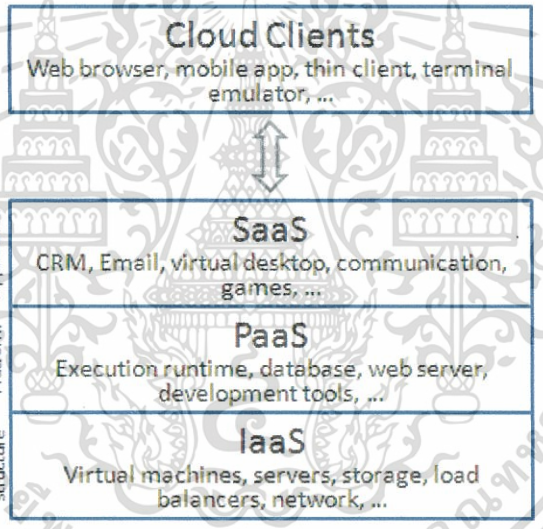
2.2.4 ประเภทของบริการที่นำเสนอในการ ประมวลผลในกลุ่มเมฆ

1) การให้บริการซอฟต์แวร์ หรือ Software as a Service (SaaS)
จะให้บริการการประมวลผลแอปพลิเคชันที่แม่ข่ายของผู้ให้บริการ และเปิดให้การบริการทางด้านซอฟต์แวร์ต่างๆ

ซึ่งบริการ Software as a Service ที่ใกล้ตัวเรามากที่สุดก็คือ GMail นั่นเอง นอกจากนั้นก็เช่น Google Docs หรือ Google Apps ที่เป็นรูปแบบของการใช้งานซอฟต์แวร์ผ่านเว็บบราวเซอร์ ทำบน Server ของ Google ทำให้เราไม่ต้องการเครื่องที่มีกำลังประมวลผลสูงหรือพื้นที่เก็บข้อมูลมากๆ

2) การให้บริการแพลตฟอร์ม หรือ Platform as a Service (PaaS)
เป็นการประมวลผล ซึ่งมีระบบปฏิบัติการ และการสนับสนุนเว็บแอปพลิเคชันเข้ามาร่วมด้วย

3) การให้บริการโครงสร้างพื้นฐาน หรือ Infrastructure as a Service (IaaS)
เป็นการให้บริการเฉพาะโครงสร้างพื้นฐาน มีประโยชน์ในการประมวลผลทรัพยากรจำนวนมาก ตัวอย่างบริการอื่นๆในกลุ่มนี้ก็เช่น Google Compute Engine, Amazon Web Services, Microsoft Azure



รูป 2.11 ประเภทของการให้บริการของ Cloud Computing

2.2.5 ข้อดีและเสียของ Cloud computing

ข้อดีของ Cloud Computing

1) ลดต้นทุนค่าดูแลบำรุงรักษาเนื่องจากค่าบริการได้รวมค่าใช้จ่ายตามที่ใช้งานจริง เช่น ค่าจ้างพนักงาน ค่าซ่อมแซม ค่าลิขสิทธิ์
2) มีความยืดหยุ่นในการเพิ่มหรือลดระบบตามความต้องการ
3) ได้เครื่องเซิร์ฟเวอร์ที่มีประสิทธิภาพ มีระบบสำรองข้อมูลที่ดี มีเครือข่ายความเร็วสูง
4) มีผู้เชี่ยวชาญดูแลระบบและพร้อมให้บริการช่วยเหลือ 24 ชั่วโมง

ข้อเสียของ Cloud Computing

1) เนื่องจากเป็นการใช้ทรัพยากรที่มาจากหลายที่หลายแห่งทำให้อาจมีปัญหาในเรื่องของความต่อเนื่อง และความเร็วในการเข้าถึงทรัพยากรมากกว่าการใช้บริการแม่ข่ายที่อยู่ภายในองค์กร
2) ยังไม่มีการรับประกันในการท างานอย่างต่อเนื่องของระบบและความปลอดภัยของข้อมูล
3) แพลตฟอร์มยังไม่มีมาตรฐาน ทำให้ผู้ใช้มีข้อจำกัดสำหรับตัวเลือกในการพัฒนาหรือติดตั้งระบบ

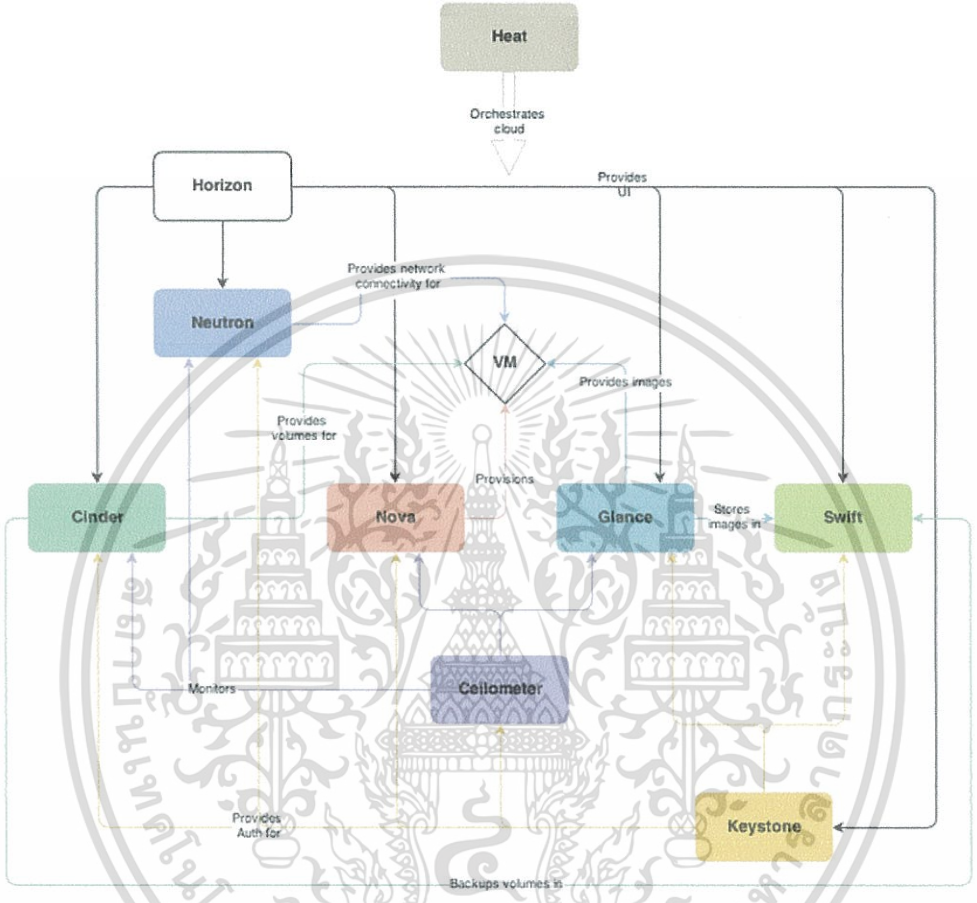
2.2.6 ทำไมบริการ Cloud Computing จึงได้รับความนิยม?

Cloud Computing คือบริการที่เราใช้หรือเช่าใช้ระบบคอมพิวเตอร์หรือทรัพยากรด้านคอมพิวเตอร์ ของผู้ให้บริการ เพื่อนำมาใช้ในการทำงาน โดยที่เราไม่จำเป็นต้องลงทุนซื้อ Hardware และ Software เองทั้งระบบ ไม่ต้องวางระบบเครือข่ายเอง ลดความรับผิดชอบในการดูแลระบบลง (เพราะผู้ให้บริการจะเป็นผู้ดูแลให้เอง) แถมตอนอัพเกรดระบบยังทำได้ง่ายกว่า ผู้ใช้ทุกคนสามารถเข้าถึงระบบ ข้อมูลต่างๆ ผ่านอินเทอร์เน็ต สามารถจัดการ บริหารทรัพยากรของระบบ ผ่านเครือข่าย และมีการแบ่งใช้ทรัพยากรร่วมกัน (shared services) ได้ด้วย และการจ่ายเงินเพื่อเช่าระบบ ก็สามารถจ่ายตามความต้องการของเรา ใช้เท่าไหร่ จ่ายเท่านั้นได้ หากวันใดความต้องการมีมากขึ้นก็สามารถซื้อเพิ่มเติมเพื่อเพิ่มศักยภาพของระบบ Cloud Computing ได้ โดยที่ไม่ต้องอัพเกรดระบบและเครื่องคอมพิวเตอร์ให้วุ่นวาย

2.3 OpenStack

2.3.1 องค์ประกอบของ OpenStack

ความสัมพันธ์ในการทำงานร่วมกันของแต่ละ Component ใน OpenStack



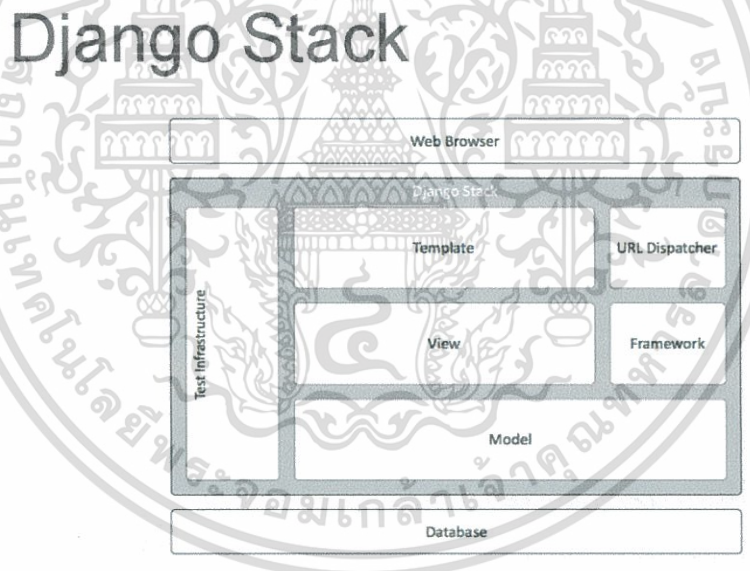
รูป 2.12 OpenStack Architecture

1) Heat – จะทำหน้าที่เป็น Orchestration tools เตรียมสถานการณ์ตั้งค่าต่างๆ ของแต่ล ะ Component ไว้เป็น Template เพื่อให้สะดวกในการปรับใช้งานแอพพลิเคชั่นได้ง่ายขึ้น
2) Horizon – คอยแสดงผล User interface ของ Component ดังนี้ Cinder , Neutron , Nova , Glance , Swift และ Keystone
3) Neutron – ให้บริการเชื่อมต่อเครือข่ายสำหรับ VM
4) Cinder – ให้บริการสร้างฮาร์ดดิสก์สำหรับ VM และสำรองข้อมูลฮาร์ดดิสก์ไปให้ Swift
5) Nova – จัดเตรียมและทำหน้าที่ควบคุม VM

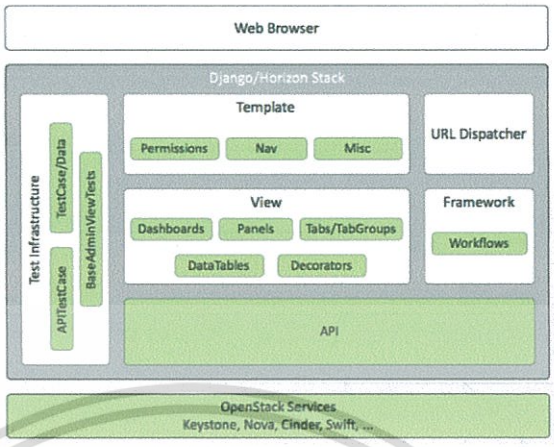
6) Glance – เป็นตัวจัดการไฟล์ Image ISO ที่จะไปทำการติดตั้งใน VM โดยจัดเก็บไว้ใน Swift
7) Swift – เป็น OpenStack object store project ที่ให้บริการ Cloud storage software เพื่อที่จะทำการจัดเก็บและเรียกใช้ข้อมูลต่างๆได้ ประมาณว่าเป็นไดเรคทอรี่ ที่สามารถเอาไป mount กับเครื่อง VM เพื่อให้เข้าถึงไฟล์ และใช้จัดเก็บไฟล์ได้
8) Ceilometer – คอยวัดปริมาณการใช้งานของ Cinder , Neutron , Nova และ Glance
9) Keystone – คอยตรวจสอบสิทธิ์การใช้งานการเข้าถึงสำหรับ Cinder . Neutron , Nova , Ceilometer , Glance และ Swift

2.3.2 Horizon

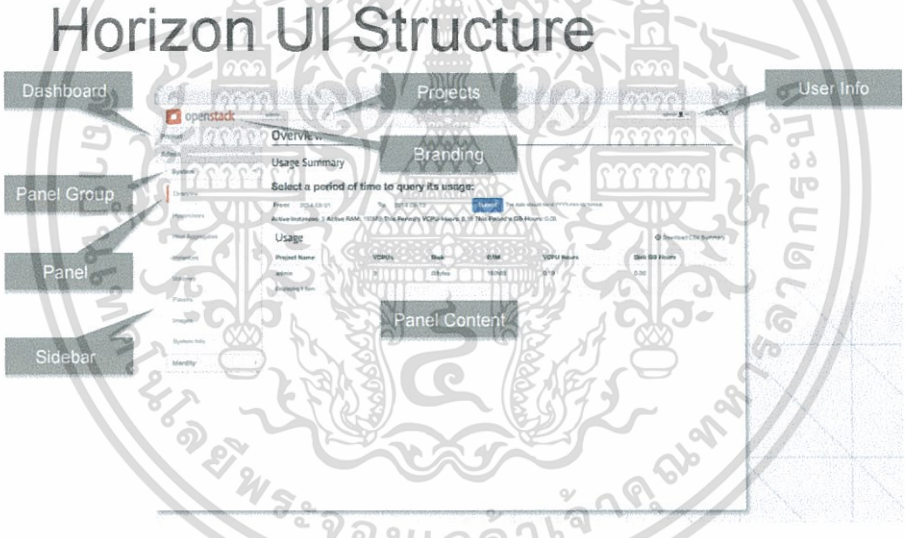
สถาปัตยกรรม Horizon นั้นเป็น Django-based application โดยใช้งานผ่าน Apache และ WSGI เพื่อให้สามารถเข้าถึงการใช้งาน Services ต่างๆของ OpenStack โดยมีเทคโนโลยีที่สามารถใช้ร่วมได้เช่น Bootstrap, jQuery เป็นต้น



รูป 2.13 Django stack Architecture



รูป 2.14 Horizon stack ใน Django



รูป 2.15 Horizon แดชบอร์ด

2.3.3 Nova

Nova ถูกออกแบบมาให้เอื้ออำนวยต่อความ Scalable, ตรงต่อความต้องการ/การเรียกร้อง และ Self-service access ต่อการ Compute ทรัพยากร

2.3.3.1 API

Nova นั้นใช้ v2 API มาเป็นเวลานาน ปัจจุบันกำลังจะเปลี่ยนไปใช้ API v2.1 ซึ่ง v2.1 API นั้นสร้างโดยใช้ Microversions ซึ่ง API Microversions นั้นอนุญาตการเปลี่ยนแปลง API โดยที่ยังสามารถรักษาความ Compatible กับรุ่นเก่าไว้ได้

