การประมวลผลบนเมฆ
CLOUD COMPUTING

ภัทรพล เปรมบุตรชัย

วิษณุพงศ ประยาทิศดิฐ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

การประมวลผลบนเมฆ

CLOUD COMPUTING

ภัทรพล เปรมกุศลชัย

วิชยุทธ ประชากิตติกุล

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

ปริญญานิพนธ์ปีการศึกษา 2557

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบการประมวลผลแบบกลุ่มเมฆ

CLOUD COMPUTING

ผู้จัดทำ

1. นาย ภัทรพล เปรมกุศลชัย รหัสนักศึกษา 54010974
2. นาย วิชยุทธ ประชากิตติกุล รหัสนักศึกษา 54011191

..... อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การประมวลผลบนเมฆ

นาย ภัทรพล เปรมกุศลชัย 54010974
นาย วิชยุทธ ประชากิตติกุล 54011191
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2557

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้มีวัตถุประสงค์ในการนำเทคโนโลยีการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) เข้าประยุกต์ใช้ทำคลาวไคล์แอนต์ (Cloud Client) เวอร์ชวลเดสก์ทอปอินฟราสตรัคเจอร์ (Virtual Desktop Infrastructure) ซึ่งก็คือการนำเทคโนโลยีเวอร์ชวลไลเซชัน (Virtualization) เข้ามาช่วยปรับปรุงระบบพีซี (PC: Personal Computer) ของผู้ใช้งาน แทนที่จะต้องซื้อฮาร์ดแวร์ (Hardware) ที่มีประสิทธิภาพสูงมาใช้งานเป็นพีซีแต่ละเครื่องสำหรับผู้ใช้งานแต่ละคน ซึ่งผู้ใช้งานแต่ละคนต่างก็ไม่ได้ใช้งานประสิทธิภาพของฮาร์ดแวร์เหล่านั้นอย่างเต็มที่ตลอดเวลา เทคโนโลยีเวอร์ชวลเดสก์ทอปอินฟราสตรัคเจอร์จะทำการยุบรวมอิมเมจ (Image) ของคอมพิวเตอร์ในองค์กรทั้งหมดมาอยู่บนเวอร์ชวลไลเซชันอินฟราสตรัคเจอร์ (Virtualization Infrastructure) ให้ใช้ฮาร์ดแวร์ร่วมกันทั้งซีพียู (CPU), แรม (RAM) และฮาร์ดไดรฟ์ (Hard Drives) และให้ผู้ใช้งานทำการเข้าถึงอิมเมจเหล่านี้ผ่านรีโมทไคลแอนต์ซอฟแวร์ (Remote Client Software) แทน

ในส่วนของการให้บริการการประมวลผลแบบกลุ่มเมฆ (Cloud Service) จะมุ่งเน้นเพื่อการให้บริการทางด้านเทคโนโลยีสารสนเทศแก่ผู้ใช้งาน ซึ่งเป็นระบบการประมวลผลที่ตอบสนองต่อความต้องการของผู้ใช้งาน กล่าวคือ ผู้ใช้งานจะสามารถกำหนดความต้องการของทรัพยากรที่ใช้งานได้ เพื่อความเหมาะสมของวัตถุประสงค์หรือเป้าหมายของการนำไปใช้ โดยการให้บริการประมวลผลแบบกลุ่มเมฆจะอยู่ที่ค่าใช้บริการที่จะคิดตามปริมาณการใช้งานจริง รวมทั้งสามารถใช้ทรัพยากรสารสนเทศที่มีความซับซ้อนได้อย่างมีประสิทธิภาพและเป็นการประหยัดค่าใช้จ่ายกับทรัพยากรสารสนเทศเหล่านั้น

CLOUD COMPUTING

Mr. Pattaraphol Premkusolchai 54010974
Mr. Wichayut Prachakittikul 54011191
Dr. Voravat Limpoka Advisor
Academic Year 2014

ABSTRACT

This report is intended for the Cloud Computing technology to use Cloud clientVirtual Desktop Infrastructure (VDI), that is Virtualization technology to help improve PC (PC: Personal Computer) of users. Instead of we have to buy high efficient Hardware to use a PC for each individual users.The Virtual Desktop Infrastructure Technology will be merged image of all the computer in theorganizationon the Virtualization Infrastructure, use the Hardware together both CPUs , memory (RAM) and Hard Drives, and users can access to use the image by remote client software.

In part of Cloud Service, we will focus on providing information technology services to users. The processing system of Cloud service is responsive to the requirements of the user. In addition, user can define the requirement of resource that can be use, appropriate to the purpose or goal of adoption. By providing cloud computing services will be depend on price of service that calculated from volume of actual use. Including to it can be use the complex of information resources efficiently and reduce cost of information resource.

กิตติกรรมประกาศ

คณะผู้จัดทำขอขอบคุณอาจารย์ที่ปรึกษาดร.วรวัฒน์ ลิ้มโภคาที่คอยให้ความสนใจสอบถามถึงความคืบหน้าของงานให้คำแนะนำและให้ความช่วยเหลือในเรื่องต่างๆอีกทั้งยังให้แนวคิดและประสบการณ์ที่ดีแก่คณะผู้จัดทำนอกจากนี้ยังขอขอบคุณคุณศิลาณี จรัสวชิรกุล และคุณชานนท์ ทรัพย์สำราญ ที่ช่วยเหลือให้คำแนะนำที่ดีกับคณะผู้จัดทำมาโดยตลอด

ขอขอบพระคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสถาบันการศึกษาในอดีต ที่ให้โอกาสดีๆทางการศึกษาแก่ข้าพเจ้ามาโดยตลอด

ขอขอบคุณเพื่อนๆ ในสาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง รวมทั้งเพื่อนต่างสาขาวิชาที่เป็นกำลังใจให้กันตลอดการทำงาน

สำหรับคุณงามความดีอันใดที่เกิดจากรายงานเล่มนี้คณะผู้จัดทำขอมอบให้บิดามารดาซึ่งเป็นที่รักและเคารพยิ่งตลอดจนครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้และประสบการณ์ที่ดีแก่คณะผู้จัดทำ

นาย ภัทรพล เปรมกุศลชัย

นาย วิชยุทธ ประชากิตติกุล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1ความสำคัญและที่มาของโครงการ

การเปลี่ยนแปลงของโครงสร้างธุรกิจในยุคปัจจุบันที่มีการแข่งขันสูง ความรวดเร็วในการทำธุรกิจถือเป็นปัจจัยหลักในการเอาชนะคู่แข่งขันได้ ระบบไอทีจึงเป็นสิ่งสำคัญที่ต้องสามารถตอบสนองการเติบโตและการเปลี่ยนแปลงอย่างรวดเร็วของโมเดลธุรกิจ ซึ่งการเปลี่ยนแปลงทางธุรกิจอย่างรวดเร็วส่งผลให้เกิดการแนวทางจัดการระบบไอทีแบบใหม่ที่เรียกว่า Cloud Computing

Cloud Computing เป็นระบบการบริหารจัดการทรัพยากรไอทีแบบใหม่ ที่มีความยืดหยุ่นสูง เพื่อรองรับกับธุรกิจยุคใหม่ที่พร้อมจะเปลี่ยนแปลงในทุกวินาที ระบบการจัดการทรัพยากรทางด้านไอทีแบบ Cloud Computing ส่งผลให้ธุรกิจสามารถเพิ่ม ลด และปรับเปลี่ยนระบบไอทีได้ทันทีที่ต้องการตามนโยบายการดำเนินธุรกิจ โดยไม่จำเป็นต้องรอการสั่งซื้ออุปกรณ์เพิ่มเติม เพราะโครงสร้างของระบบถูกออกแบบให้ทำงานอยู่บนระบบเสมือน (Virtualization) ทำให้สามารถเพิ่ม ลด และปรับเปลี่ยนโครงสร้างระบบไอทีที่รองรับการทำงานของ Software และ Application ที่จำเป็นต่อธุรกิจได้ทันที

1.2 วัตถุประสงค์ของโครงงาน

วัตถุประสงค์ของรายงานฉบับนี้จัดทำเพื่อศึกษาและทดลอง การทำงานของ storage ในรูปแบบต่างๆบน filesystem 2 แบบและการใช้งานระบบการประมวลผลแบบกลุ่มเมฆ (Openstack) เพื่อที่จะทำให้ทราบและเข้าใจถึงโครงสร้างหลักการทำงาน โดยเฉพาะ Swift (Object storage), Cinder (Block storage), Glance (Image services), neutron (network) และ ceilometer (Telemetry Service) รวมถึงประโยชน์ที่จะได้รับในการใช้งาน

1.3ขอบเขตของโครงงาน

เป็นการประมวลผลแบบกลุ่มเมฆที่สามารถใช้งานได้จริง ซึ่งเป็นการให้บริการด้าน Object storage (Swift) ,Block storage (Cinder) และ Image services (Glance) ซึ่งสามารถที่จะส่งข้อมูลเข้าไปใน Cloud ได้และสามารถที่จะดึงข้องมูลออกจาก Cloud ได้โดยผ่านโปรแกรมที่ใช้จัดการไฟล์สำหรับ Cloud storage คือ CloudBerry Lab และสามารถทดสอบการส่งข้อมูลที่เป็นแบบ Object storage และ Block storage แบบไม่อยู่บน Cloud โดยในส่วนของ neutron (network) สามารถ