2.3.3.2 Hypervisors

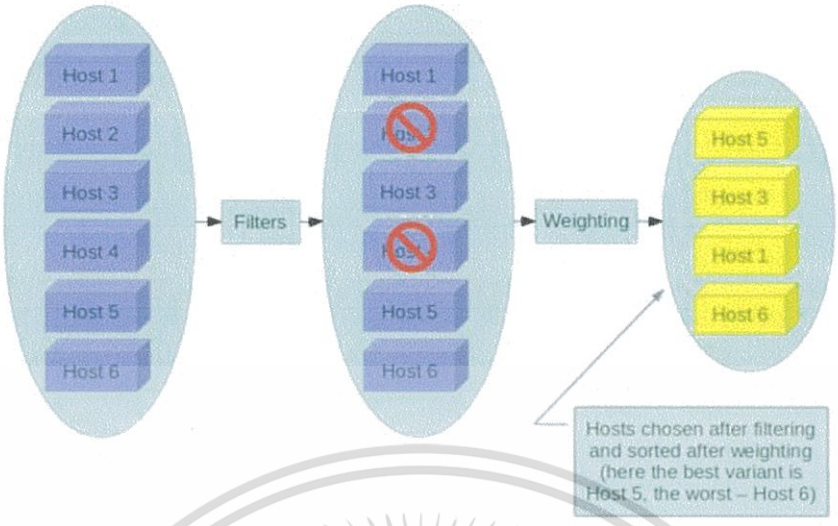
OpenStack compute สามารถทำงานผ่าน APIs ของไฮเปอร์ไวเซอร์ (XenAPI สำหรับ XenServer / XCP, libvirt สำหรับ KVM หรือ QEMU ฯลฯ)

โดย Nova Juno นั้นรองรับ Matrix ดังนั้นเมื่อการตัดสินใจว่า Capabilities ใดต้องคำนึงถึงPrinciples เหล่านี้

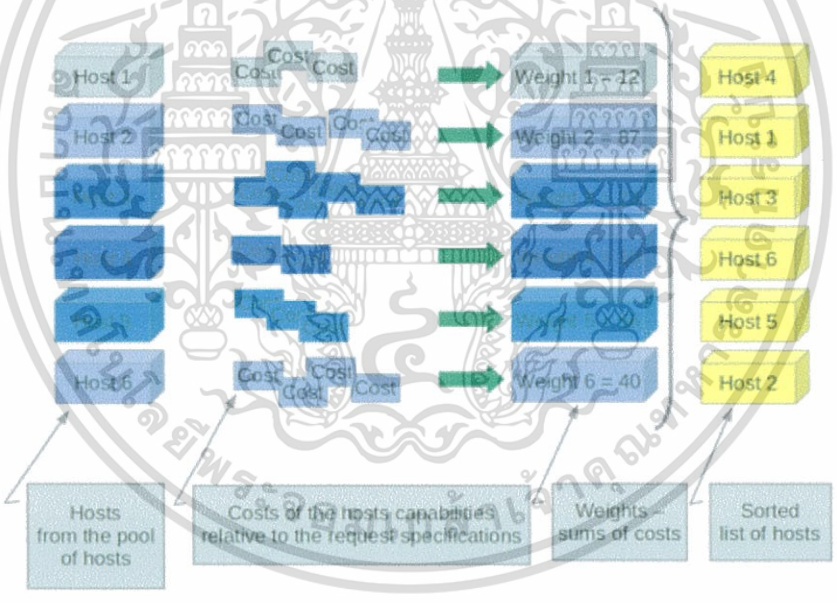
1) Inclusivity – Requirements ควรมีความครอบคลุมต่อการใช้งานของ User
2) Bootstrapping – Use case test ที่เหมาะสมที่สุดคือการคำนึงถึง Starting point สำหรับ Compute deploy คือ Empty data center ด้วย Machines ใหม่และ Network Connectivity
3) Competition - ช่วงแรกเริ่มผู้นำ Cloud compute service คือ Amazon EC2 จึงทำให้มีการตรวจสอบว่า Feature การใช้งานนั้นมีอยู่ใน EC2 หรือไม่ ซึ่งปรากฏว่าการทำเช่นนี้ไม่ได้เกิดประโยชน์ในการใช้งานอย่างสูงสุด แต่อย่างใด
4) Reality – ปัจจุบันมี Drivers ที่สามารถใช้งานกับ Nova ได้แต่ละตัวจะ Support Feature set ของมันเอง

2.3.3.3 Scheduling

Filter scheduler support การ Filtering และ Weighting เพื่อทำการตัดสินใจว่า Instance ใหม่นั้นควรจะสร้างไว้ที่ใด ซึ่ง Scheduler นี้จะ support กับการทำงานกับCompute nodes เท่านั้น Filter scheduler ใช้ Weights ระหว่างทำงาน Weigher คือการเลือก Host ที่ดีที่สุดจาก Group ของ Valid hosts โดยให้น้ำหนักแก่ทุก Host ใน List



รูป 2.16 การทำ Filtering



รูป 2.17 การทำ Weighting

2.3.3.4 Host Aggregates

Host Aggregates นั้นสามารถเปรียบได้ว่าเป็น Mechanism ในการเพิ่ม Availability Zone ของ Partition โดย Availability Zone จะ visible แก่ Users ส่วน Host Aggregates จะ visible ต่อ Administrators เท่านั้น

2.3.3.5 Threading model

OpenStack services ใช้ Green thread model ในการทำ threading และ Implemented โดยใช้ Python libraries โดย Green threads ใช้ cooperative model ของ Threading Thread context switches สามารถเกิดขึ้นได้เมื่อ Python libraries (evenlet, greenlet) ทำการ Call ในมุมมองของ Operating system แต่ละ OpenStack service run แบบ Single thread

2.3.3.6 Internationalization

Nova ใช้ oslo.i18n library เพื่อ support internationalization ซึ่ง Term ของ Internationalization ใช้เพื่ออนุญาตให้สามารถ Adapt Software ใน ภาษา หรือ technical ในวงกว้าง รวมทั้ง support non-ASCII character sets, แปลง Strings

2.3.3.7 AMQP and Nova

AMQP คือ messaging technology ที่ถูกเลือกโดย OpenStack cloud ซึ่ง AMQP broker ไม่ว่าจะเป็น RabbitMQ หรือ Qpid จะอยู่ระหว่าง 2 Nova components และ Allow ให้สามารถติดต่อสื่อสารกันได้อย่าง Loosely coupled และแม่นยำมากขึ้นโดย Nova components นั้นอาศัย Remote Procedure Calls (RPC) ในการ communicate ต่อกัน

2.3.3.8 Hooks

Hooks อำนวยเรื่อง Mechanism เพื่อ extend Nova กับ custom code ผ่านทาง Plugin

2.3.3.9 Block Device Mapping ใน Nova

Nova มี concept ของ Block devices ซึ่งสามารถแบ่งออกเป็น Cloud instances ได้ เมื่อพูดถึง Block device mapping เราจะกล่าวถึง 2 สิ่งนี้คือ API/CLI structure & syntax สำหรับระบุถึง Block devices สำหรับ Instance boot request และ Data structure internal ของ Nova ที่ใช้สำหรับ Record และ จัดเก็บซึ่งจะพบใน block_device_mapping table

2.3.4 Neutron

เช่นเดียวกับหลายบริการ OpenStack นิวตรอนเป็นการตั้งค่าชั้นสูงเนื่องจากเป็นสถาปัตยกรรมแบบปลั๊กอิน ปลั๊กอินเหล่านี้รองรับอุปกรณ์เครือข่ายและซอฟต์แวร์ที่แตกต่างกัน

2.3.4.1 Concepts

Neutron ให้บริการด้าน network ระหว่างอุปกรณ์อินเตอร์เฟซที่จัดการตามการให้บริการ OpenStack อื่น ๆ (ส่วนมากNova) มันช่วยให้ผู้ใช้สามารถสร้างเครือข่ายของตัวเองและจากนั้นแนบการเชื่อมต่อเซิร์ฟเวอร์กับพวกเขา สถาปัตยกรรมแบ็กเอนด์แบบเสียบได้ช่วยให้ผู้ใช้ใช้ประโยชน์จาก commodity gear หรืออุปกรณ์ที่ผู้ขายได้รับการสนับสนุน และการขยายอนุญาตให้เพิ่มบริการเครือข่ายซอฟต์แวร์หรือฮาร์ดแวร์ที่จะบูรณการ แกนหลักของ Neutron API

ประกอบด้วยการสนับสนุน Layer 2 networking และการจัดการ IP address(IPAM) เช่นเดียวกับส่วนขยายสำหรับ Layer 3 โครงสร้างเราเตอร์ที่ช่วยให้สามารถกำหนดเส้นทางระหว่างLayer 2 networks และ gatewaysไปยังnetworkภายนอก Neutronประกอบด้วยรายการที่เพิ่มขึ้นของปลั๊กอินที่ช่วยให้การทำงานร่วมกับความหลากหลายในเชิงพาณิชย์ Network เทคโนโลยีที่เป็นopen source รวมทั้ง Routers, Switches, Virtual switches และ Software-defined networking (SDN) Controllers

2.3.4.2 Ports

Portsใน Neutron หมายถึงการเชื่อมต่อของ virtual switch การเชื่อมต่อเหล่านี้เป็นที่ที่ตัวอย่างและnetwork services แนบไปกับเครือข่าย เมื่อเชื่อมต่อกับเครือข่ายย่อยพวกเขากำหนด MAC และIP addresses ของอินเตอร์เฟซที่เสียบเข้ากับพวกเขา

2.3.4.3 Networks

Neutron กำหนด network คือที่แยกได้ Layer 2 network segments ผู้ควบคุมจะเห็นเครือข่ายที่เป็น logical switches ดำเนินการโดย Linux bridging tools เปิด vSwitch หรือบางซอฟแวร์อื่น ๆ ซึ่งแตกต่างจากเครือข่ายทางกายภาพที่จะถูกกำหนดโดยทั้งผู้ควบคุมหรือผู้ใช้ของ OpenStack

2.3.4.4 Subnets

Subnets ใน Neutron เป็นตัวแทนกลุ่มของ IP addresses (IPv4 หรือ IPv6) ที่เกี่ยวข้องกับระบบเครือข่าย ที่อยู่เหล่านี้จะถูกกำหนดให้กับตัวอย่างที่เกี่ยวข้องกับเครือข่าย

2.3.4.5 Routers

เกตเวย์ระหว่างเครือข่าย

2.3.4.6 Private and Floating IPs

Private and floating IP addresses หมายถึงไอพีแอดเดรสที่กำหนดให้ตัวอย่าง Private IP addresses สามารถมองเห็นได้ในกรณีและมักจะเป็นส่วนหนึ่งของเครือข่ายส่วนตัวที่ให้สำหรับผู้เช่า เครือข่ายนี้จะช่วยให้ตัวอย่างของผู้เช่าในการสื่อสารในขณะที่แยกจากผู้เช่าคนอื่น ๆ Private IP addresses มักจะมองไม่เห็นอินเทอร์เน็ต Floating IPs เป็น virtual IPsที่Neutron ส่งไปยัง private IP ของตัวอย่างผ่านการแปลงที่อยู่เครือข่าย (NAT) นี้จะกระทำโดยทั่วไปบนโหนดเครือข่ายซึ่งมีการเข้าถึงอินเทอร์เน็ตสำหรับตัวอย่าง

2.3.4.7 Network Services

นอกเหนือจากorchestrationระดับต่ำของ Neutronของ layer 1 ผ่าน 3 ส่วนเช่น IP addresses, subnet และ routers ยังสามารถจัดการบริการระดับที่สูงขึ้น ตัวอย่างเช่น Neutron ให้ load balancer เป็นบริการ (LBaaS) ที่ใช้ ha-proxy กระจาย traffic ระหว่างการประมวลผลหลายกรณี Vendor enhancements, Network Namespaces

FIXME Network namespaces แยกกรณีของ network interfaces และตารางกำหนดเส้นทางที่ทำงานอิสระจากกัน

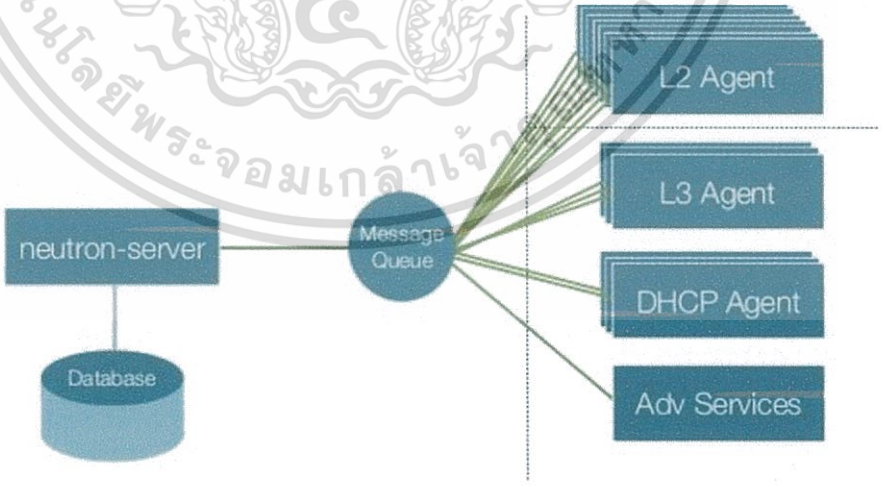
FIXME Namespaces ช่วยให้หลายกรณีของตารางเส้นทางที่จะร่วมอยู่ใน Linux box เดียวกัน (เช่นเส้นทางเสมือนจริงและการส่งต่อ [VRF] ในเราเตอร์) ภายในเครือข่ายและช่องว่างเครือข่ายย่อยต่อผู้เช่า พวกเขาแนะนำขอบเขตทั้งหมดของความยืดหยุ่นของเครือข่ายซึ่งมีความสำคัญต่อ production OpenStack ในการนำไปใช้งาน แต่ยังสามารถขัดแย้งกับ logic ที่ใช้โดยผู้ดูแลระบบ IP network ที่มีประสบการณ์ และนำไปสู่การแก้ไขปัญหา

FIXME ประโยชน์ใหญ่ของการดำเนินการของ namespaces ใน neutron คือการที่ผู้เช่าสามารถสร้างทับซ้อนกันของ IP address สถานการณ์ที่ให้อิสระกับผู้ใช้ระบบคลาวด์เพราะพวกเขามีอิสระที่จะสร้างเครือข่ายย่อยใดๆ ของทางเลือก โดยไม่ต้องกลัวที่ขัดแย้งกันกับที่ของผู้เช่าอื่นๆ Linux network namespace จะต้องทำงานบนโหนด neutron-l3-agent หรือ neutron-dhcp-agentถ้า IP ทับซ้อนในการใช้ ดังนั้นโฮสต์ที่ใช้กระบวนการเหล่านี้จะต้องสนับสนุน network namespaces