ที่จะจัดการเครือข่ายและ IP address ในระบบประมวลผลกลุ่มเมฆได้และ ceilometer (Telemetry Service) สามารถที่จะวัดค่า Resource ต่างๆที่ได้มีการใช้งานในระบบประมวลผลกลุ่มเมฆได้

1.4 วิธีการดำเนินการ

1) ศึกษาการใช้งานของระบบไฟล์ (File System) 2 แบบคือ Ext4 และ XFS
2) ศึกษาการใช้งานและทดลองการสร้างvolume3แบบคือ striped , replicated และ distributed โดยใช้ GlusterFS
3) ศึกษาการทำงานและทดลองการทำ storage แบบ Object storage โดยใช้ GlusterFS-Swift และ Block storage โดย ISCII บนระบบไฟล์ (File System) ที่เป็น Ext4 และ XFS
4) ศึกษาโปรแกรมที่ใช้จัดการไฟล์สำหรับ Cloud storage คือ CloudBerry Lab
5) การวางแผนศึกษาระบบที่มีการให้บริการการประมวลผลแบบกลุ่มเมฆ (Cloud) ที่มีอยู่ในปัจจุบัน ตัว software ที่นำมาใช้คือ OpenStack
6) การวิเคราะห์และกำหนดรายการของสิ่งที่ระบบต้องการเครื่องมือและซอฟต์แวร์ที่จำเป็น ออกแบบโครงสร้างของระบบแผนภาพการทำงานระหว่างระบบและผู้ใช้งานเพื่อให้เป็นไปตามวัตถุประสงค์ของโครงงาน
7) การทดลองการติดตั้งปรับค่าตามแนวทางที่ได้วิเคราะห์และออกแบบไว้และปรับปรุงส่วนต่างๆของระบบและแก้ปัญหาต่างๆเพื่อทำให้ระบบทั้งหมดสามารถทำงานได้สอดคล้องกันและมีประสิทธิภาพตามวัตถุประสงค์ที่วางไว้

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้รับความรู้ความเข้าใจถึงโครงสร้างและการทำงานของการประมวลผลแบบกลุ่มเมฆ (Cloud)
2) ได้รับความรู้ความเข้าใจในการใช้เครื่องมือจัดการระบบการประมวลผลแบบกลุ่มเมฆ (OpenStack) เช่น Swift (Object storage), Cinder (Block storage), Glance (Image services), neutron (network) และ ceilometer (Telemetry Service) เป็นต้น
3) ทำให้เข้าใจถึงกระบวนการพัฒนาระบบตั้งแต่การวางแผนการวิเคราะห์การออกแบบการสร้างและการนำไปใช้งาน
4) ได้รับความเข้าใจเกี่ยวกับการให้บริการคลาวน์ ซึ่งเป็นการให้บริการด้านโครงสร้างพื้นฐาน Iaas (Infrastructure as a Service)
5) ได้รับความรู้และทักษะในการใช้ลินุกซ์ (Linux) เช่น CentOS

6) ได้รับความรู้ความเข้าใจถึงการทำงานของระบบไฟล์ (File System) ของ Ext4 และ XFS
7) ได้รับความรู้ความเข้าใจในการใช้การทำ storage ในรูปแบบต่างๆ ของ GlusterFS
8) ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานการใช้งานของ Object storage และ Block storage ทั้งบน Cloud และไม่เกี่ยวกับ Cloud
9) ได้รับความรู้เกี่ยวกับ Software Define network (SDN)

1.6 ส่วนประกอบของรายงาน

รายงานฉบับนี้ได้แบ่งเนื้อหาออกเป็น 5 บทด้วยกันคือ

บทที่ 1 บทนำกล่าวถึงความสำคัญและที่มาของโครงงานวัตถุประสงค์ของโครงงานขอบเขตของโครงงานวิธีการดำเนินการประโยชน์ที่คาดว่าจะได้รับและส่วนประกอบของรายงาน

บทที่ 2 ทฤษฎีที่เกี่ยวข้องกล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงงานประกอบด้วยอะไรบ้างให้บรรยายทฤษฎีทั้งหมดโดยละเอียด

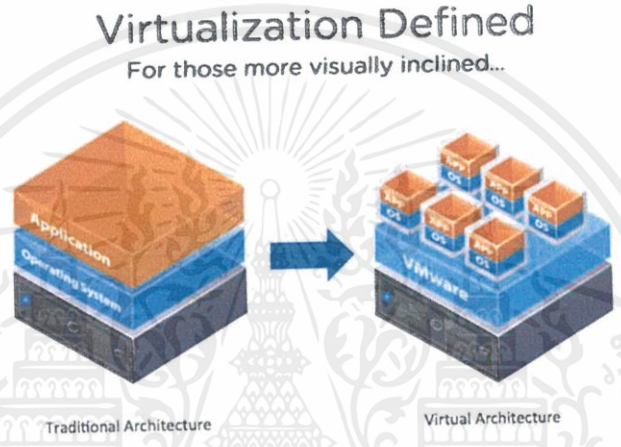
บทที่ 3 การออกแบบและพัฒนากล่าวถึงรายละเอียดของโครงงานนี้ส่วนที่ได้ออกแบบและพัฒนาขึ้นการทำงานของระบบ

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงการเตรียมการทดลองทั้งการจัดเตรียมฮาร์ดแวร์ ซอฟต์แวร์ สภาวะแวดล้อมในการทำการทดลอง ข้อมูลทดสอบ การทำงานหรือการจำลองการทำงานของระบบผลการทดลอง

บทที่ 5 บทสรุปและข้อแนะนำ กล่าวถึงการวิจารณ์ระบบที่ได้ทำการศึกษา ทดลอง ปัญหาอุปสรรคและการแก้ไขปัญหา รวมถึงแนวทางการพัฒนาต่อ

บทที่ 2
ทฤษฎีพื้นฐานที่เกี่ยวข้อง

2.1 เทคโนโลยีการทำเวอร์ชวลไลเซชัน (Virtualization Technology)

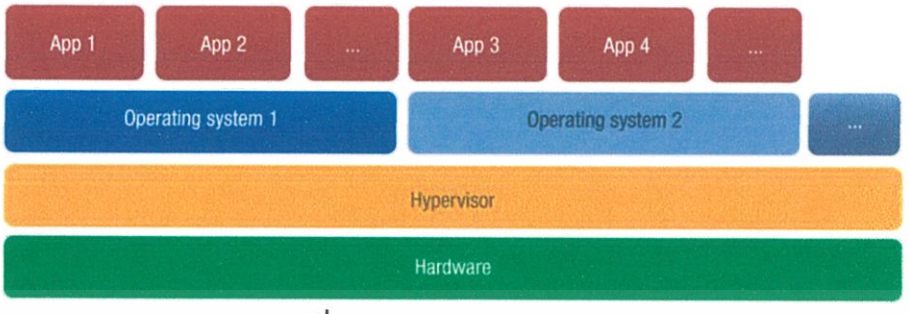


รูปที่ 2.1 ภาพแสดง Virtualization Technology

Virtualization คือ การจำลองเครื่องเสมือนด้วยซอฟต์แวร์ ที่ทำให้คอมพิวเตอร์หนึ่งเครื่องสามารถทำงานเป็นเครื่องเสมือนหลายๆ ระบบได้ โดยแต่ละระบบมีทรัพยากรหน่วยความจำ, ฮาร์ดดิสก์ และอุปกรณ์เน็ตเวิร์กเสมือนที่เป็นอิสระต่อกัน เครื่องเสมือนแต่ละเครื่องจึงสามารถมีระบบปฏิบัติการและซอฟต์แวร์เป็นของตน เองโดยอิสระ

2.1.1 ประเภทของ Hypervisor

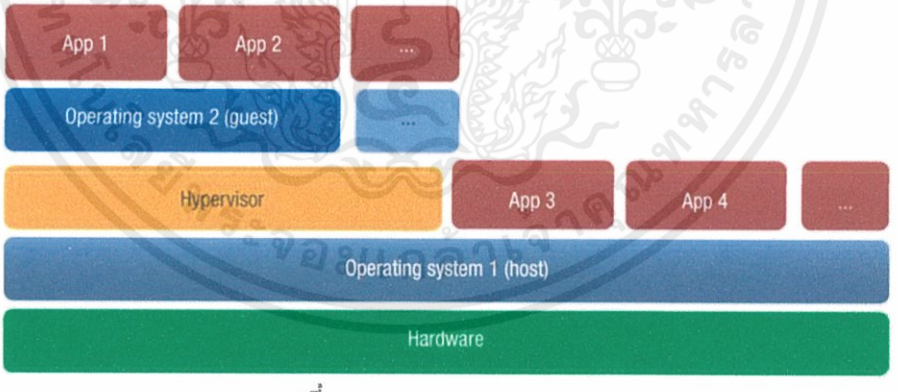
การสร้างระบบ Virtualization ได้กำหนดชื่อเรียกตัว software ที่ทำหน้าที่ virtual ว่า Hypervisor หรือVirtual Machine Manager (VMM) ได้มีการแบ่ง hypervisor ออกเป็น 2 ประเภทคือ



รูปที่ 2.2 Type 1 Hypervisor

"Type 1 Hypervisor" สถาปัตยกรรมเนทีฟ (Native บางทีเรียก Bare-Metal หรือ Enterprise Architecture) เป็นสถาปัตยกรรมที่ไฮเปอร์ไวเซอร์ถูกติดตั้งโดยตรงลงบนฮาร์ดแวร์ เสมือนหนึ่งเป็นระบบปฏิบัติการของเครื่องไปโดยปริยาย ด้วยโครงสร้างแบบนี้ทำให้ทำงานได้ประสิทธิภาพสูงกว่าสถาปัตยกรรมแบบโฮสต์อย่างมิอาจเทียบเคียง

จุดด้อยของสถาปัตยกรรมแบบเนทีฟ ก็คือ การรองรับของฮาร์ดแวร์ เครื่องที่จะนำมาใช้งานบนโครงสร้างนี้ต้องผ่านการรับรองจากผู้ผลิตไฮเปอร์ไวเซอร์เสียก่อน ถึงจะมั่นใจว่าสามารถทำงานร่วมกันได้เป็นอย่างดี แม้จะรองรับบนฮาร์ดแวร์เฉพาะบางรุ่น แต่ด้วยความร้อนแรงของโลกเสมือนทำให้เครื่องเซิร์ฟเวอร์รุ่นใหม่ๆ ล้วนตบเท้าดาหน้าเข้ามาเป็นสาวกโดยพร้อมเพรียงกัน จะหาเครื่องเซิร์ฟเวอร์รุ่นที่ทำงานกับไฮเปอร์ไวเซอร์แบบเนทีฟ ไม่ใช่เรื่องยากเกินไปนักในยุคปัจจุบัน

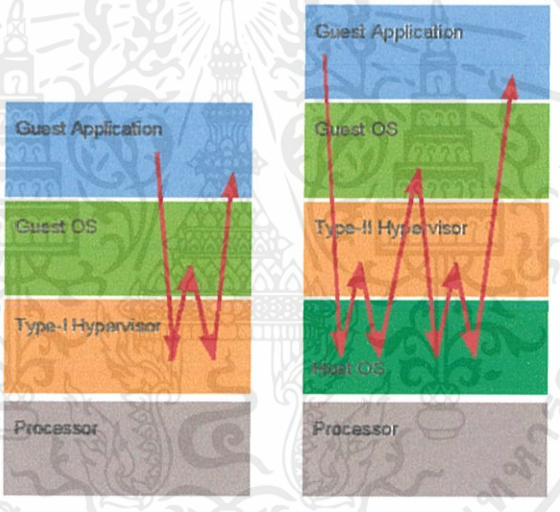


รูปที่ 2.3 Type 2 Hypervisor

"Type 2 Hypervisor"ในสถาปัตยกรรมแบบนี้ นอกจากที่เราจะต้องมีตัวเครื่อง คือ ฮาร์ดแวร์ หรือที่เรียกว่า "**โฮสต์ (Host)**" แล้ว เรายังต้องมี "**ระบบปฏิบัติการโฮสต์ (Host Operating System)**" อยู่ด้วย

ไฮเปอร์ไวเซอร์ (Virtualization Layer) จะถูกติดตั้งลงบนระบบปฏิบัติการโฮสต์ ซึ่งถือได้ว่าไฮเปอร์ไวเซอร์เป็นแอพพลิเคชั่นตัวหนึ่งบนโฮสต์ จากนั้นเราจึงจะสามารถสร้างจักรกลเสมือนขึ้นมาได้ ระบบปฏิบัติการที่ถูกติดตั้งลงบนจักรกลเสมือนจะถูกเรียกว่า "ระบบปฏิบัติการเกสต์ (Guest Operation System)" โดยที่ระบบปฏิบัติการโฮสต์ และระบบปฏิบัติการเกสต์ จะเป็นตัวเดียวกันหรือต่างกันก็ตามสะดวก สถาปัตยกรรมแบบโฮสต์ นิยมใช้สำหรับการทดสอบระบบ หรือการเขียนโปรแกรม ซึ่งส่วนใหญ่มักนิยมติดตั้งลงบนเครื่องพีซี หรือโน๊ตบุ๊ค และใช้งานส่วนตัว

ข้อดีของสถาปัตยกรรมโฮสต์ คือ สามารถติดตั้งได้ง่าย, ไม่สนใจฮาร์ดแวร์ด้านล่าง, มีเพียงระบบปฏิบัติการบนเครื่องโฮสต์ที่รองรับก็เพียงพอ ปัจจุบันทั้งวินโดวส์, ลินุกซ์ รวมถึงแม็คโอเอส (Mac OS) ก็มีไฮเปอร์ไวเซอร์แบบนี้ให้ใช้งานกัน

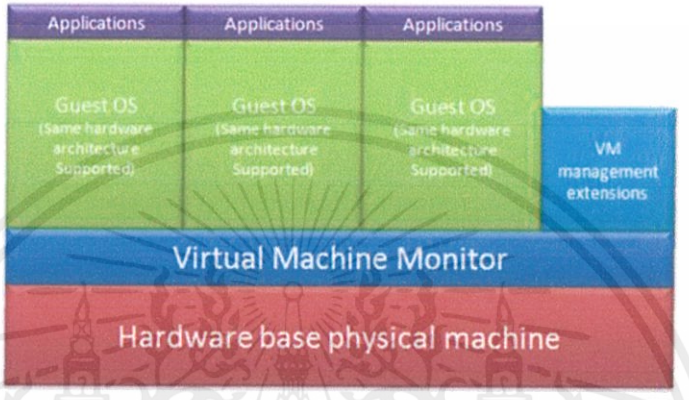


รูปที่ 2.4 Type 1: Native หรือ Bare Metal และ Type 2: Hosted

สำหรับข้อด้อย คือ จักรกลเสมือนจะทำงานได้ช้า เพราะถ้าดูจากโครงสร้างแล้ว กว่าแอพพลิเคชั่นบนจักรกลเสมือนจะเข้าถึงฮาร์ดแวร์จริงๆ จะต้องผ่านระบบปฏิบัติการตัวใหญ่ถึง 2 ตัว คือ ระบบปฏิบัติการโฮสต์ และระบบปฏิบัติการของเกสต์เอง แล้วยังต้องนับไฮเปอร์ไวเซอร์เข้าไปอีก ผู้ที่เคยใช้งานจักรกลเสมือนบนโครงสร้างแบบนี้ ถึงกับข้องใจในความล่าช้า เข็ดขยาดโลกเสมือนไปเลยก็มี แต่อย่าลืมว่าโครงสร้างนี้ถูกออกแบบมาสำหรับการใช้งานเพื่อทดสอบหรือพัฒนาโปรแกรม รวมถึงงานกาฝากขนาดเล็กเท่านั้น เช่น เราอาจจะมีโน๊ตบุ๊คที่ติดตั้ง Windows 7 ไว้ แล้วต้องการทดสอบการทำงานของ Windows Server 2008 R2 เราก็เพียงแต่สร้างจักรกลเสมือนขึ้นมาเพื่อติดตั้ง Windows Server 2008 R2 เท่านั้น ทำให้เราไม่จำเป็นต้องหาเครื่องใหม่มาใช้ในการทดสอบ เป็นต้น

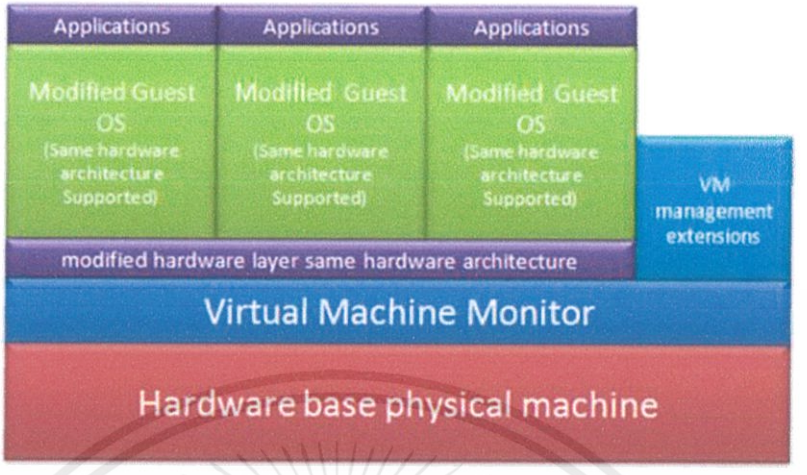
2.1.2 สถาปัตยกรรมของการทำเวอร์ชวลไลเซชั่น