2.3.4.8 Open vSwitch

Open source programmable virtual switch รองรับ OpenFlow, 802.1Q VLANs, LACP, STP ที่รองรับ KVM และ Xen OVS เป็นพื้นฐานสำหรับSDN/network virtualization platform ที่แตกต่างกัน ควบคุมความยืดหยุ่นใน user-space Fast datapath ในkernel Port อาจมีมากกว่าหนึ่งอินเตอร์เฟซการสนับสนุน IEEE 802.1Q ติดแท็ก VLANไปอินเตอร์เฟซแพ็คเก็ตจะถูกส่งต่อโดย flow Fine-grained ACLsและ QoS (L2-L4 การจับคู่การกระทำ)

2.3.4.9 Neutron Architecture



รูป 2.18 Neutron Architecture

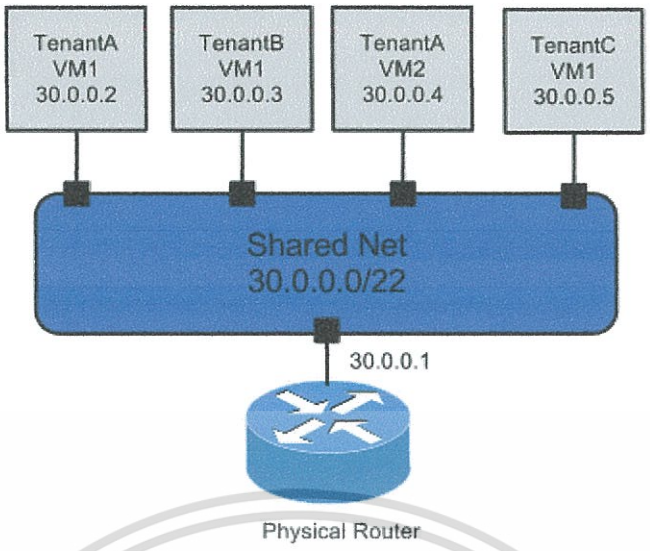
Neutron ประกอบไปด้วย Components ดังนี้

1) API เป็นแกนหลักของNeutron APIประกอบด้วยการสนับสนุนLayer 2 networkingและการจัดการIP address(IPAM) เช่นเดียวกับส่วนขยายสำหรับ Layer 3 ตัว API extension นั้น allow administrators และ tenants ทำการ create "routers" ที่ connect กับ L2 networks หรือที่รู้จักกันว่า "neutron-l3-agent" คือ ช่วยให้สามารถกำหนดเส้นทางระหว่างLayer 2 networks และ gatewaysไปยังnetworkภายนอก OpenStack Networking includes ตัว Plug-ins ที่ Enable การ Interoperability ซึ่งก็คือการที่ทำให้ Interfaces สามารถเข้าใจในการทำงานร่วมกัน Products, open source network technologies หรือ System อื่นๆ ซึ่งก็คือ Routers, switches, virtual switches, software-defined network (SDN) controllers โดยปราศจากการ Restriction ในเรื่องการ Access หรือ Implementation
2) L2 Agents Run บน Hypervisor Host, ติดต่อกับ Neutron Server ผ่าน RPC, ทำหน้าที่แจ้งเตือน Server เมื่อมีการ Add/Remove Devices ใดๆจาก Host, ทำหน้าที่ในการ Wire ตัว Devices ใหม่ให้ทำการ Plugin ไปยังส่วน NW ที่ถูกต้อง และดูแลเรื่อง Security Groups ว่าถูกต้องตาม Rules ใดๆที่กำหนดไหม
3) L3 Agent ปกติแล้วจะอยู่บน Network node แต่ในปัจจุบันในรุ่น Juno ได้มีการออกแบบให้อยู่บน Compute Node แล้วเช่นเดียวกัน, จัดการเรื่อง Floating IPs

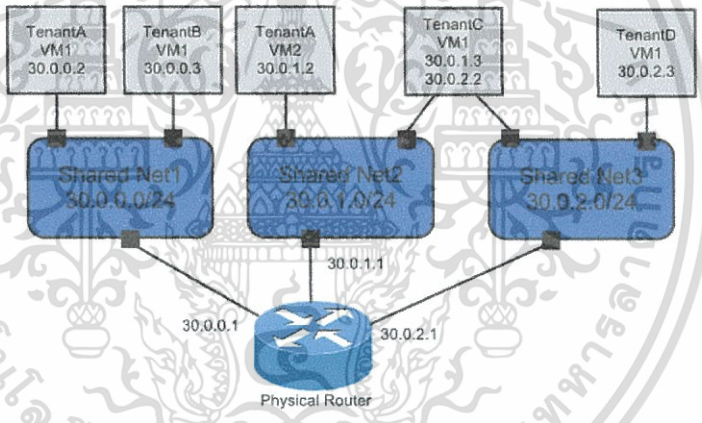
2.3.4.10 รูปแบบของเครือข่ายใน Component Neutron

มีทั้งหมด 4 แบบดังต่อไป นี้ Flat network, VLAN , VXLAN , GRE

1) Flat Network ในรูปแบบเครือข่าย Flat network นั้น Instance ทั้งหมดจะอยู่บนเครือข่ายเดียวกัน ซึ่งแปลว่าสามารถถูกเข้าถึงได้ร่วมกันโดย tenant อื่นได้ และไม่มีการติด tag VLAN หรือการแยกเครือข่ายออกจากกัน ยกตัวอย่างเช่น มี tenant 2 tenant แชร์เครือข่ายร่วมกัน และ IP เครือข่ายก็คือ 30.0.0.0/22 โดย VM จาก tenant A อาจจะถูกกำหนด IP เป็น 30.0.0.2 และ VM จาก tenant B อาจจะถูกกำหนดเป็น 30.0.0.3 ก็ได้ ซึ่งทั้งหมดนี้หมายถึงผู้ใช้ทั้งสองสามารถเห็น Traffic ของกันและกันได้

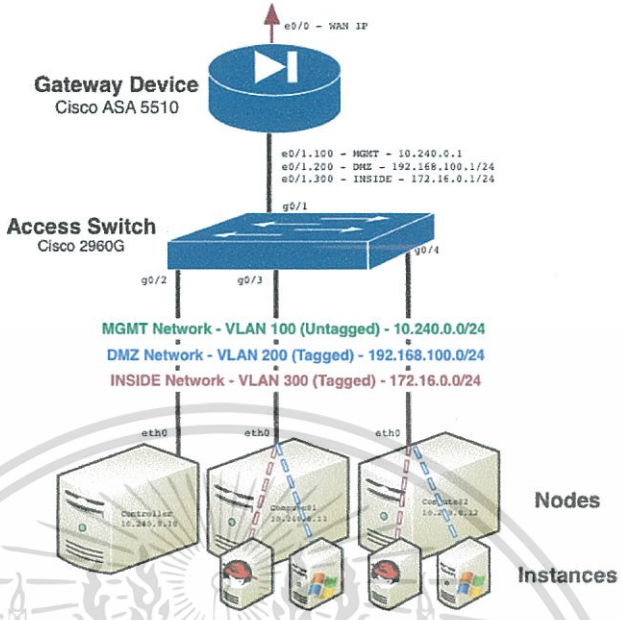


รูป 2.19 Single Flat Network



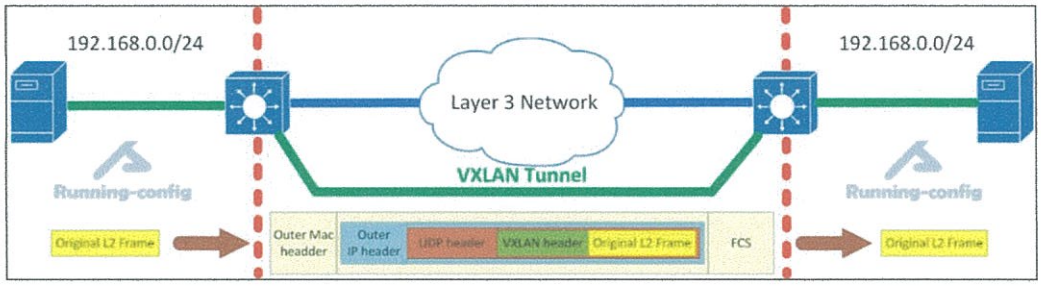
รูป 2.20 Multiple Flat Network

2) VLAN ในเครือข่ายรูปแบบ VLAN นั้น tenants จะถูกแยกออกจากกันเพราะแต่ละ tenant จะถูกกำหนดโดยให้อยู่ต่าง VLAN กัน ซึ่งหมายความว่าในแต่ละ tenant สามารถใช้วง subnet เดียวกันได้ ตัวอย่างเช่น VM1 จาก tenant 1 สามารถกำหนด IP 10.4.128.3 และ VM1 จาก tenant 2 ก็สามารถกำหนด IP 10.4.12.3 ได้เช่นกันโดยไม่มีการ conflict ip ซึ่งกันและกัน ซึ่งจะทำให้ผู้ใช้บริการใช้งานได้สะดวกสบายยิ่งขึ้นเนื่องจากไม่ต้องมากังวลเรื่องที่แต่ละ tenant ต้องการกำหนด subnet เหมือนกันและ IP address เดียวกัน เพราะ VLAN ช่วยทำให้มันแยกออกจากกัน



รูป 2.21 VLAN

3) VXLAN เป็นเทคโนโลยีที่จะช่วยให้สามารถส่งข้อมูลในระดับ Layer 2 ข้ามไประหว่างเครือข่าย Layer 3 ได้ โดยมีหลักการทำงานคือ เมื่อได้รับแพ็คเก็ตที่อุปกรณ์ต้นทางต้องการส่งข้อมูลในระดับ Layer 2 ไปยังอุปกรณ์ปลายทางในฝั่งตรงกันข้ามเข้ามา มันจะทำการห่อหุ้ม (encapsulation) แพ็คเก็ตนั้นด้วยแพ็คเก็ต Layer 3 UDP ก่อนที่จะทำการส่งข้อมูลออกไปบนเครือข่าย Layer 3 ซึ่งจะทำให้แพ็คเก็ตนั้นมี header 2 ชั้นด้วยกัน ชั้นนอกจะเป็น IP และ UDP header ของ VXLAN สำหรับใช้ในการส่งข้อมูลจาก site ต้นทางไปยัง site ปลายทางข้ามระหว่างเครือข่าย Layer 3 ส่วนชั้นที่ 2 จะเป็น header ดั้งเดิมของแพ็คเก็ตนั้น ๆ ที่เครื่องต้นทางต้องการส่งข้อมูลไปยังเครื่องปลายทาง

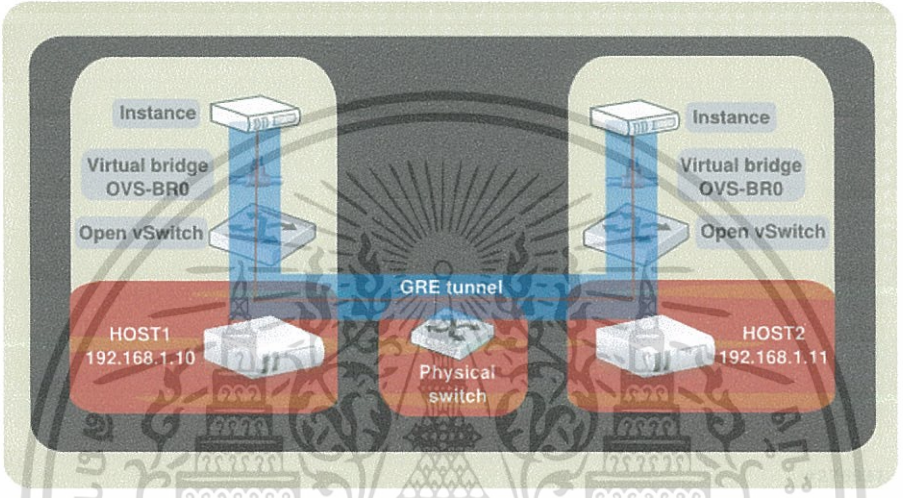


รูป 2.22 VXLAN

2.3.4.11 Generic Routing Encapsulation

GRE คือโปรโตคอลที่ห่อหุ้มข้อมูล Packet เพื่อหาเส้นทางไปยังโปรโตคอลอื่นๆ บน IP networks มาตรฐาน

GRE ถูกพัฒนาขึ้นให้เหมือนกับอุโมงค์ซึ่งหมายถึงการขนส่งโปรโตคอลของ OSI layer 3 บน IP network โดยจุดสำคัญคือ GRE จะสร้างๆ ลักษณะการเชื่อมต่อส่วนตัวแบบ point-to-point เหมือนๆกับรูปแบบของ VPN หน้าที่หลักของ GRE คือให้บริการเชื่อมต่อกันระหว่าง VM ทุกตัวในเครือข่ายของ tenant และทำการแยก VM แต่ละตัวใน แต่ละเครือข่ายของ tenant



รูป 2.23 GRE

ทำไมถึงเลือกใช้ GRE ข้อดีของ GRE

1) อุโมงค์ GRE สามารถห่อโปรโตคอลๆหลายโปรโตคอลบนหนึ่งโปรโตคอล backbone
2) อุโมงค์ GRE ช่วยแก้ปัญหาสำหรับเครือข่ายที่มีจำนวน hops ที่จำกัด
3) อุโมงค์ GRE สามารถ เชื่อมต่อกับ Sub-networks ที่อยู่ต่างคลาส IP ได้
4) อุโมงค์ GRE อนุญาตให้ใช้ VPN ผ่าน WANs ได้
5) อุโมงค์ GRE สามารถรักษาความปลอดภัยและความเป็นส่วนตัวของผู้ใช้งานได้โดย สามารถให้ระบุ Username และ Password ที่ถูกต้องก่อนใช้งาน
6) ปลอดภัยหากข้อมูลถูกโจรกรรมระหว่างถูกส่ง เพราะมีการเข้ารหัสข้อมูลไว้

2.3.5 Swift

2.3.5.1 Swift Architecture

Object Store ของ OpenStack ("Swift") ถูกออกแบบมาเพื่อให้มีการจัดเก็บขนาดใหญ่ของข้อมูลที่สามารถเข้าถึงได้ผ่านทาง APIs ซึ่งแตกต่างจากไฟล์เซิร์ฟเวอร์แบบดั้งเดิมก็มีการกระจายอย่างสมบูรณ์การจัดเก็บหลายสำเนาของแต่ละวัตถุเพื่อให้บรรลุความพร้อมมากขึ้นและขยายขีดความสามารถ Swift ให้การทำงานของผู้ใช้ต่อไปนี้

1) เก็บและเรียก objects (file)
2) ตั้งค่าและปรับเปลี่ยน metadata บน objects (แท็ก)
3) Versions objects

การให้บริการหน้าเว็บแบบคงที่และวัตถุผ่านทาง HTTP ในความเป็นจริงในแผนภาพบล็อกโพสต์นี้มีการทำหน้าที่ให้บริการ Swift ของ Rackspace
Swift architectureมีการกระจายมากที่จะป้องกันไม่ให้จุดเดียวของความล้มเหลวใด ๆ เช่นเดียวกับขนาดในแนวนอน จะรวมถึงองค์ประกอบต่อไปนี้

1) Proxy server (swift-proxy-server) ยอมรับการร้องขอเข้ามาผ่านทาง OpenStack Object APIหรือเพียงแค่ raw HTTP
2) Account servers จัดการบัญชีกำหนดด้วย object storage service
3) Container servers จัดการการทำแผนที่ของ containers (เช่นโฟลเดอร์) ภายใน object store service
4) Object servers จัดการวัตถุที่เกิดขึ้นจริง (เช่นไฟล์) บน storage nodes