มีทางเลือกมากมายในการ สนับสนุนเทคโนโลยีการทำงานของเวอร์ชวลไลเซชัน แต่มีเทคนิคอยู่ 2 รูปแบบที่เป็นผู้นำทางด้านเทคโนโลยีนี้อยู่ก็คือเทคนิคแบบ Full virtualization และ Para-virtualization



รูปที่ 2.5 แผนภาพแสดงการทำงานของ Full Virtualization

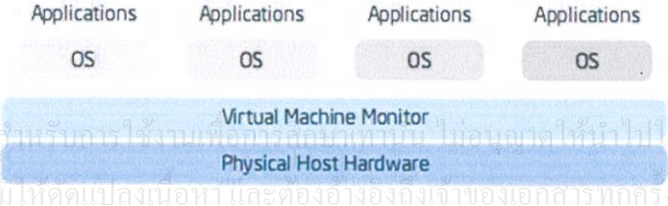
1) การทำ Full virtualization สำหรับการทำเวชวลไลเซชั่นในรูปแบบนี้ ถูกออก แบบเพื่อเตรียมการทำให้เป็นรูปแบบเสมือนทั้งหมดของฮาร์ดแวร์ และสร้างระบบเสมือนที่สมบูรณ์ ในที่นี้จะทำให้เราสามารถที่นำ ระบบปฏิบัติการอื่นๆ มาติดตั้งและสามารถที่จะทำงานอยู่บนเครื่องคอมพิวเตอร์เดียวกันได้ ซึ่งเราจะเรียกว่าระบบปฏิบัติที่มาติดตั้งเพิ่มเติมนี้ว่า 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



รูปที่ 2.6 แผนภาพแสดงการทำงานของ Paravirtualization

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

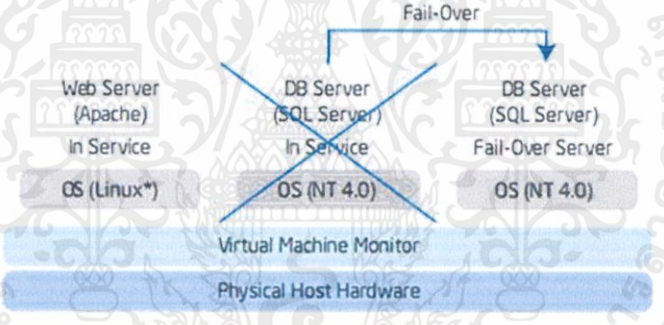
2.1.3 การใช้งานเทคโนโลยีเวชวลไลเซชั่น (Usage Virtualization Technology)



รูปที่ 2.7 โครงสร้างการใช้ Virtualization Technology ให้มี Consolidation

Consolidation: การรวบรวมทรัพยากรต่างๆ เช่น Storage และ Server เข้ามาอยู่จุดเดียว

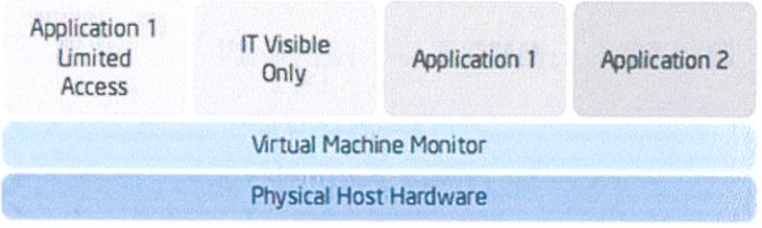
- เพิ่มอัตราการใช้งานฮาร์ดแวร์ที่มีอยู่อย่างจำกัด ให้เป็นไปอย่างมีประสิทธิภาพและคุ้มค่ามากที่สุดเนื่องจากฮาร์ดแวร์บางตัวอาจจะใช้งานไม่มากนัก ขณะที่บางตัวใช้งานมากจนเกินไป
- ช่วยประหยัดพลังงานค่าไฟ ประหยัดพื้นที่ ประหยัดค่าใช้จ่ายต่างๆ
- การดูแลบริหารจัดการและบำรุงรักษาระบบทำได้ง่ายขึ้น เนื่องจากมีฮาร์ดแวร์เพียงชุดเดียวเท่านั้น
- สามารถติดตั้งได้หลายระบบปฏิบัติการและแอพพลิเคชั่นบนสภาพแวดล้อมที่แตกต่างกัน เนื่องจากบางแอพพลิเคชัน (Legacy Applications) จำเป็นต้องทำงานบนระบบปฏิบัติการที่จำเพาะ เป็นการยืดอายุการใช้งานแอพพลิเคชั่นเก่าโดยมีค่าใช้จ่ายไม่มากและมีความ เสี่ยงน้อยที่สุด



รูปที่ 2.8 โครงสร้างการใช้ Virtualization Technology ให้มี Reliability

Reliability: ความน่าเชื่อถือในการทำงาน

- เพิ่มความสะดวกและความคล่องตัวทางธุรกิจ ระบบเสมือนสามารถจัดเตรียมเพื่อรองรับหรือปรับขนาดได้ภายในไม่กี่นาทีเพื่อรองรับการติดตั้งแอพพลิเคชั่นใหม่ ภาระงานที่เพิ่มขึ้นหรือกรณีที่เกิดความผิดพลาดขึ้นในระบบ
- การ Backup และ Recovery สามารถทำงานได้อย่างรวดเร็วขึ้น ภายใต้ Virtual Machine เดียวกัน
- รองรับการทำงานหลากหลายอย่าง อาทิ System Migration, Backup และ Recovery



รูปที่ 2.9 โครงสร้างการใช้ Virtualization Technology ให้มี Security

Security: การรักษาความปลอดภัยที่ดีขึ้น

- สามารถกำหนดระดับความปลอดภัยให้แต่ละระบบเสมือนให้มีความแตกต่างกันได้
- การโจมตีในรูปแบบดิจิตอล (Virus, Hacker) จะถูกแยกออกจากกัน สำหรับแต่ละระบบ
- การทำงานที่ล้มเหลวส่วนใหญ่มาจากซอร์ฟแวร์ จึงมั่นใจได้ว่าความผิดพลาดที่เกิดขึ้นจะไม่ส่งผลกระทบต่อระบบเสมือนอื่นๆ

2.2 ระบบประมวลผลกลุ่มเมฆ (Cloud Computing)

คำว่า เมฆ หมายถึงเครือข่าย (Network) หรืออินเทอร์เน็ต(Internet) ในอีกทางหนึ่งสามารถกล่าวได้ว่า เมฆ คือบางสิ่งที่ปรากฏในสถานที่ห่างไกล เมฆสามารถให้บริการผ่านเครือข่ายเช่นในเครือข่ายสาธารณะหรือบนเครือข่ายส่วนตัวคือ WAN, LAN หรือ VPN "การใช้งานเช่น อีเมล (e-mail) การประชุมผ่านเว็บ (web conferencing) การจัดการลูกค้าสัมพันธ์ (customer relationship management: CRM) ทั้งหมดทำงานในเมฆ"

Cloud Computing หมายถึง การจัดการ (manipulating) การกำหนดค่า (configuring) และการเข้าถึง (accessing) การใช้งานออนไลน์ มันมีการจัดเก็บข้อมูลออนไลน์โครงสร้างพื้นฐานและการประยุกต์ใช้งาน

รูปที่ 2.10 โครงสร้างระบบประมวลผลกลุ่มเมฆ

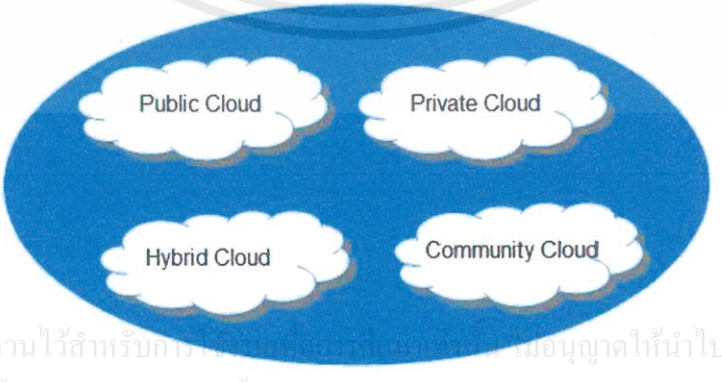
เราไม่จำเป็นต้องติดตั้งส่วนของซอฟต์แวร์บนพีซีของเรา และนี่คือวิธีการที่ Cloud Computing แก้ปัญหาการพึ่งพาแพลตฟอร์ม (platform dependency issues) ดังนั้น Cloud Computing คือการสร้างธุรกิจของเราให้ทำงานแบบเปลี่ยนแปลงได้ง่ายและทำงานร่วมกัน

แนวคิดพื้นฐานมีบริการที่แน่นอนและรูปแบบการทำงานเบื้องหลังที่ทำให้ Cloud Computing เป็นไปได้และเข้าถึงผู้ใช้งานได้ มีรูปแบบการทำงานได้ดังนี้

- รูปแบบการพัฒนา (Deployment Models)
- รูปแบบการบริการ (Service Models)

2.2.1 รูปแบบการพัฒนา (Deployment Models)

รูปแบบการพัฒนาที่กำหนดการเข้าถึงของคลาวด์ แบ่งออกเป็น 4 ประเภท



รูปที่ 2.11 รูปแบบการพัฒนา

Public Cloud ช่วยให้ระบบและบริการให้สามารถเข้าถึงได้ง่ายให้กับประชาชนทั่วไป Public Cloud อาจจะมีความปลอดภัยน้อยลงเพราะมันเปิดกว้าง เช่น อีเมล

Private Cloud ช่วยให้ระบบและบริการที่สามารถเข้าถึงได้ภายในองค์กร มันมีการรักษาความปลอดภัยที่เพิ่มขึ้นเนื่องจากมีลักษณะความเป็นส่วนตัว

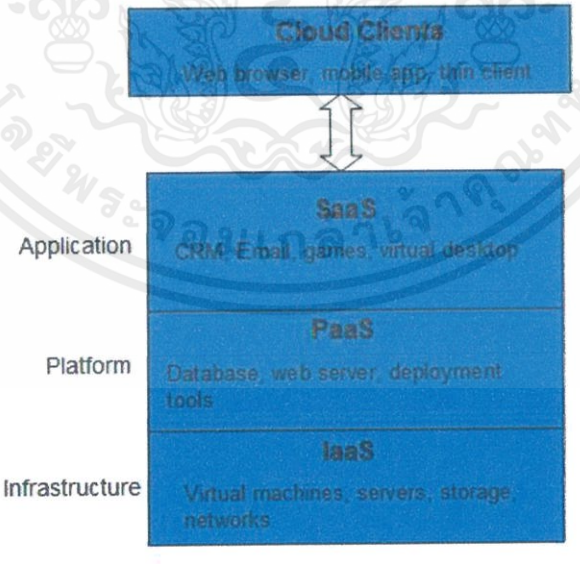
Community Cloud ช่วยให้ระบบและบริการที่สามารถเข้าถึงได้โดยกลุ่มขององค์กร

Hybrid Cloudเป็นส่วนผสมของ Public และ Private แต่กิจกรรมที่สำคัญจะดำเนินการโดยใช้ Private ในขณะที่กิจกรรมที่ไม่สำคัญจะมีการดำเนินการโดยใช้ Public

2.2.2 รูปแบบการบริการ (Service Models)

- โครงสร้างพื้นฐานเป็นบริการ (Infrastructure as a Service : IaaS)
- แพลทฟอร์มที่เป็นบริการ (Platform as a Service : PaaS)
- ซอฟต์แวร์ที่เป็นบริการ (Software as a Service : SaaS)

โครงสร้างพื้นฐานเป็นบริการ (IaaS)เป็นระดับพื้นฐานที่สุดของการบริการ แต่ละรุ่นให้บริการจะใช้ประโยชน์จากรูปแบบการบริการพื้นฐานเช่นการสืบทอดกลไกการรักษาความปลอดภัยและการจัดการจากแบบจำลองพื้นฐานดังแสดงในแผนภาพต่อไปนี้

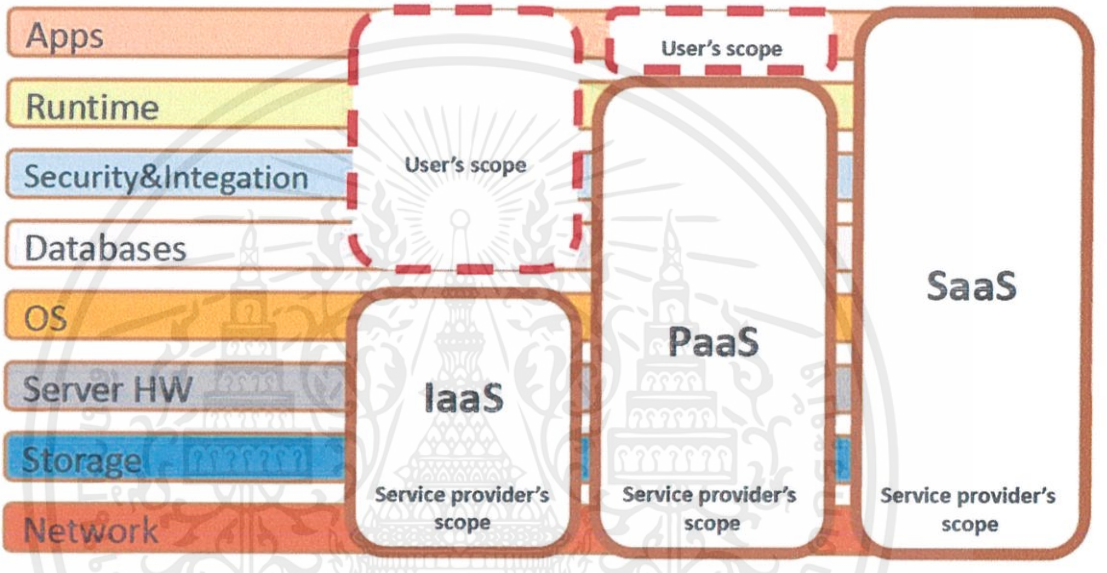


รูปที่ 2.12 รูปแบบการบริการ

โครงสร้างพื้นฐานเป็นบริการ (IAAS) ให้การเข้าถึงทรัพยากรพื้นฐานเช่นเครื่องกายภาพเสมือนเครื่องเสมือนการเก็บรักษาและอื่น ๆ

แพลทฟอร์มที่เป็นบริการ (PAAS) ให้สภาพแวดล้อมรันไทม์สำหรับการใช้งานการพัฒนาและการใช้งานเครื่องมืออื่น ๆ

ซอฟต์แวร์ที่เป็นบริการ (SAAS) ช่วยให้บริการงานซอฟต์แวร์แก่ผู้ใช้



รูปที่ 2.13 รูปแบบการบริการ

2.2.3 องค์ประกอบของระบบประมวลผลกลุ่มเมฆ

ระบบประมวลผลกลุ่มเมฆ จำเป็นต้องอาศัยองค์ประกอบที่สำคัญคือ

1) อินเตอร์เน็ตที่มีช่องสัญญาณสูงจนเกือบจะไม่มีจำกัด (Nearly unlimited bandwidth)
2) เทคโนโลยีระบบเสมือนจริง (Increasingly sophisticated virtualization technologies)
3) สถาปัตยกรรมเครือข่ายที่รองรับการเข้าถึงพร้อมกันจำนวนมาก (Multitenant Architectures)
4) ลักษณะการใช้งานได้ของเซิร์ฟเวอร์ประสิทธิภาพสูง (Availability of extremely powerful servers)