2.3.6 Cinder

2.3.6.1 Cinder Architecture

หน่วยบริการจัดเก็บข้อมูล Cinder Block Storage จะถูกรันบนหนึ่งโหนดหรือมากกว่า โดยในแต่ละ Cinder จะใช้ ฐานข้อมูลแบบ SQL ซึ่งจะมีการเชื่อมโยงข้อมูลถึงกันทั้งหมดของ Cinder ภายในระบบ สำหรับการใช้งานเล็กๆนี้เหมือนจะดีที่สุด แต่สำหรับการใช้ขนาดใหญ่และโดยเฉพาะอย่างยิ่งมุ่งเน้นด้านความปลอดภัย Cinder จะมีระบบเคลื่อนย้ายการเก็บข้อมูลหลายๆชนิด

2.3.7 Keystone

2.3.7.1 Keystone Architecture

1) The Services
2) Keystone จัดเป็นกลุ่มการให้บริการภายในหนึ่งปลายทางหรือหลายๆปลายทาง บริการต่างๆเหล่านี้ถูกนำมาใช้ในรูปแบบผสมโดย Frontend ด้วย เช่น การเรียก

รับรองความถูกต้องจะตรวจสอบผู้ใช้ หรือ ข้อมูลประจำตัวโดยถ้าระบบตรวจสอบการยืนยันสิทธิ์สำเร็จถูกต้อง จะสร้างและส่ง token กลับไปยัง Token Service

3) Identity Service ตรวจสอบข้อมูลประจำตัวและรับรองความถูกต้องของข้อมูลผู้ใช้งาน

4) Resource Service ให้ข้อมูลเกี่ยวกับโครงการและโดเมน

5) Assignment Service ให้ข้อมูลเกี่ยวกับบทบาทและการกำหนดบทบาทให้หน่วยงานที่มีการจัดการโดยเอกลักษณ์และการบริการทรัพยากร

6) Token Service ตรวจสอบและจัดการ Token ที่ได้รับการยืนยันความถูกต้องแล้ว

7) Catalog Service ช่วยให้มี Endpoint registry เพื่อการสืบค้น Endpoint

8) Policy Service ช่วยให้มีเครื่องมือ rule-based authorization และช่วยให้มันมี Rule management interface

2.3.7.2 Application Construction

Keystone เป็นการให้บริการแบบ HTTP Frontend เช่นเดียวกับส่วนอื่นๆของ OpenStack ซึ่งจะสำเร็จได้โดยต้องตั้งค่า Python WSGI Interfaces และแอพพลิเคชั่น โดย HTTP Endpoint ของแอพพลิคชั่น จะกำหนดขึ้นจากไปป์ไลน์ของ WSGI middleware เช่นดังรูป

2.3.7.4 Service Back ends

แต่ละบริการสามารถกำหนดค่าให้ใช้แบ็กเอนด์เพื่อให้ Keystone เพื่อให้พอดีกับหลากหลายของสภาพแวดล้อมและความต้องการ แบ็กเอนด์สำหรับแต่ละบริการที่มีการกำหนดไว้ในแฟ้ม keystone.conf พร้อมกับคีย์ที่สำคัญที่อยู่ภายใต้ในแต่ละการบริการ

2.3.7.5 Data Model

Keystone ได้รับการออกแบบขึ้นมาให้คล้อยตามรูปแบบที่หลากหลายของ Backend และสามารถรับชนิดของข้อมูลได้มากขึ้นกว่าเดิมและรู้ว่าจะทำอะไรกับข้อมูลนั้นแล้วส่งไปยัง Backend ข้อมูลชนิดหลักๆ มีดังต่อไปนี้

1) User – มีบัญชีข้อมูลประจำตัว ซึ่งอาจเกี่ยวข้องกับโครงการหรือโดเมน 1 project หรือมากกว่านั้น

2) Group – กลุ่มของ User อาจเกี่ยวข้องกับโครงการหรือโดเมน 1 project หรือมากกว่านั้น

3) Project – หน่วยของการเป็นเจ้าของใน OpenStack ประกอบไปด้วย 1 users หรือมากกว่านั้น

4) Domain - หน่วยของการเป็นเจ้าของใน OpenStack ประกอบไปด้วย users ,groups, และ project

5) Role – ชิ้นส่วนชั้นแรกของ metadata ซึ่งเกี่ยวข้องกับคู่ของ user-project หลายคู่

6) Token – ระบุข้อมูลประจำตัวที่เกี่ยวข้องระหว่าง user หรือ user กับ project

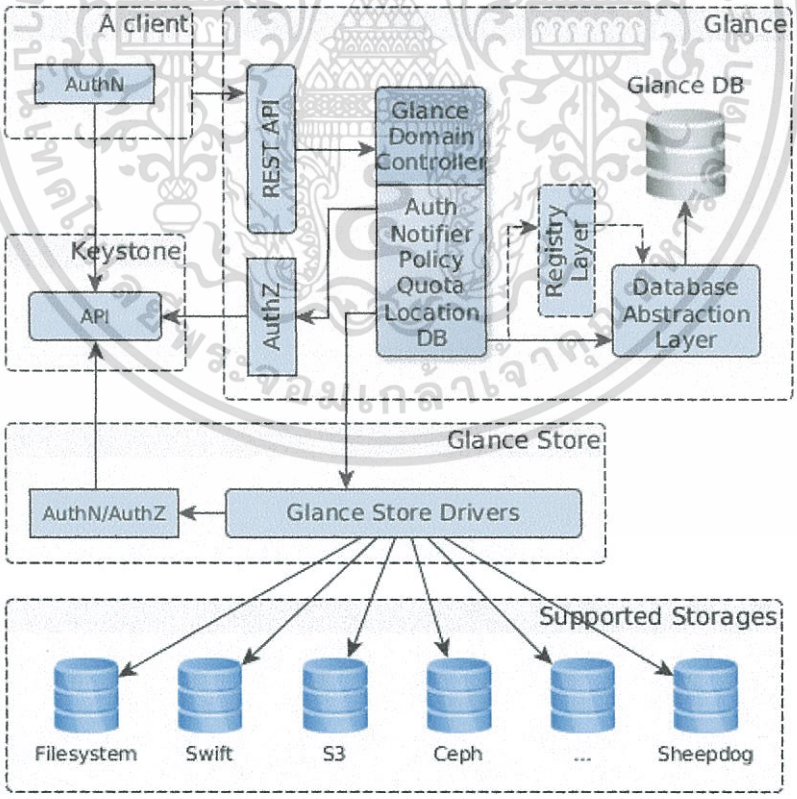
7) Extras – แหล่งเก็บ key-value metadata ที่เกี่ยวข้องกันระหว่างคู่ user-project

8) Rule – อธิบายจำนวนชุดของเงื่อนไขสำหรับการดำเนินการและในปัจจุบันการปรับปรุงการตรวจสอบให้ผู้ใช้สามารถเข้าถึง Cloud OpenStack แบบส่วนตัวหรือสาธารณะโดยใช้สิทธิที่เหมือนกันและ Keystone สามารถกำหนดค่าให้ใช้แบ็กเอนด์หลายตัวพร้อมกันและทำงานร่วมกับ LDAP ง่ายขึ้น

2.3.8 Glance

2.3.8.1 Glance Architecture

Glance มีสถาปัตยกรรมไคลเอ็นต์เซิร์ฟเวอร์และให้ผู้ใช้ REST API ที่ผ่านการร้องขอไปยังเซิร์ฟเวอร์ที่จะดำเนินการ การดำเนินงานภายในเซิร์ฟเวอร์มีการจัดการโดยสรุปควบคุมโดเมนแบ่งออกเป็นชั้น แต่ละชั้นการดำเนินงานของตัวเอง ไฟล์การดำเนินงานทั้งหมดจะดำเนินการโดยใช้ไลบรารี glance_store ซึ่งทำหน้าที่ในการมีปฏิสัมพันธ์กับระบบจัดเก็บภายนอก backend หรือระบบแฟ้มทั่วไป และกำหนดให้รูปแบบฟอร์มอินเตอร์เฟซในการเข้าถึง Glance ใช้ฐานข้อมูลกลางของ SQL-based (Glance DB) ซึ่งใช้ร่วมกันกับทุก Component ในระบบ Openstack

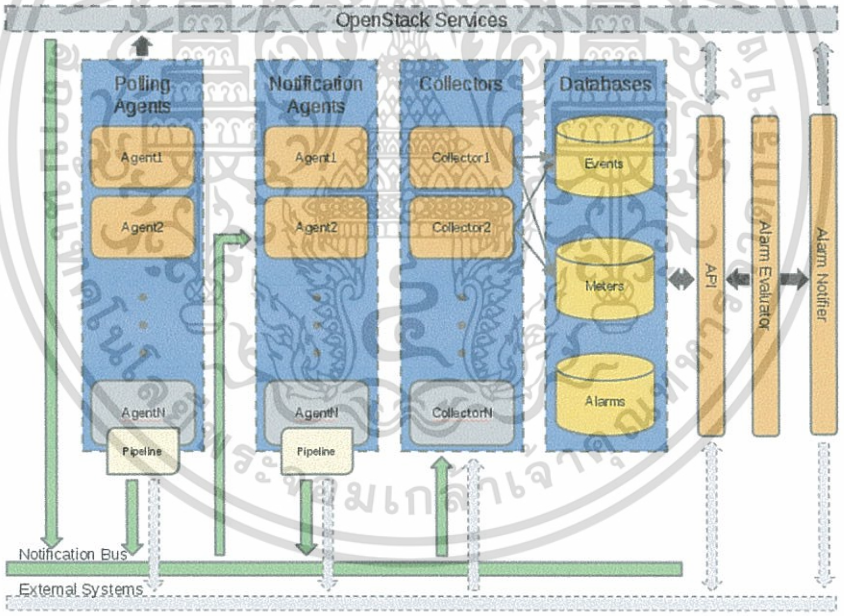


รูป 2.24 Glance Architecture

1) A client – แอพพลิเคชั่นใดๆ ที่กำลังใช้ Glance ในขณะนั้น
2) REST API – exposes Glance functionality ผ่าน REST.
3) Database Abstraction Layer(DAL) – แอพพลิเคชั่นอินเตอร์เฟสที่ให้ การติดต่อสื่อสารระหว่าง Glance และ ฐานข้อมูล
4) Glance Domain Controller – ทำหน้าที่เป็นตัวกลางในการ implement หน้าที่หลักๆของ Glance เช่น การยืนยันตัว การแจ้งเตือน นโยบาย และการเชื่อมต่อฐานข้อมูล
5) Glance Store – จัดการปฏิสัมพันธ์ระหว่าง Glance และที่เก็บข้อมูลในส่วนต่างๆ
6) Registry Layer - ตัวเลือกการจัดชั้นการสื่อสารที่ปลอดภัยระหว่างโดเมนและ DAL โดยใช้บริการที่แยกต่างหาก

2.3.9 Ceilometer

2.3.9.1 Ceilometer Architecture



รูป 2.25 Ceilometer Architecture

แต่ละส่วนบริการของ Ceilometer ถูกออกแบบมาในรูปทิศทางแนวนอน ตัวทำงานหรือโหนดเพิ่มเติมสามารถเพิ่มขึ้นได้ขึ้นอยู่กับ expected load Ceilometer มีส่วนบริการหลักๆอยู่ทั้งหมด 5 ส่วน ดังนี้โดย ข้อมูลจะถูกเก็บสะสมและแจ้งเตือน แยกเป็นอิสระออกจากกัน แต่ยังคงทำงานร่วมกันเป็นระบบที่สมบูรณ์ ได้แก่ Gathering the data(ส่วนรวบรวมข้อมูล), Processing the data(ส่วนประมวลผลข้อมูล), Storing the data(ส่วนการจัดเก็บข้อมูล), Accessing the data(ส่วนการเข้าถึงข้อมูล) และ Evaluating the data(ส่วนการประเมินข้อมูล)รวมทั้งปรับปรุง

ประสิทธิภาพเพิ่มขึ้น และเพิ่มมิเตอร์การวัดสำหรับการใช้งานในส่วนของเครือข่ายเช่น load balancer , firewalls และ VPNs

2.3.10 Heat

2.3.10.1 Heat Architecture

1) Heat ใช้ติดต่อสื่อสารกับ heat-api เพื่อดำเนินการ AWS CloudFormation APIs และนักพัฒนาสามารถใช้ REST API ได้โดยตรง
2) Heat-api ให้บริการ OpenStack-native REST API ซึ่งประมวลผลการร้องขอ API โดยส่งไปยังส่วนของ heat-engine บน RPC
3) Heat-api-cfn ให้บริการ AWS Query API ซึ่งเข้ากันได้กับ AWS CloudFormation และประมวลผลการร้องขอ API โดยส่งไปยังส่วนของ heat-engine บน RPC
4) Heat-engine ทำหน้าที่จัดการการ launch ตัว Templates และแสดง event กลับไปยัง API consumer

ความสามารถใหม่ของ Heat ในเวอร์ชั่น JUNO นั้นสามารถย้อนกลับกระบวนการทำงานได้ง่ายขึ้นหากเกิดความผิดพลาดในการใช้งาน ผู้ดูแลระบบสามารถมอบหมายสิทธิประโยชน์การสร้างทรัพยากรให้กับผู้ใช้ที่ไม่ใช่ผู้ดูแล และปรับปรุงการดำเนินงานของทรัพยากรชนิดใหม่และมีความยืดหยุ่นดีขึ้น

2.4 What is Docker?

Docker เป็น Open platform สำหรับการพัฒนา (Developing), ย้าย (Shipping), ใช้งาน (Running) ตัว Applications Docker ถูกออกแบบมาเพื่อให้เราใช้งาน Develop, Ship, Run Application ของเราได้เร็วขึ้น ด้วย Docker เราสามารถแยก Applications จาก Infrastructure และใช้ Infrastructure เหมือนกับการจัดการ Application Docker ช่วยในการย้าย (Ship) Code/ Run Code/Test Code/Deploy ได้เร็วขึ้น และลด Cycle หรือช่วงเวลาในการเขียน Code กับ Running code ที่เราอาจสูญเสียไปเปล่าๆได้

Docker กระทำสิ่งดังกล่าวโดยการรวม (Combine) ตัว Lightweight container virtualization platform ด้วย Workflows และ tooling ที่ช่วยในการจัดการ (Manage) และ Deploy Applications ของเรา ณ บริเวณ Core, Docker จะให้ช่องทางในการ Run Application (Almost any application) อย่าง Securely isolated (ไม่ยุ่งกันและปลอดภัย) ภายใน Container

Isolation และ Security จะช่วยให้เราสามารถ Run ได้หลาย Containers ได้พร้อมๆกันบน Host ของเรา ด้าน Lightweight nature ของ Containers นั้นจะ Run โดยปราศจาก Extra load ของ Hypervisor ซึ่งก็หมายถึงเราสามารถใช้ Hardware ได้อย่างคุ้มค่ามากยิ่งขึ้น

2.4.1 ประโยชน์ที่จะได้จากการใช้ Container Virtualization

ซึ่งเปรียบเป็น Tool และ Platform ก็คือ

1) ให้ Applications เราไปอยู่ใน Containers ของ Docker
2) สามารถนำ Containers ของ Docker ไปใช้ในอนาคตได้ กล่าวคือสามารถส่งต่อให้ Team พัฒนาหรือทดสอบในอนาคตไปใช้ต่อได้สะดวกยิ่งขึ้น
3) การ Deploy Applications ไปยัง Production environment สามารถทำได้ไม่ว่าจะอยู่ใน Local datacenter หรือ Cloud ก็ตาม

2.4.2 ทำไม Docker?