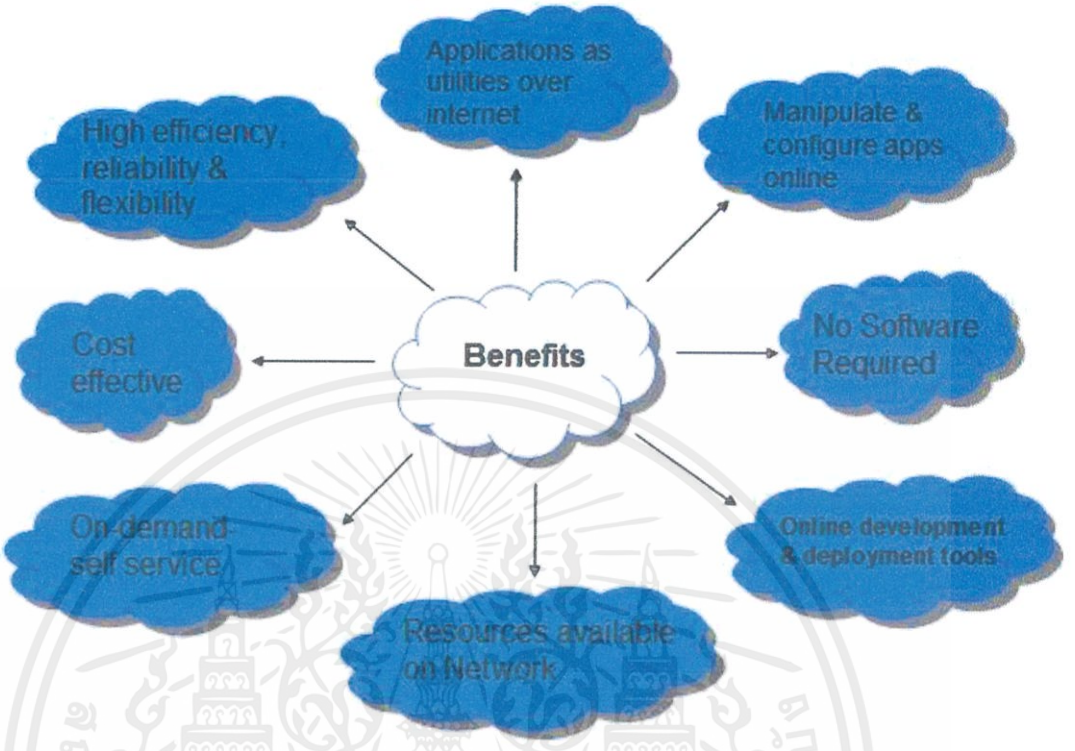
2.2.4 โครงสร้างการประมวลผลแบบกลุ่มเมฆ

การประมวลผลแบบกลุ่มเมฆจะมีโครงสร้างดังนี้

- กลุ่มเมฆของเซิร์ฟเวอร์ (cloud server) ซึ่งเป็นเซิร์ฟเวอร์จำนวนมหาศาลนับหมื่นนับแสนเครื่องที่ตั้งอยู่ในที่เดียวกัน กลุ่มเมฆนี้ต่อเชื่อมเข้าหากันด้วยเครือข่ายเป็นระบบ Grid ในระบบนี้จะใช้ซอฟต์แวร์ Virtualization ในการทำงานเพื่อให้โปรแกรมประยุกต์ ขึ้นกับระบบน้อยที่สุด
- ส่วนติดต่อกับผู้ใช้ (User interaction interface) ทำหน้าที่รับคำขอบริการจากผู้ใช้ในรูปแบบเว็บโปรโตคอล · ส่วนจัดเก็บรายการบริการ (Services Catalog) เก็บและบริหารรายการของบริการ ผู้ใช้สามารถค้นดูบริการที่มีจากที่นี่
- ส่วนบริหารงาน (system management) ทำหน้าที่กำหนดทรัพยากรที่เหมาะสม เมื่อผู้ใช้เรียกใช้บริการ เมื่อมีการขอใช้บริการ ข้อมูลการขอ request จะถูกส่งผ่านให้ส่วนนี้
- ส่วนจัดหาทรัพยากร (provisioning services) จากนั้นส่วนบริหารงานจะติดต่อกับส่วนนี้ เพื่อจองทรัพยากรจากกลุ่มเมฆ และเรียกใช้โปรแกรมประยุกต์แบบเว็บที่เหมาะสมให้ เมื่อโปรแกรมประยุกต์ทำงานแล้วก็จะส่งผลที่ได้ให้ผู้ใช้ที่เรียกใช้บริการต่อไป
- ส่วนตรวจสอบข้อมูลการใช้งาน (Monitoring and Metering) เพื่อใช้ในการเก็บค่าบริการหรือเก็บข้อมูลสถิติเพื่อปรับปรุงระบบต่อไป

2.2.5 ประโยชน์ที่ได้รับ

- สามารถเข้าถึงการใช้งานที่เป็นสาธารณูปโภคผ่านทางอินเทอร์เน็ต
- จัดการและกำหนดค่าโปรแกรมประยุกต์ออนไลน์ได้ตลอดเวลา
- มันไม่จำเป็นต้องมีการติดตั้งซอฟต์แวร์ในการเข้าถึงหรือจัดการกับโปรแกรมคลาวด์
- Cloud Computing มีการพัฒนาและการใช้งานเครื่องมือออนไลน์, การเขียนโปรแกรมสภาพแวดล้อมรันไทม์ผ่านแพลทฟอร์มที่เป็นรูปแบบการบริการ.
- ทรัพยากรคลาวด์มีอยู่ผ่านเครือข่ายในลักษณะที่ให้แพลตฟอร์มการเข้าถึงได้อย่างอิสระตามประเภทของลูกค้า
- Cloud Computing มีความต้องการในการให้บริการด้วยตนเอง. ทรัพยากรที่สามารถนำมาใช้โดยไม่มีต้องโต้ตอบกับผู้ให้บริการคลาวด์
- Cloud Computing มีค่าใช้จ่ายเพิ่มตามประสิทธิภาพที่สูงขึ้น
- Cloud Computing มีสมดุลภาระที่ทำให้มันน่าเชื่อถือมากขึ้น



รูปที่ 2.14 ประโยชน์ของ Cloud

2.2.6 ข้อจำกัดของระบบประมวลผลกลุ่มเมฆ

แม้ว่าหลักการของระบบประมวลผลกลุ่มเมฆจะมีประโยชน์ต่อภาพรวมทั้งส่วนของผู้ใช้ ผู้พัฒนา และผู้ให้บริการ แต่ยังคงมีข้อจำกัดบางประการที่มีผลต่อการให้บริการบนสภาพแวดล้อมกลุ่มเมฆ

1) ความปลอดภัยและความเป็นส่วนตัว

เมื่อข้อมูลและแอพพลิเคชั่นถูกส่งไปยังกลุ่มเมฆผ่านเครือข่ายอินเตอร์เน็ต อีกทั้งสภาพแวดล้อมที่กระบวนการทำงานต้องอาศัยความสามารถของกลุ่มเมฆเซิร์ฟเวอร์หลายกลุ่มบนเครือข่าย องค์กรธุรกิจและผู้ใช้ระดับบุคคลอาจไม่มั่นใจและกังวลเกี่ยวกับมาตรฐานความปลอดภัยของข้อมูลบนเครือข่ายอินเตอร์เน็ต นอกจากนี้โดยสภาพแวดล้อมของกลุ่มเมฆที่ข้อมูลการสนทนา และประวัติการเข้าใช้บริการเครือข่าย (Log) จะไม่ได้ถูกจัดเก็บบนระบบไอทีขององค์กร แต่กระจายไปบนเครือข่ายอินเตอร์เน็ต จึงมีความเสี่ยง หากข้อมูลการติดต่อสื่อสารระหว่างองค์กรธุรกิจซึ่งเป็นความลับทางการค้าอาจถูกจารกรรมจากเครือข่าย

2) ไม่มีมาตรฐานของ Platform

ผู้ให้บริการกลุ่มเมฆมีมาตรฐานแพลทฟอร์มที่แตกต่างกัน โดยอะเมซอน เว็บ เซอร์วิส เป็นแบบซอฟต์แวร์ฟรีโปรแกรม (Linux, Apache, MySQL, Perl/PHP : LAMP) **ขณะที่** Google App Engine **เป็นแบบมาตรฐานเฉพาะผลิตภัณฑ์** (Proprietary Formats) และผู้ใช้วินโดว์มักจะใช้บริการจาก GoGrid ดังนั้นสำหรับผู้พัฒนาแอพพลิเคชั่น หากต้องการ **ให้ผลงานครอบคลุมตลาดผู้ใช้หลาย ๆ กลุ่ม ก็ต้องพัฒนาแอพพลิเคชั่นบนหลายแพลทฟอร์ม** ซึ่งเป็นเรื่องยุ่งยาก

3) ความเชื่อถือได้ (Reliability)

โอกาสที่บริการกลุ่มเมฆจะล่มหรือไม่สามารถให้บริการได้ในบางขณะจะส่งผล กระทบอย่างมากต่อผู้ใช้ในระบบ

4) คุณสมบัติด้านการเคลื่อนย้ายข้อมูล (Portabilitiy)

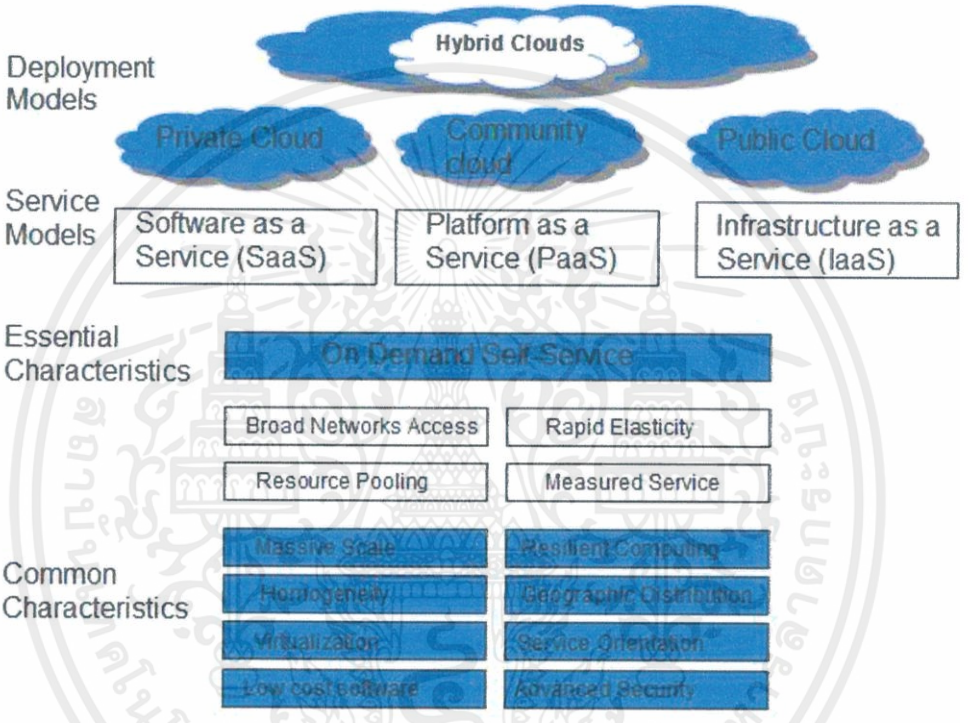
ตามหลักการทำงานแบบแบ่งปันประสิทธิภาพของระบบไอทีบนกลุ่มเมฆหลาย ๆ กลุ่มหมายถึงกระบวนการ ประมวลผลแต่ละชิ้นงานอาจเริ่มต้นและสิ้นสุดลงโดยผ่านการ **ทำงานบนกลุ่มเมฆ (เซิร์ฟเวอร์) มากกว่า 1 กลุ่ม ในขั้นตอนการเคลื่อนย้ายข้อมูลจากกลุ่ม** เมฆหนึ่งไปยังอีกกลุ่มเมฆหนึ่ง แม้จะเกิดความคุ้มค่าของการใช้ประสิทธิภาพของคอมพิวเตอร์ **แต่อาจสิ้นเปลืองทรัพยากรด้านการสื่อสาร** (Bandwidth) **บนเครือข่ายอินเตอร์เน็ตในมูล** ค่าที่สูงกว่า

5) **เซิร์ฟเวอร์ทางกายภาพ**

ระบบประมวลกลุ่มเมฆให้บริการด้วยเทคโนโลยีเสมือนจริง (Virtualization) อย่างไรก็ตามในความเป็นจริงอุปกรณ์ฮาร์ดแวร์ที่ทำหน้าที่เซิร์ฟเวอร์ทางกายภาพยังคงมีอยู่ **จริง ซึ่งมีโอกาสที่ติดตั้งกระจายอยู่ในประเทศต่าง ๆ ทั่วทุกมุมโลก ประเด็นที่ยังคงเป็นกังวล** คือข้อมูลทางธุรกิจและข้อมูลที่มีผลต่อความมั่นคงของประเทศ อาจถูกจัดเก็บบนเซิร์ฟเวอร์ **ในประเทศอื่น และมีความเสี่ยงที่รัฐบาล หรือทางการ ตลอดจนภาคเอกชนของประเทศที่** เป็นที่ตั้งของเซิร์ฟเวอร์จะสามารถเข้าถึงข้อมูลเหล่านั้นได้

2.2.7 คุณสมบัติที่จำเป็นของ Cloud Computing

เนื่องจาก Cloud เป็นบริการ มันไม่มีความหมายที่ตายตัว จึงมีหน่วยงาน NIST ของสหรัฐสร้างกฎเกณฑ์ สำหรับนำไปประเมินผลิตภัณฑ์ โดย Cloud คุณสมบัติ 5 ข้อ เป็นหลักสากล หากจะพิจารณาว่าอะไรเป็น Cloud หรือไม่ จำอันนี้ไปใช้ได้เลย คือ



รูปที่ 2.15 รูปแสดงข้อจำกัดของ Cloud

1. On-demand / Self-Service	O ความสามารถที่เปิดให้ consumer ขอใช้บริการได้เองเมื่อต้องการใช้และไม่ต้องรอให้ใครมาดำเนินการ
2. Broad network access	O ความสามารถให้บริการแก่ consumer ผ่านระบบ network และผ่านอุปกรณ์ที่หลากหลาย เช่น PC, table, mobile phone
3. Resource Pooling	O การที่ผู้ให้บริการ (Provider) สร้าง resource เป็น pool เพื่อรองรับผู้ใช้งานหลายๆ ราย (multi-tenant) และตอบสนองความต้องการของผู้ใช้งานที่มีความต้องการ (demand) แตกต่างกัน O ผู้ใช้งานสามารถใช้งานเครื่องใดก็ได้ แต่บางครั้งก็อาจมีการร้องขอแบบเจาะจงในบางข้อเช่น การระบุประเทศที่ตั้งของ Server เป็นต้น
4. Rapid elasticity	O ความสามารถในการเตรียม resource ที่ยืดหยุ่น เพิ่มได้/ลดได้ ย้ายได้ มีให้ใช้ตลอดไม่จำกัดจำนวน ไม่จำกัดเวลา
5. Measured Services	O มีรูปแบบของมิเตอร์ ทำให้ consumer และ provider สามารถ monitor การใช้งาน, ควบคุมการใช้งาน, ออก report

ตารางที่ 2.1 แสดงประโยชน์ของ Cloud Computing

2.3 Openstack

รูปที่ 2.16 สัญลักษณ์ Openstack

Openstack เป็นโครงการที่เกิดจากความร่วมมือของ Rackspace ที่มีชื่อว่า คลาวด์ไฟล์ (Cloud File) และ NASA ที่มีชื่อว่า เนบิลล่า (Nebula) โดยสัญญาอนุญาตของ Openstack เป็น apache License 2.0 ซึ่ง Openstack เป็นซอฟต์แวร์ที่นำไปจัดการระบบการประมวลผลแบบกลุ่มเมฆที่ให้บริการด้านโครงสร้างพื้นฐาน Iaas (Infrastructure as a service) เป็น Open Source ที่ไม่ต้องเสียค่าใช้จ่ายในการใช้บริการ

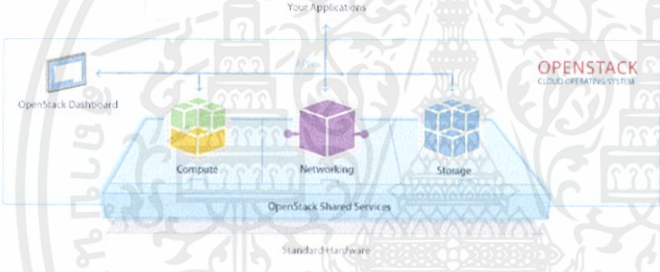
โดย Openstack จะใช้ภาษาไพทอน (Python) ในการเขียน และการจัดการประเภทของการประมวลผลแบบกลุ่มเมฆทั้ง 2 ประเภทได้แก่ พับบลิคคลาวด์ (Public Cloud) และไพรเวท คลาวด์ (Private Cloud)

2.3.1องค์ประกอบของ Openstack

การทำงานของ Openstack เกิดขึ้นจากองค์ประกอบหลายๆ ส่วน ที่เรียกว่า Openstack component โดยในรุ่น Icehouse ประกอบด้วย Component ดังนี้

- Horizon (Dashboard Service): เป็นส่วนที่จัดการเกี่ยวกับ Web interface ทั้งหมดในบริการ OpenStack
- Nova (Compute Service): ทำหน้าที่ควบคุม VM โดยจะใช้ Hypervisor เป็น VMWare, Hyper-V หรือ ตัวอื่นๆ ได้ แต่โดยทั่วไปจะใช้ KVM เพราะฟรีและมีใน Linux อยู่แล้ว
- Neutron(Networking Service): ทำหน้าที่เป็น Virtual network เพื่อให้เราสามารถเข้าใช้งานเครื่อง VM ที่สร้างขึ้นมาได้
- Swift: เป็น Object Storage ประมาณว่าเป็นไดเรคทอรี่ ที่สามารถเอาไป mount กับเครื่อง VM เพื่อให้เข้าถึงไฟล์ และใช้จัดเก็บไฟล์ได้
- Cinder: เป็น Block storage คือ มันจะสร้างฮาร์ดดิสก์เสมือนเพื่อให้เราใช้ติดตั้ง OS ลงในเครื่อง VM ที่สร้างขึ้นมา และสามารถเพิ่มdiskให้กับVMได้

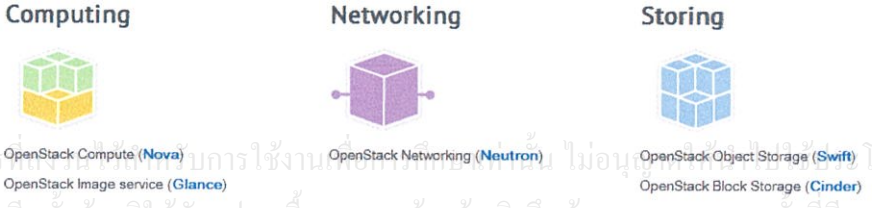
- Keystone(Identity Service): เป็นส่วนจัดการเรื่อง authentication และ authorization ทุกบริการของOpenstack
- Glance(Image Service): จะเป็นตัวจัดเก็บและเรียกดู virtual machine images และ OpenStack Compute จะนำไปใช้ในระหว่างการจัดเตรียม
- Ceilometer(Telemetry Service): ทำหน้าที่วัดการใช้งาน resources ไม่ว่าจะเป็น CPU, Memory, Storage และ Network หรือสถิติการใช้บริการต่างๆเช่น ค่าบริการ
- Heat(Orchestration service): ทำหน้าที่เป็น Orchestration Tools โดยเตรียม Config ต่างๆ ไว้เป็น Template ใช้ในการ Deploy App ได้ง่ายขึ้น
- Trove(Database service):ช่วยให้สามารถปรับขนาดและสร้างความน่าเชื่อถือของการบริการฐานข้อมูลบนคลาวด์ทั้งแบบrelational และ non-relational database