2.4.2.1 Faster delivery of your applications

Docker ช่วยในเรื่องการประหยัดเวลา กล่าวคือ Developer สามารถ Develop บน Local container ที่มี (Contain) Applications และ Services อยู่แล้วค่อย Integrate ไปยัง Continuous integration (หรือเสมือน Global นั่นเอง) และ Deployment workflow

ต้องการให้ Environment ของเราทำงานได้ดีขึ้น เราต้องสร้าง Standard container format ขึ้นมาเพื่อนการทำงานที่สามารถนำมาประสานงานกันได้อย่างสะดวกยิ่งขึ้น

เช่น Developer เขียน Code Locally และแบ่งปันกับคนอื่นผ่าน Docker เมื่อมั่นใจใน Code ที่พัฒนาแล้วก็นำไป Test, Execute บน Testing environment ต่างๆที่กำหนดแล้วอาศัย Docker images ที่มีไปทำ Production และ Deploy code ออกมา

2.4.2.2 Deploying and scaling more easily

Docker container-based platform นั้นสามารถ Run บน Developer local host, physical, virtual machines ใน Datacenter หรือ Cloud ก็ได้ซึ่งหมายถึงมีความง่ายต่อการ Deploy หรือ Scaling มาก Docker containers run ได้เกือบทุกๆที่เช่น Physical servers, virtual machines, data centers, public/private cloud ง่ายต่อการย้าย Applications จาก Testing environment ไปยัง Cloud หรือย้ายกลับมา

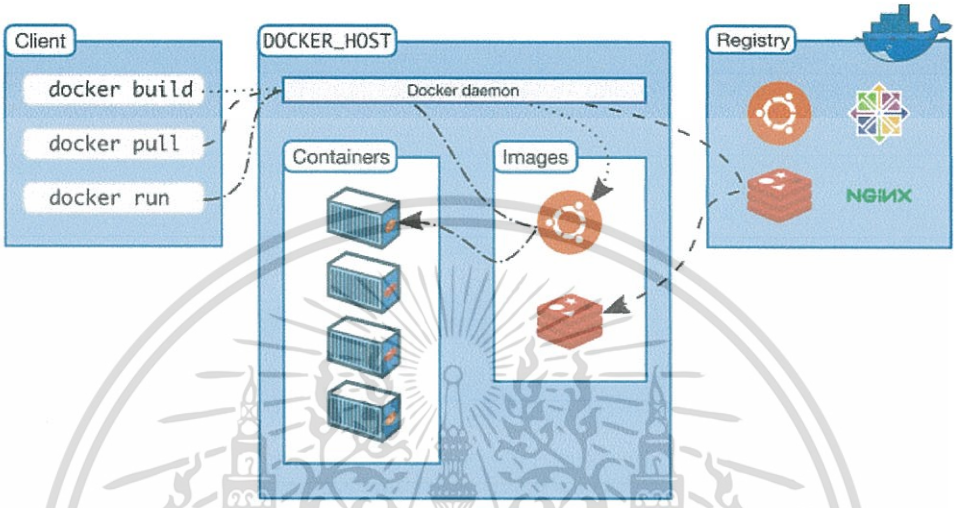
2.4.2.3 Achieving higher density and running more workloads

Docker นั้น Lightweight และมีความเร็ว จึงมีประโยชน์อย่างมากต่อ High density environments เช่น เอื้อต่อกรณีการสร้าง Cloud/PaaS ของเราขึ้นมา หรือ สำหรับ Small Medium deployments ที่เราต้องการใช้ทรัพยากรให้คุ้มที่สุด เป็นต้น

2.4.3 องค์ประกอบของ Container?

1) Docker Open source container virtualization platform.
2) Docker Hub SaaS platform สำหรับในการ Sharing, Managing Docker containers

2.4.4 สถาปัตยกรรมของ Docker?



รูป 2.26 สถาปัตยกรรม Docker

1) Docker นั้นใช้รูปแบบสถาปัตยกรรมแบบ Client-Server ซึ่ง Docker Client นั้นจะสื่อสารกับ Docker Daemon ซึ่งมีหน้าที่ในการ สร้าง ประมวลผล และ จัดสรร Containers
2) ซึ่งทั้ง Docker Client และ Docker Daemon นั้นสามารถติดต่อกันโดยใช้วิธีเชื่อมต่อ Docker Client แล้ว Remote ไปยัง Docker Daemon
3) Docker Client และ Docker daemon นั้นสื่อสารกันผ่าน Sockets หรือผ่าน RESTful API
4) Docker Daemon รันอยู่บน host machine ซึ่งผู้ใช้จะไม่ได้ติดต่อสื่อสารกับ Docker Daemon โดยตรงแต่ผ่านทาง Docker Client
5) Docker Client เป็นระบบ Primary User Interface ของ Docker ซึ่งรับคำสั่งของ User ซึ่งส่งคำสั่งไปไปและส่งกลับผ่าน Docker Daemon

2.4.4.1 Docker Component

1) Docker Images(build) ใช้สำหรับการสร้าง Containers ซึ่ง Docker นั้นให้บริการสร้าง images ใหม่แบบง่ายหรือสามารถอัพเดท images ที่มีอยู่แล้วก็ได้ หรือสามารถนำ images ที่ผู้อื่นสร้างไว้แล้วมาใช้ก้ได้เช่นกัน

2) Docker registries(pull) ทำหน้าที่จัดการ images ซึ่งอาจมาจาก public หรือ private stores ที่มาจากการ upload หรือ download ซึ่ง public stores ของ docker นั้นอยู่ที่ Docker Hub ซึ่งมีให้บริการ images มากมาย
3) Docker run(containers) เปรียบเสมือน Directory ซึ่งเก็บค่าติดตั้งต่างๆสำหรับการรัน Application โดยแต่ละContainer ไม่เกี่ยวข้องกันและถูกสร้างมาจาก Docker image โดย Containers นั้นสามารถรัน,เริ่มต้นการทำงาน,หยุดการทำงาน,เคลื่อนย้าย,ลบทิ้ง

บทที่ 3

การออกแบบ

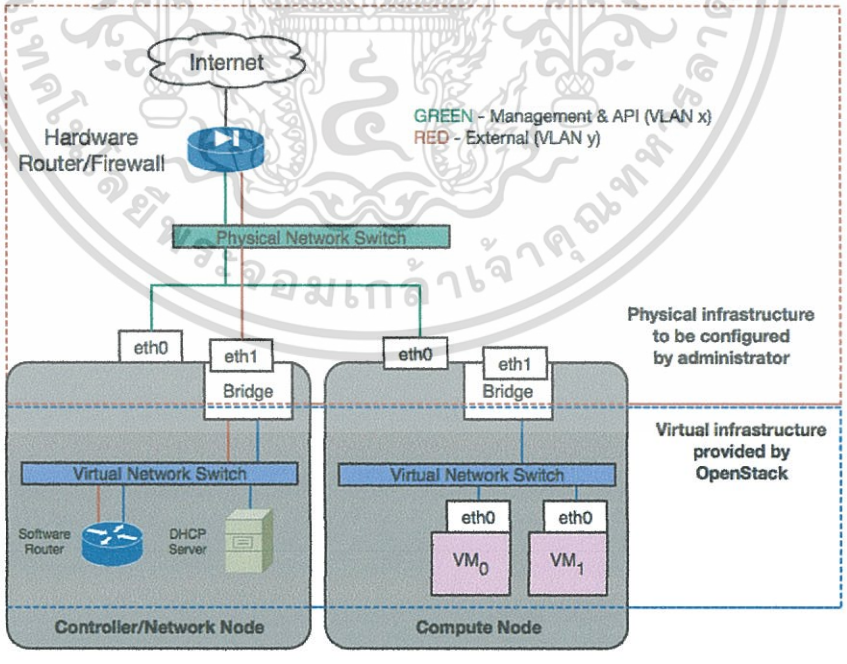
3.1 การจัดเตรียมโครงสร้างพื้นฐานทางกายภาพ (Preparing the physical infrastructure)

โดยการออกแบบมีเป้าหมายในการสร้างสภาพแวดล้อมที่สามารถปรับขนาดได้สูงสำหรับหลายระดับความซ้ำซ้อนทางเครือข่าย หรือมีเป้าหมายที่จะให้ความยืดหยุ่นของเครือข่ายหรือแพลตฟอร์มประมวลผล

OpenStack นั้นเป็นเครือข่ายสามารถให้บริการหลายบทบาทภายในคลาวด์ที่แตกต่างกัน โดยวัตถุประสงค์คือต้องการความปลอดภัย และรองรับฮาร์ดแวร์ต่างๆ

โดยที่ www.openstack.org ให้สถาปัตยกรรมอ้างอิงสำหรับคลาวด์นิวตรอนตามที่ที่เกี่ยวข้องกับการรวมกันของโหนด Controller Node, Network Node และ Compute Node

ก่อนที่จะมีการติดตั้ง OpenStack โครงสร้างพื้นฐานของเครือข่ายทางกายภาพจะต้องกำหนดค่าให้การสนับสนุนเครือข่ายที่จำเป็นสำหรับการดำเนินงานคลาวด์ ในแผนภาพต่อไปนี้ ได้เน้นพื้นที่ของความรับผิดชอบในการดูแลระบบเครือข่าย



รูป 3.1 โครงสร้างพื้นฐานของเครือข่ายทางกายภาพ

โครงสร้างพื้นฐานของเครือข่ายทางกายภาพจะต้องกำหนดค่าเพื่อสนับสนุน OpenStack เครือข่าย ในแผนภาพนี้พื้นที่สีแดงเป็นความรับผิดชอบของผู้ดูแลระบบเครือข่าย ที่อาจรวมถึงความจำเป็นในการกำหนดค่าสวิทช์ทางกายภาพ ไฟร์วอลล์ หรือเราเตอร์ รวมทั้งอินเตอร์เฟซบนเซิร์ฟเวอร์ และพื้นที่สีน้ำเงินจะเป็นส่วนที่จัดการโดย OpenStack ซึ่งจะเป็นส่วนของ Virtual infrastructure

3.1.1 ชนิดของการจราจรเครือข่าย (Type of network traffic)

สถาปัตยกรรมอ้างอิงสำหรับ OpenStack เครือข่ายจะกำหนดอย่างน้อยสี่ประเภทที่แตกต่างกันของเครือข่าย

1) Management Network เครือข่ายการจัดการ (Management Network) ใช้สำหรับการสื่อสารภายในระหว่างโฮสต์ สำหรับการให้บริการ เช่นบริการส่งข้อความ และบริการฐานข้อมูล โดยโฮสต์ทั้งหมดจะสื่อสารกันผ่านทางเครือข่ายนี้ เครือข่ายการจัดการสามารถกำหนดค่าเป็นเครือข่ายแบบเดี่ยวบนอินเตอร์เฟซ หรือแบบรวมกับเครือข่ายอื่น
2) API Network เครือข่าย API จะใช้ในการเปิดเผย OpenStack APIs กับผู้ใช้ของคลาวด์ และบริการภายในคลาวด์ ที่อยู่ปลายทางสำหรับการให้บริการเช่น Keystone, Neutron, Glance, และ Horizon จะถูกจัดหาจากเครือข่าย API มันเป็นเรื่องธรรมดาที่จะกำหนดค่าที่อยู่ IP เดียวบนอินเตอร์เฟซเฉพาะที่จะทำหน้าที่เป็นผู้รับที่อยู่สำหรับการให้บริการต่าง ๆ เช่นเดียวกับการจัดการที่อยู่สำหรับโฮสต์ของตัวเอง
3) External Network เครือข่ายภายนอกช่วยให้เราเตอร์นิวตรอนที่มีการเข้าถึงเครือข่าย เมื่อเราเตอร์ได้รับการกำหนดค่าเครือข่ายนี้จะกลายเป็นที่มาของ floating IP addresses สำหรับอินสแตนซ์ และ load balancer VIPs ที่อยู่ IP ในเครือข่ายนี้ควรจะสามารถเข้าถึงได้โดยลูกค้าบนอินเทอร์เน็ตใดๆ

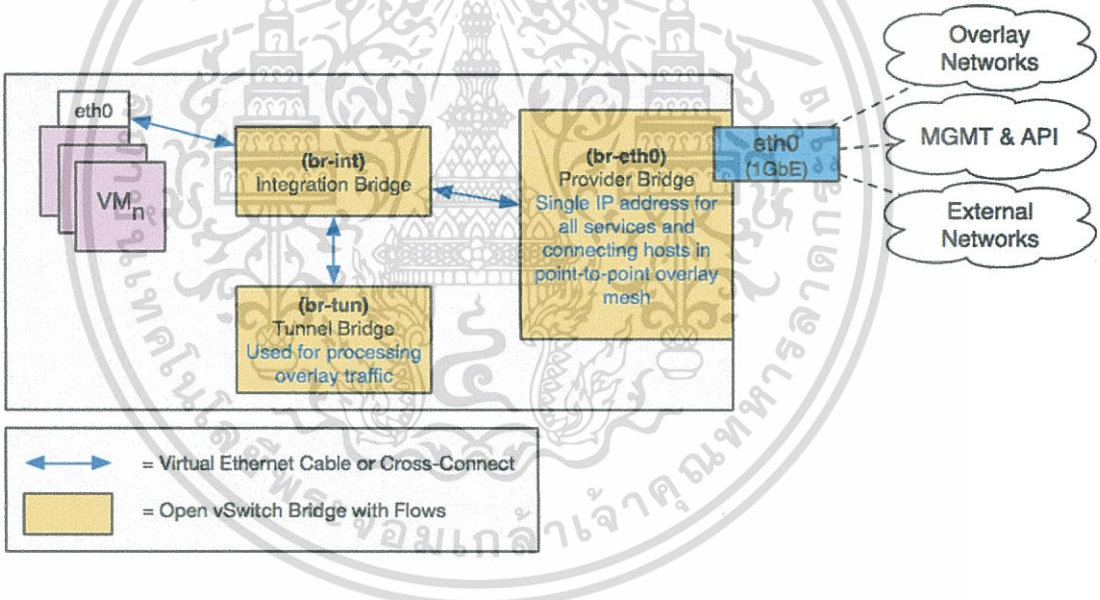
4) Guest Network เครือข่ายเกสต์เป็นเครือข่ายเฉพาะเพื่อการจราจรอินสแตนซ์ ตัวเลือกสำหรับเครือข่ายเกสต์ไปรวมถึงเครือข่ายท้องถิ่นที่ถูกจำกัดไปยังโหนดเฉพาะ การใช้เครือข่ายซ้อนทับเสมือนกระทำด้วยการห่อหุ้ม GRE

3.2 การเชื่อมต่อเซิฟเวอร์ทางกายภาพ (Physical server connections)

จำนวนของอินเตอร์เฟซที่จำเป็นต่อโฮสต์จะขึ้นอยู่กับชนิดของคลาวด์ถูกสร้างขึ้น การรักษาความปลอดภัย และความต้องการประสิทธิภาพการทำงานขององค์กร

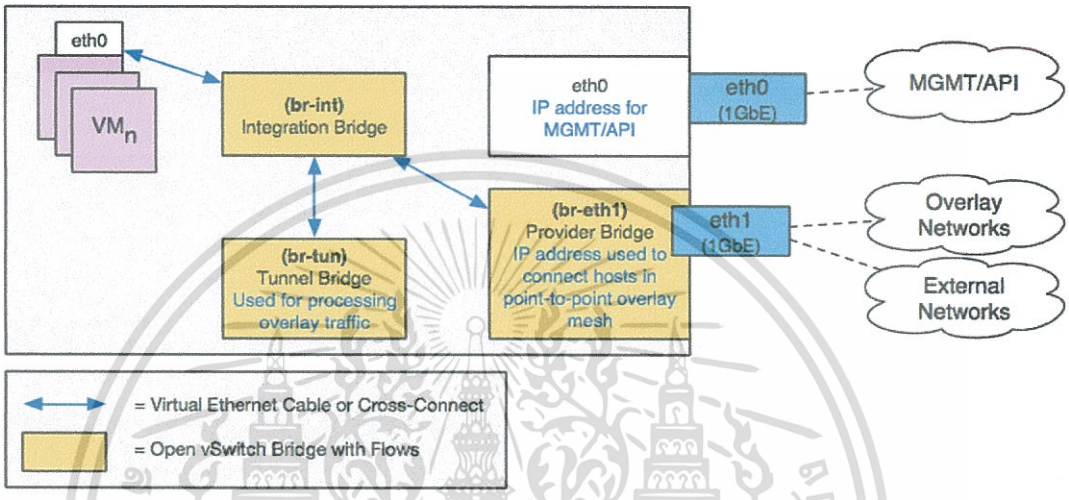
3.2.1 Interface

1) อินเตอร์เฟซเดียว (single interface) ในรูป 3.2 นี้บริการ OpenStack ทั้งหมดและการจราจรการจัดการจะติดต่อทางกายภาพเช่นเดียวกับการจราจรเกสต์



รูป 3.2 โครงสร้าง single interface

2) อินเตอร์เฟสแบบหลายอินเตอร์เฟส (Multiple interfaces) ในแผนรูป 3.3 แสดงการติดต่อทางกายภาพโดยเฉพาะจัดการกับจราจรกำกับการไปมาของอินสแตนซ์หรือบริการอื่น ๆ ของเครือข่าย OpenStack เช่น LBaaS และ FWaaS ขณะที่อินเตอร์เฟสอื่นจัดการกับ OpenStack API และการจัดการจราจร

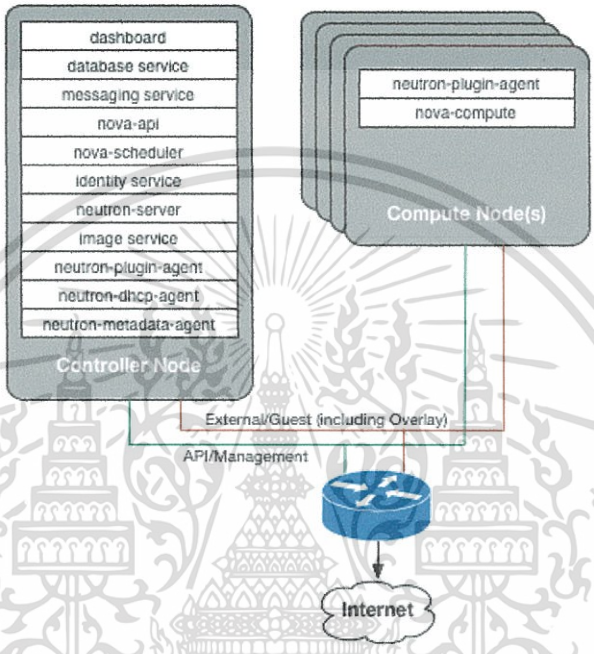


รูป 3.3 โครงสร้าง Multiple interfaces

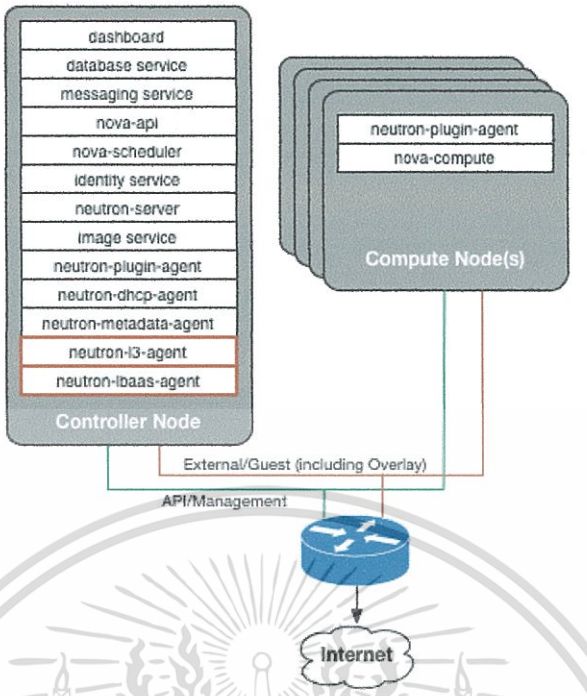
3.2.2 Node

1) A single controller with one or more compute node ในสภาพแวดล้อมที่ประกอบด้วยคอนโทรลเลอร์เดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลคอนโทรลเลอร์อาจจะจัดการกับทุกบริการเครือข่ายและการบริการอื่น ๆ ของเครือข่ายOpenStack ในขณะที่โหนดประมวลผลให้ทรัพยากรการประมวลผล แผนภาพต่อไปนี้โหนดประมวลผลจะจัดการ OpenStack management และnetworking services โดยไม่ใช้ layer 3 agent แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes ในรูป 3.4 สะท้อนให้เห็นถึงการใช้ควบคุมเดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลที่นิวตรอนให้เพียงเลเยอร์ 2 เชื่อมต่อกับอินสแตนซ์ เราเตอร์ภายนอกเป็นสิ่งจำเป็นในการจัดการเส้นทางระหว่างกลุ่มเครือข่าย และในรูป 3.5 โหนดประมวลผลจะจัดการ OpenStack management และ networking services โดยเพิ่ม layer 3 agent แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes และ สะท้อนให้เห็นถึงการใช้คอนโทรลเลอร์โหนดเดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลใน

การกำหนดค่าเครือข่ายที่ใช้ Neutron L3 agent ส่วนเราเตอร์ซอฟแวร์ที่สร้างขึ้นด้วยนิวตรอนอยู่ในโหนดคอนโทรลเลอร์และจัดการเส้นทางระหว่างเครือข่ายที่เชื่อมต่อผู้เช่า

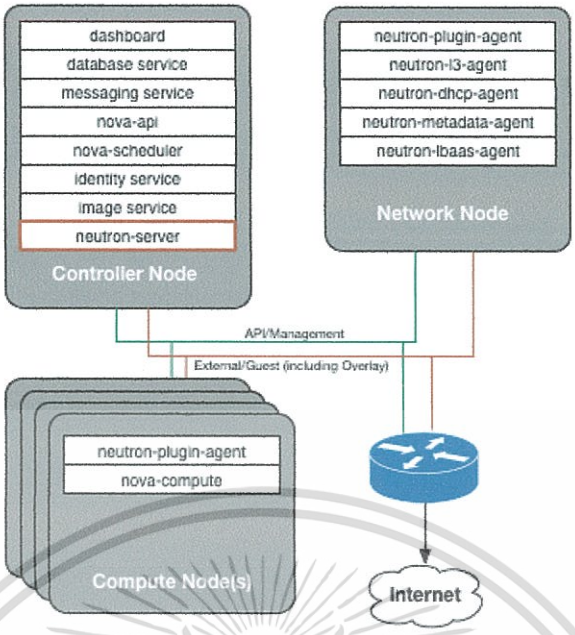


รูป 3.4 โครงสร้างที่มี Controller Node ต่อกับ หลายๆCompute Node

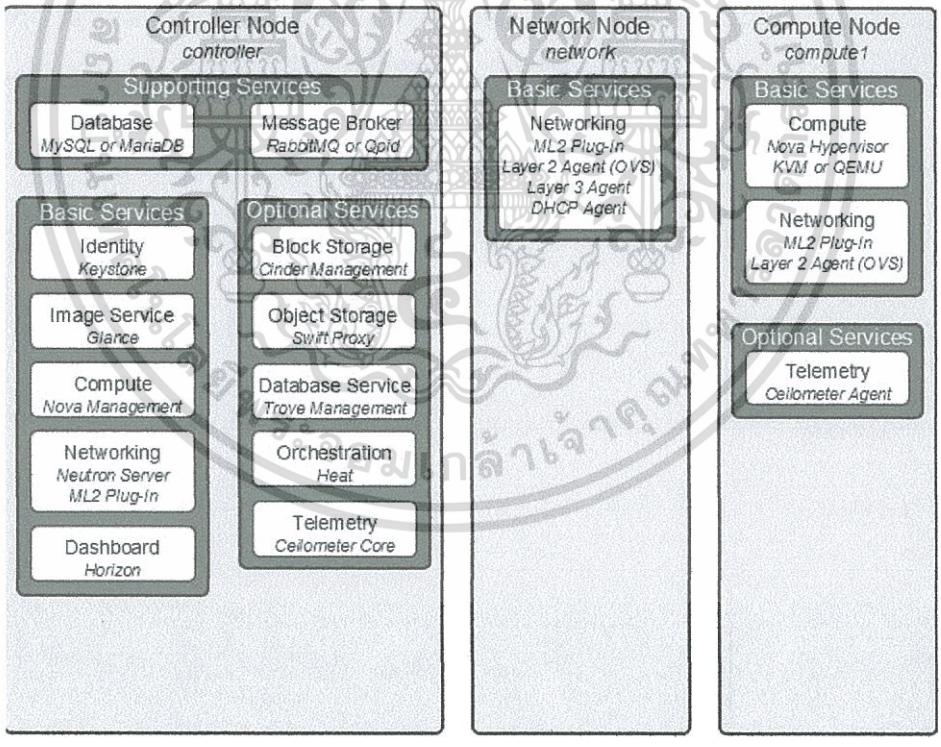


รูป 3.5 โครงสร้างที่มี Controller Node

2) A single controller plus network node with one or more compute nodes โหนดเครือข่ายเป็นสิ่งหนึ่งที่มีความมุ่งมั่นที่จะจัดการส่วนใหญ่หรือทั้งหมด ของเครือข่าย OpenStack บริการเครือข่ายรวมทั้ง L3 agent, DHCP agent, metadata agent และอื่นๆ การใช้โหนดเครือข่ายเฉพาะให้การรักษาความปลอดภัยและความยืดหยุ่นเพิ่มขึ้น ใน รูป 3.6 แสดงให้เห็นถึงโหนดเครือข่ายบริการโฮสติ้งเครือข่าย ของเครือข่ายOpenStack ทั้งหมดรวมทั้ง Neutron L3 agent นิวตรอน API มีการติดตั้งบนโหนดคอนโทรลเลอร์ แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes รูป 3.6 สะท้อนให้เห็นถึงการใช้โหนดเครือข่ายเฉพาะในการกำหนดค่าเครือข่ายที่ใช้ Neutron L3 agent ส่วนเราเตอร์ซอฟแวร์ที่สร้างขึ้นด้วยนิวตรอนอยู่ในโหนดเครือข่าย และการจัดการเส้นทางระหว่างเครือข่ายที่เชื่อมต่อผู้เช่าบริการ API นิวตรอนเซิร์ฟเวอร์ยังคงอยู่ในโหนดคอนโทรลเลอร์



รูป 3.6 โครงสร้างที่มี Controller Node



รูป 3.7 ส่วนประกอบของแต่ละ Node

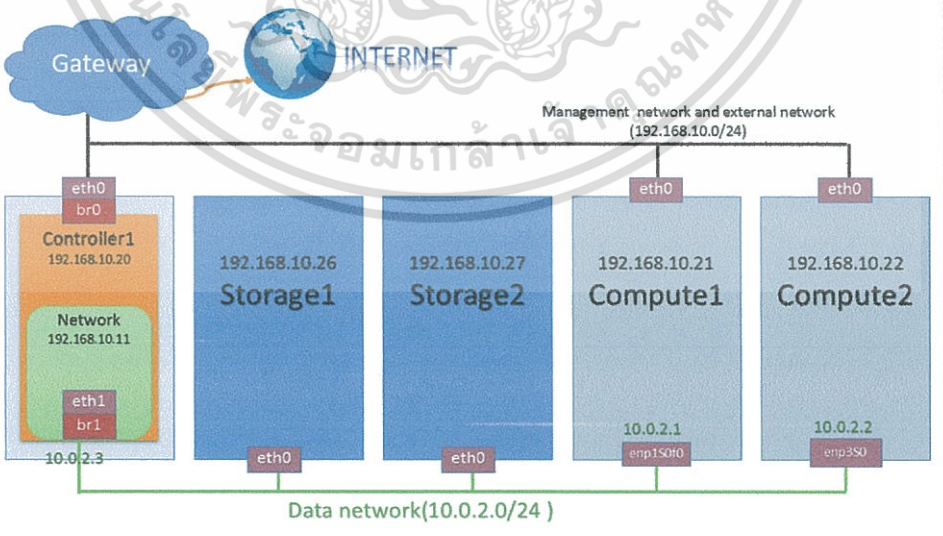
Controller node จะประมวลผลในส่วนของ Identity service, Image Service, ส่วนการบริหารจัดการของ Compute และ Networking, Networking plug-in และ dashboard นอกจากนี้ยังมีการสนับสนุนการบริการเช่น database, message broker, and Network Time Protocol (NTP) สามารถเลือกให้ Controller node นี้ทำงานในส่วนของ Block Storage, Object Storage, Database Service, Orchestration และ Telemetry ส่วนประกอบเหล่านี้มีคุณสมบัติเพิ่มเติมสำหรับสภาพแวดล้อมของคุณ

Network node จะรันในส่วนของ Networking plug-in, layer 2 agent, และ several layer 3 agents ที่จัดและใช้งานเครือข่ายของผู้เช่า Layer 2 services รวมถึงการการจัดเตรียมของเครือข่ายเสมือน และ tunnels ส่วน Layer 3 services รวมถึง routing, NAT และ DHCP โหนดนี้ยังจัดการภายนอก (อินเทอร์เน็ต) สำหรับการเชื่อมต่อเครื่องเสมือนผู้เช่าหรืออินสแตนซ์

Compute node จะรัน hypervisor portion ของ Compute ที่ดำเนินการเครื่องเสมือนของผู้เช่าหรืออินสแตนซ์ โดยค่าเริ่มต้น Compute ใช้ KVM เป็นไฮเปอร์ไวเซอร์ นอกจากนี้ Compute node ยังรัน Networking plug-in และ layer 2 agent ที่ทำงานเครือข่ายผู้เช่าและดำเนินการตามกลุ่มรักษาความปลอดภัยสามารถเลือกให้ Compu'te node ทำงานเป็นเป็น Telemetry agent. ส่วนประกอบเหล่านี้มีคุณสมบัติเพิ่มเติมสำหรับสภาพแวดล้อมของคุณ

3.3 OpenStack Cloud Network

3.3.1 เครือข่ายทางกายภาพ (Physical network)

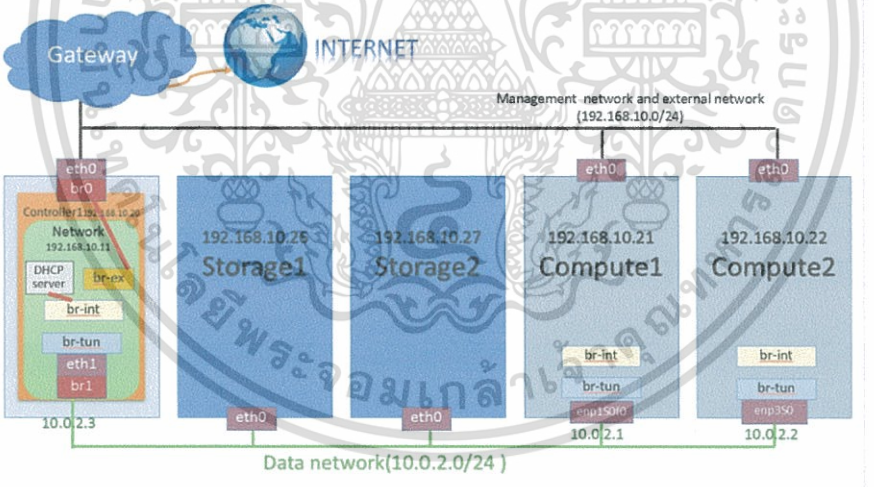


รูป 3.8 โครงสร้างทั้งหมดของPhysical network

จากรูป 3.8 เป็นเครือข่ายทางกายภาพที่ใช้ในการทดลอง

- ประกอบด้วย Controller 1 node, Storage 2 node, และ Compute 2 nodes
- Controller เป็น virtual machine (192.168.10.20) ที่มี br0 ซึ่งเชื่อมต่อกับ eth0 ของเครื่องทางกายภาพ (192.168.10.11) โดยเครื่องทางกายภาพมี 1 interface เชื่อมต่อกับ Management network (เพื่อจัดการ node ต่างๆในระบบ) และ external network (เพื่อใช้รีโมทมาควบคุมระบบ)
- Storage (192.168.10.26, 192.168.27) มีตัวละ 1 interface เชื่อมต่อเข้ากับ Management network
- Compute มี 2 interfaces โดย interface แรก (192.168.10.21 , 192.168.10.22) เพื่อเชื่อมต่อ Management network ส่วนอีก interface (10.0.2.1 , 10.0.2.2) เพื่อเชื่อมต่อกับ Data network
- Network มี 2 interfaces โดย interface แรก (192.168.10.11) เพื่อเชื่อมต่อ Management network และ รับส่งข้อมูลจากอินเตอร์เน็ตส่วนอีก interface (10.0.2.3) เพื่อเชื่อมต่อกับ Data network เพื่อรับส่งข้อมูลกับ Compute nodes

3.3.2 Neutron network with GRE



รูป 3.9 โครงสร้างทั้งหมดของPhysical network GRE

3.4 Heat and Docker

3.4.1 Heat

ใช้ VM 1 ตัว (Ubuntu) ในการรันเว็บงานที่ต้องการขึ้นมา ซึ่งภายใน Ubuntu VM นั้นจะประกอบไปด้วยการติดตั้ง 2 อย่างดังนี้

1) 1.Web server - เพื่อให้สามารถรันเว็บแอพพลิเคชั่น

2) 2.Database – เพื่อให้สามารถเชื่อมต่อกับฐานข้อมูลของเว็บแอพพลิเคชั่น

3.4.2 Docker

ประกอบไปด้วย 2 container ดังนี้

1) 1.Web server – เพื่อให้สามารถรันเว็บแอพพลิเคชั่นและใช้ Framework ของ laravel
2) MySQL Database – เพื่อให้สามารถเชื่อมต่อกับฐานข้อมูลของเว็บแอพพลิเคชั่น และติดตั้ง phpmyadmin เพื่อสำหรับเข้าถึงการใช้งาน MySQL Database

บทที่ 4

การทดลองและผลการทดลอง

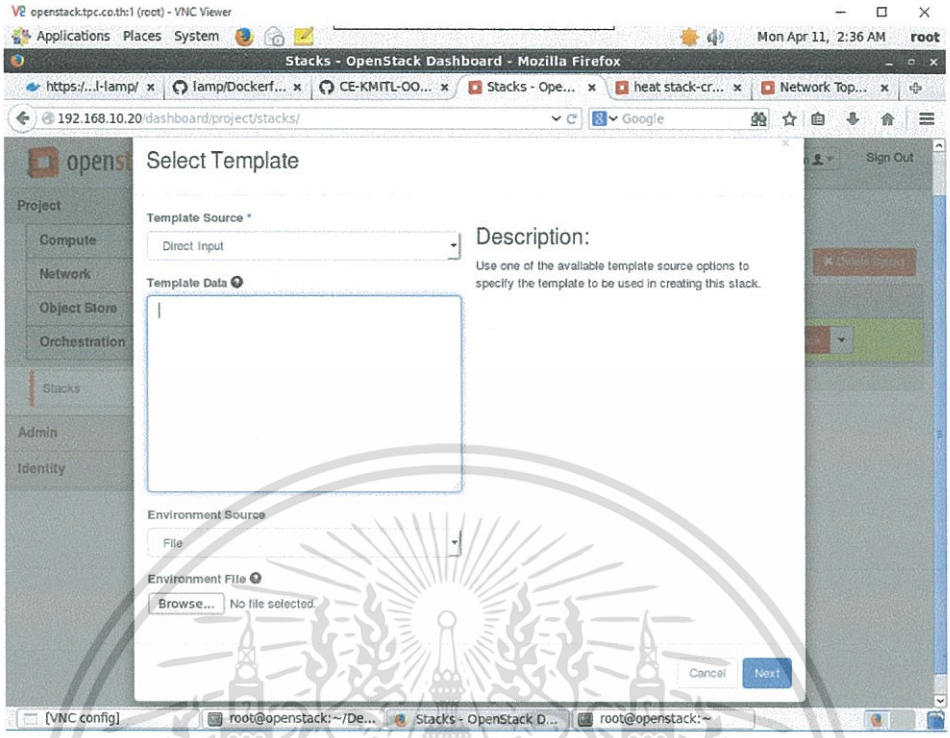
4.1 การทดลอง

ทดลองเปิดใช้งานเว็บแอพพลิเคชั่นของเราโดยอาศัยเทคโนโลยีแบบไหนถึงจะง่ายกว่า เร็วกว่า ประหยัดทรัพยากรมากกว่ากัน

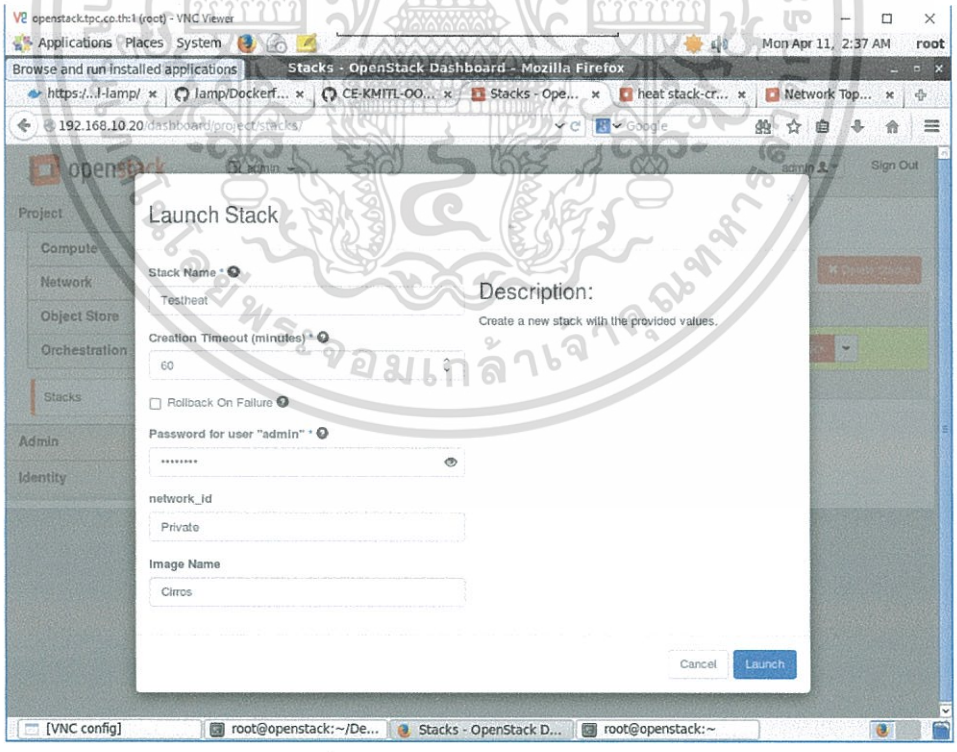
1) เปิดใช้งานแอพพลิเคชั่นด้วยวิธีการอาศัยเทคโนโลยี Virtualization on Hardware(Heat Template OpenStack) โดยในรูป 4.1 คือ Template ที่ใช้สำหรับการสร้าง Multiple VMs และจากรูป 4.2 คือหน้าต่างสำหรับการนำ Template ป้อนเข้าสู่ Component Heat เพื่อทำการสร้าง Multiple VMs ตาม Specification ที่ User กำหนดไว้ใน Template ส่วนในรูป 4.3 หาก Template มี Format ที่ถูกต้อง จะเข้าสู่หน้าต่างที่ให้ User ป้อนค่าต่างๆที่ User ต้องการใช้ในการสร้าง Multiple VMs ส่วนของรูป 4.5 เมื่อป้อนข้อมูลเสร็จสิ้นและเริ่มทำการ Launch Stack หรือเริ่มการรัน Heat ระบบจะเริ่มทำการประมวลผลและสร้าง Multiples VMs และท้ายสุดในส่วนของรูป 4.5 ระบบทำการจำลอง Multiple VMs ขึ้นมาตามที่ Heat template ระบุไว้

```
heat_template_version: 2013-05-23
parameters:
  image_name:
    type: string
    label: Image Name
    default: Cirros
  network_id:
    type: string
    default: 80bcc12b-b546-47ba-8a24-0c0e278e57dd
resources:
  my_instance1:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name }
      flavor: m1.small
      networks:
        - network : { get_param : network_id }
  my_instance2:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name }
      flavor: m1.small
      networks:
        - network : { get_param : network_id }
  my_instance3:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name }
      flavor: m1.small
      networks:
        - network : { get_param : network_id }
```

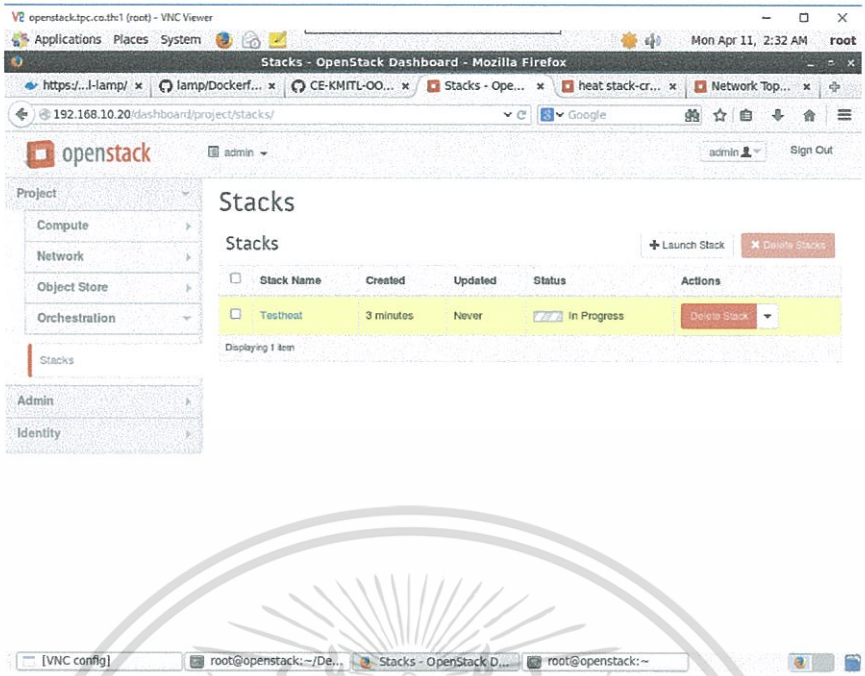
รูป 4.1 Heat template



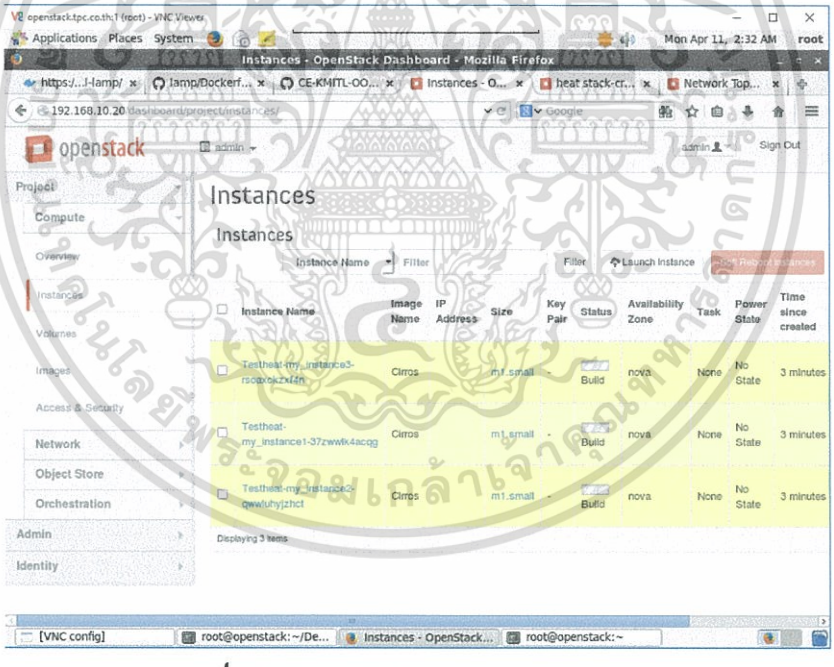
รูป 4.2 Import Template เข้าสู่ Heat



รูป 4.3Input heat data

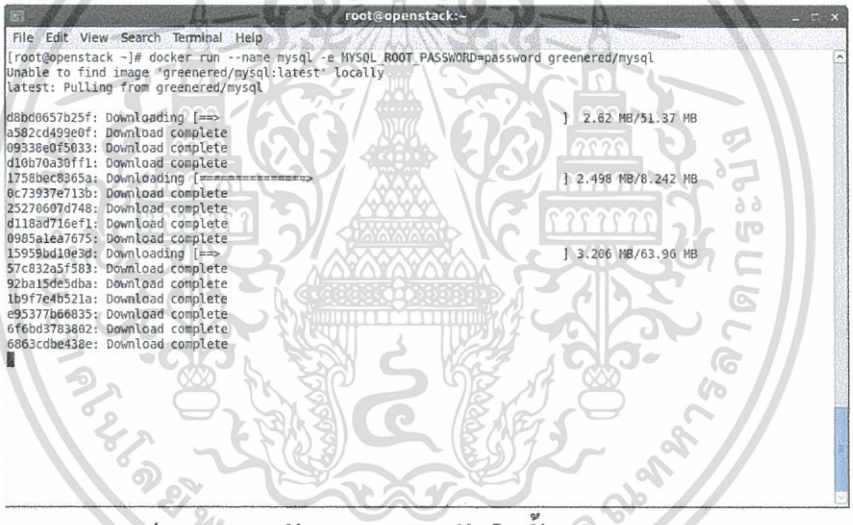


รูป 4.4 Heat created



รูป 4.5 Multiple VMs created

2) การเปิดใช้งานแอพพลิเคชั่นด้วยวิธีการอาศัยเทคโนโลยี Virtualization of Software(Docker) โดยในรูป 4.6 คือทำการติดตั้งระบบฐานข้อมูล mysql ลงใน Container และรูป 4.7 เพื่อทำการติดตั้งระบบ phpmyadmin เพื่อสำหรับ access เข้าถึงการใช้ฐานข้อมูล mysql โดยใช้คำสั่ง link เพื่อเชื่อมต่อข้อมูลระหว่าง 2 Container เข้าด้วยกัน ส่วนของรูป 4.8 นั้นเพื่อทำการติดตั้งระบบ webserver ในการเปิดหน้า web และเมื่อรันเสร็จสิ้น เมื่อลองเข้า localhost ภายใน web browser จะพบข้อความ Forbidden เนื่องจากยังไม่มีไฟล์ Web ใดๆภายใน Documentroot และท้ายสุดส่วนของรูป 4.9 จะรัน Image สำหรับติดตั้ง php:5.6 apache webserver แบบเชื่อมข้อมูลครั้งนี้จะทำการเชื่อมกับข้อมูลเว็บภายในเครื่องด้วยโดยใช้คำสั่ง –v ตำแหน่งไฟล์ในเครื่อง:ตำแหน่งไฟล์ในคอนเทนเนอร์ เมื่อเปิด localhost ภายใน browser ครั้งนี้จะพบกับหน้าเว็บของ php เป็นอันเสร็จสิ้นการติดตั้ง webserver



รูป 4.6 ทำการรัน image สำหรับติดตั้ง mysql

```
root@openstack:~
File Edit View Search Terminal Help
[root@openstack ~]# docker run --name phpmyadmin --link mysql:mysql -d greenered/phpmyadmin
Unable to find image 'greenered/phpmyadmin:latest' locally
latest: Pulling from greenered/phpmyadmin

1348eb573a47: Pull complete
1b384c584f7b: Downloading [=========================================>          ] 4.945 MB/5.893 MB
4956c749a4a9: Downloading [==========>                                         ] 2.065 MB/9.951 MB
67f5f3215d8e: Download complete
8741999fb061: Download complete
cc8014c7f007: Download complete
69e2365d26f5: Download complete
b239d51e09ff: Download complete
ad29423daf09: Download complete
```

รูป 4.7 ทำการรัน image สำหรับติดตั้ง phpmyadmin

```
$ sudo docker run --name php5.6 -d -p 80:80 php:5.6-apache
Unable to find image 'php:5.6-apache' locally
5.6-apache: Pulling from library/php
7268d8f794c4: Pull complete
a3ed95caeb02: Pull complete
38331772e700: Pull complete
74507bbf90f9: Pull complete
c6734ca38ed8: Pull complete
616f76e75b9d: Pull complete
763f79680cbb: Pull complete
e70b2d142af2: Pull complete
62012af41161: Pull complete
33a120b6dfa1: Pull complete
ea474957253d: Pull complete
757eabb832b4: Pull complete
286426d94368: Pull complete
cde52c0a5f98: Pull complete
```

รูป 4.8 ทำการรัน image สำหรับติดตั้ง php:5.6 apache webserver

```
mkdir /home/`whoami`/Site
sudo docker rm php5.6
sudo docker run --name php5.6 -d -v /home/`whoami`/Site/:/var/www/html/  -p 80:80 php:5.6-apache
```

รูป 4.9 ทำการรัน image

4.2 ผลการทดลอง

จากการทดลองผลสรุปได้ว่า การเปิดใช้งานแอพพลิเคชั่นด้วยวิธีการอาศัยเทคโนโลยี Virtualization of Software (Docker) นั้นมีความรวดเร็วกว่า ง่ายกว่า และประหยัดทรัพยากรมากกว่า เนื่องจาก Docker นั้นมีความเบาและเร็วกว่า จำลอง Environment ในการรัน Web Application โดยต้องการแค่ Computer 1 เครื่องที่มีการติดตั้งโปรแกรม Docker ในขณะที่เปิดใช้

งานแอพพลิเคชั่นด้วยวิธีการอาศัยเทคโนโลยี Virtualization on Hardware(Heat template OPENSTACK) นั้นจะต้องอาศัยเครื่องคอมพิวเตอร์ Physical ถึง 5 เครื่อง ประกอบไปด้วย Controller 1 เครื่อง Compute 2 เครื่อง และ Storage 2 เครื่อง

บทที่ 5
บทสรุปและข้อแนะนำ

5.1 สรุปและวิจารณ์

5.1.1 OpenStack Storage (Glance, Cinder, Swift) with GlusterFS

1) Object Storage (Swift) Swift เป็นโมดูลที่ใช้เก็บไฟล์ที่ Instance สามารถเรียกใช้ได้ผ่าน API โมดูลนี้จะเก็บไฟล์แบบ Redundancy (ซ้ำซ้อน) นั่นคือสร้างไฟล์ไว้หลายก็อปปี้และกระจายไฟล์หลายๆ ก็อปปี้นี้ไปยังฮาร์ดดิสก์หลายตัวที่กระจายกันไปอยู่หลายเครื่อง การทำ Redundancy นี้เกิดขึ้นอัตโนมัติ นั่นคือ instance เขียนไฟล์ผ่าน API เท่านั้นส่วนการสร้างก็อปปี้และการกระจายไฟล์นั้นระบบ Swift จะจัดการให้เอง การบริหารไฟล์ภายในนั้นเป็นแบบ Object Storage คือเก็บเป็นไฟล์ไปโดยไม่มีการแบ่งส่วนย่อยเป็นไดเรกทอรี โมดูล Swift นี้เทียบเท่าได้กับบริการS3 ของ Amazon
2) Block Storage (Cinder) Cinder นั้นเป็น Block Storage หรือฮาร์ดดิสก์เสมือนนั่นเอง ผู้ใช้สามารถสร้างฮาร์ดดิสก์เสมือนขึ้นมาและนำไปเชื่อมต่อกับ instance ใดที่กำลังทำงานอยู่ก็ได้ Cinder นั้นสามารถสร้างฮาร์ดดิสก์เสมือนจาก Filesystem จริงหลายรูปแบบ เช่นจากLVM, NFS, หรือ GlusterFS เป็นต้น นอกจากนี้ Cinder ยังสามารถทำ Snapshot หรือ ภาพสถานะปัจจุบันของฮาร์ดดิสก์ และเก็บไว้เป็นแบ็คอัพบน Swift ได้ด้วย โมดูล Cinder นี้นั้นเทียบเท่ากับบริการ EBS ของ Amazon AWS
3) Image Service (Glance) Glance เป็นโมดูลที่ใช้จัดการ Image ของเครื่องเสมือน ใช้ในการสร้างและเรียกใช้ Image นั่นเอง การใช้ Image นั้นทำให้ผู้ใช้สามารถสร้างเครื่องเสมือนที่มีคุณสมบัติเหมือนๆ กันได้โดยง่าย เช่นมีระบบปฏิบัติการเดียวกันหรือลงโปรแกรมเวอร์ชั่นเดียวกันเป็นต้นการทำงานเบื้องหลังนั้น image จะถูกเก็บไว้ใน Swift และถูกส่งไปให้ Nova เมื่อมีการเรียกใช้

จะเห็นได้ว่าทั้ง 3 บริการของ OpenStack นั้นเกี่ยวข้องกับการเก็บข้อมูลทั้งแบบ Object Storage (Swift และ Glance) และ Block Storage (Cinder) ดังนั้นการนำ GlusterFS มาใช้งานร่วมกับทั้ง 3บริการ จะช่วยให้สามารถจัดการบริการเกี่ยวกับ Storage ทำให้มีความสามารถมากขึ้น โดยสามารถทำให้การเก็บข้อมูลเป็นแบบไดนามิก คือสามารถเพิ่มพื้นที่การเก็บข้อมูลได้เรื่อยๆ สามารถปรับเปลี่ยนเครื่อง Storage ได้โดยที่ระบบยังออนไลน์อยู่ และยังทำให้ระบบมีการสำรองข้อมูลได้อีกด้วย

5.1.2 OpenStack Networking (Neutron)

OpenStack Networking เป็นแบบ pluggable สามารถปรับขนาดได้และใช้ API ในการขับเคลื่อนระบบสำหรับการจัดการเครือข่ายและ IP Address เช่นเดียวกับด้านอื่นๆ ของระบบคลาวด์ซึ่งมันสามารถนำมาใช้โดยผู้ดูแลระบบ

Software Defined Networking (SDN) ถูกนำมาแก้ไขข้อบกพร่องของนิวตรอน SDN เป็นเทคโนโลยีเครือข่ายที่ช่วยให้ส่วนกลางสามารถโปรแกรม control plane เพื่อจัดการ data plane ดังนั้นทำให้ตัวดำเนินการเครือข่าย และผู้ให้บริการ สามารถควบคุม และจัดการจัดการทรัพยากรเสมือนจริงของตัวเอง และเครือข่าย SDN เป็นรูปแบบเครือข่ายที่ช่วยให้การเปิดสื่อสาร API ระหว่างฮาร์ดแวร์กับระบบปฏิบัติการ และระหว่างองค์ประกอบของเครือข่าย (ทางกายภาพและเสมือนจริง) กับระบบปฏิบัติการ SDN ยังเป็นผู้รับผิดชอบในการบริหารจัดการการเปลี่ยนแปลงไปยังเครือข่ายและการถ่ายโอนการเปลี่ยนแปลงเหล่านั้นทั้งฮาร์ดแวร์เครือข่ายและเครือข่าย (ทางกายภาพและเสมือนจริง) การรวม SDN เข้ากับ Neutron โดยใช้การปลั๊กอิน ให้ระบบมีการจัดการไปอยู่ที่ส่วนกลาง และยังอำนวยความสะดวกในโปรแกรมเครือข่ายของเครือข่าย OpenStack โดยใช้ API

Software Defined Networking ทำให้สามารถสร้างเครือข่ายเสมือนที่เชื่อมต่อ Instance ต่างๆ เข้าด้วยกัน เช่นผู้ใช้สามารถสร้าง Subnet เสมือนที่เชื่อม Instance ชุดที่หนึ่งเข้าด้วยกัน และสร้างอีก Subnet ที่เชื่อม Instance ที่เหลือเข้าด้วยกันได้ โมดูล Neutron ยังสามารถสร้างและกำหนดค่าการทำงานของอุปกรณ์ Load Balancer เสมือน หรือ Firewall เสมือนได้ ซึ่งยังสามารถจัดการในกรณีที่มีหลาย tenant ทำให้เครือข่ายให้แยกจากกันได้

5.1.3 Heat และ Docker

Heat คือ main project ใน OpenStack Orchestration program ในการ implements ตัว Orchestration engine เพื่อทำการ Launch multiple Vms ที่อนุญาตให้ผู้พัฒนาโปรแกรมสามารถ ตั้งค่า/ปรับแต่ง Template โครงสร้างของทรัพยากรแต่ละด้าน (Compute , Storage และ Networking) เพื่อรองรับการใช้งาน Cloud Application (OpenStack, AWS CloudFormantion) ของผู้ใช้โดยการทำงานจะ based บน templates ซึ่งอยู่ในรูปแบบ form ของ text files ที่เป็น Codes

Docker เป็น Open platform สำหรับการพัฒนา (Developing), ย้าย (Shipping), ใช้งาน (Running) ตัว Applications Docker ถูกออกแบบมาเพื่อให้เราใช้งาน Develop, Ship, Run Application ของเราได้เร็วขึ้น ด้วย Docker เราสามารถแยก Applications จาก Infrastructure และใช้ Infrastructure เหมือนกับการจัดการ Application Docker ช่วยในการย้าย (Ship) Code/ Run Code/Test Code/Deploy ได้เร็วขึ้น และลด Cycle หรือช่วงเวลาในการเขียน Code กับ Running code ที่เราอาจสูญเสียไปเปล่าๆได้

5.2 ปัญหาอุปสรรคและแนวทางการแก้ไข

เนื่องทรัพยากรเป็นการอาศัยใช้จากบริษัทของอาจารย์ที่ปรึกษาวิทยานิพนธ์ของทางกลุ่ม จึงเกิดปัญหาที่ไม่คาดฝันบ้างเช่นไฟดับ หรือ เครื่องถูกปิดโดยไม่ทราบล่วงหน้าจึงทำให้การทำงานเกิดความล่าช้า รวมทั้งการพัฒนา OpenStack นั้นมีการปล่อยออกมาหลากหลายรุ่นทำให้การเลือกใช้มีข้อจำกัดทำให้เกิดอุปสรรคมากขึ้นเนื่องจากการไม่รับรองรุ่นบางรุ่น หรือ การเกิดข้อผิดพลาดที่อาจจะไม่สามารถแก้ได้ภายในระยะเวลาการทำโปรเจคนี้ รวมทั้งการแก้ปัญหาบัคต่างๆยังเป็นเรื่องที่ละเอียดอ่อน เนื่องจากไม่มีข้อมูลที่สามารถให้ศึกษาหรือกรณีตัวอย่างให้ศึกษามากนัก

5.3 แนวทางการพัฒนาต่อ

การพัฒนาในการใช้งาน Docker ร่วมกับ OpenStack เป็นสิ่งที่น่าสนใจทางผู้จัดทำหวังว่าการศึกษาครั้งนี้ทำให้เห็นภาพการเปรียบเทียบของทั้งสองเทคโนโลยี และสามารถตัดสินใจเลือกใช้ได้อย่างเหมาะสมรวมทั้งสามารถประยุกต์การใช้งานร่วมกันเพื่อให้เกิดประโยชน์สูงสุด

บรรณานุกรม

OpenStack . 2014. **OpenStack Developer Documentation.** [Online].

Available : http://docs.openstack.org.

Docker. 2014. **What is Docker?.** [Online].

Available : https://www.docker.com/what-docker.

Xath Cruz. 2012. **The Love and Hate of OpenStack.** [Online].

Available : http://cloudtimes.org/2012/08/21/love-hate-openstack.

Alex Collins. 2015. **Docker Linking Container** [Online].

Available: http://www.alexecollins.com/docker-linking-containers/

Romin Nirani. 2015. **Docker Tutorial Series Part8 Linking Container** [Online].

Available: https://rominirani.com/2015/07/31/docker-tutorial-series-part-8-linking-containers/