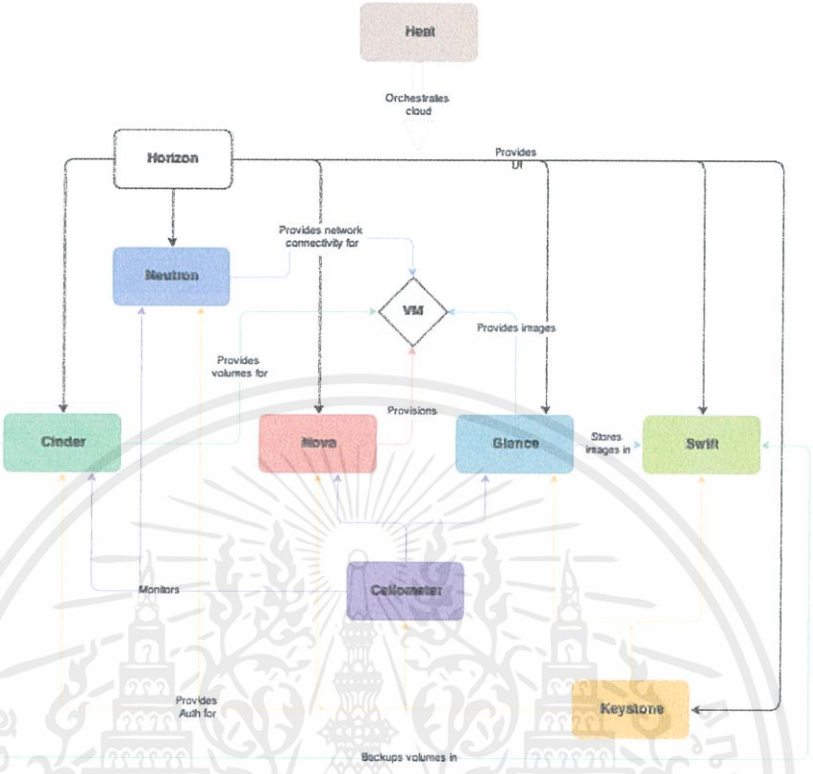
รูปที่ 2.17 รูปแบบการทำงานของ Openstack กับองค์ประกอบหลัก

2.3.2เกี่ยวกับOpenstack

OpenStack เป็นระบบปฏิบัติการแบบกลุ่มเมฆที่คอยควบคุม compute node ขนาดใหญ่, storage และทรัพยากร network ผ่านทาง datacenter ทั้งหมดถูกจัดการผ่านทาง dashboard ที่จะให้ผู้ดูแลเป็นคนควบคุมในขณะที่เพิ่มศักยภาพของผู้ใช้งานในการให้ใช้ทรัพยากรผ่านทาง web interface



รูปที่ 2.18 โครงสร้างหลักของ Openstack



รูปที่ 2.19 โครงสร้างของ Openstack

2.3.2.1 Keystone Concepts

OpenStack Identity Service (Keystones) จะเป็นจุดรวมสำหรับ policy, service catalog, token and authenticationสำหรับOpenStack clouds มันเป็นบริการที่ใช้ร่วมกันที่ให้บริการสนับสนุนสำหรับพื้นฐานโปรแกรม OpenStack อื่น ๆ เพื่อให้ฟังก์ชั่นเหล่านี้สามารถใช้เป็นศูนย์กลางสำหรับcloud ฟังก์ชั่นเหล่านี้คือ:

- Authenticate users และออก tokens สำหรับเข้าถึง services
- Authorize users ผ่าน role-based access control (RBAC)
- จัดให้มีรายการของการบริการ (API endpoints) สำหรับ cloud
- การสร้างและเก็บ policy ที่ให้บริการสามารถใช้เพื่อ authorize users ร้องขอ

OpenStack Identityจะมีหน้าที่อยู่4หัวข้อหลักคือ authorization (policy), authentication, token services และ service catalog ส่วนด้านล่างอธิบายแต่ละฟังก์ชันเหล่านี้

Authentication

เป้าหมายหลักของ Keystone มีการตรวจสอบผู้ใช้ โดยการตรวจสอบผู้ใช้จะได้การยอมรับการร้องขอการรับรองความถูกต้องและการตรวจสอบมันกับที่จัดเก็บอยู่ภายในแบ็กเอนด์เพื่อความถูกต้อง โดยทั่วไปจะหมายถึงการยอมรับ username และ password ที่ร้องขอจากผู้ใช้งาน และการตรวจสอบกับusername และ password ที่อยู่ในMySQL database อย่างไรก็ตามในขณะที่คุณจะเห็น Keystone Architecture Keystone สามารถใช้ความหลากหลายของแบ็กเอนด์เพื่อยืนยันตัวตนได้

Authorization

Authorization เป็นเรื่องที่สัมพันธ์กับ authentication หลังจากระบุผู้ใช้ ผู้ใช้จะได้รับอนุญาตให้ทำในสิ่งที่พวกเขาได้ถามว่าจะทำอย่างไร OpenStack จัดการ authorization ผ่าน rule-based engine ที่ควบคุมการกระทำของ API แต่ละบริการสามารถวางข้อจำกัด ในการเรียก APIใดๆ เพื่อที่จะสามารถได้รับการดำเนินการโดยบทบาทบางอย่างหรือผู้ใช้ ตัวอย่างนี้ถูกนำมาใช้อย่างกว้างขวางเพื่อให้ผู้ใช้เฉพาะกับ admin privileges เข้าถึงการตั้งค่าเรียก API

Tokens

Tokensเป็นศูนย์กลางของแผนการการตรวจสอบ OpenStack API เมื่อ Keystone ตรวจสอบข้อมูลประจำตัวของผู้ใช้ tokenชั่วคราวจะออกให้แก่ผู้ใช้ token นี้สามารถใช้ในการร้องขอบริการจากโปรแกรมOpenStack อื่นๆ โปรแกรม OpenStack อื่นๆ ตรวจสอบtokenได้โดยไม่ต้องประมวลผลหรือจัดการข้อมูลประจำตัวของผู้ใช้ บริการ OpenStack ยังสามารถใช้ token นี้เพื่อขอรับบริการจากโปรแกรม OpenStack อื่น ๆ ในนามของผู้ใช้

Tokens จะถูกเรียกชั่วคราวเพราะพวกเขาถูกตั้งค่าให้หมดหลังจากช่วงระยะเวลาหนึ่ง ช่วงเวลาสามารถกำหนดได้ แต่ทั่วไปจะตั้งที่ 1 วัน

Tokens ของ OpenStack สนับสนุนให้สามารถใช้สองรูปแบบคือ public key infrastructure (PKI) หรือ universally unique identifier (UUID) โดย UUID เป็นวิธีการเดิมใน Keystone รูปแบบของวิธีนี้จำเป็นต้องให้ผู้ใช้ในการส่งtokenที่มีแต่ละการร้องขอและโปรแกรม OpenStack เพื่อตรวจสอบความถูกต้อง token กับ Keystone แต่ละครั้ง ส่วนPKI (ถูกแนะนำใน OpenStack Grizzly) ช่วยให้ Keystone มี การเข้ารหัสแต่ละโทเค็น เมื่อผู้ใช้ไปใช้โทเค็น แต่ละบริการของ OpenStack มีการคัดลอกใบรับรองการเข้ารหัสลับของ Keystone เพื่อให้สามารถตรวจสอบภายในตัวมันเองได้ การเข้ารหัสนี้ช่วยลดการโหลดประมวลผลใน Keystone และทำให้ cloud มีความยืดหยุ่นมากขึ้น นอกจากความแตกต่างในการใช้งานแล้วยังแตกต่างกันอย่างมากใน

รูปแบบคือUUID เป็น 128 บิตตัวเลขในรูปแบบของ 7929c53d-c13f- 4ddd-988c-02bba7af852c ในขณะที่ PKI นี้จะมีความยาวที่ยาวมากๆของstring

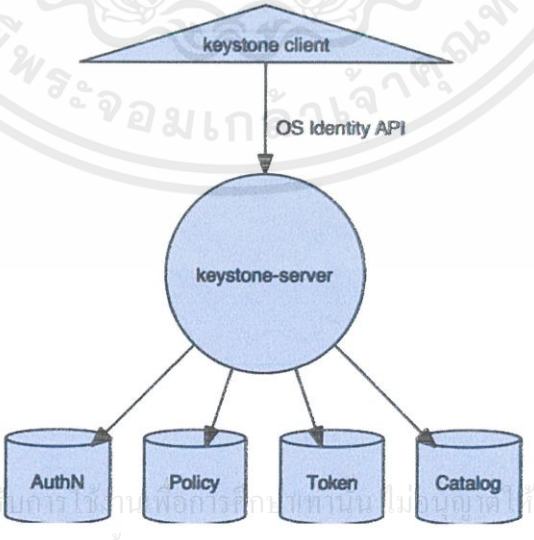
Service Catalog

ใน OpenStack Service Catalog มีการอ้างอิงถึงการนำเสนอบริการใน cloud การกรอกข้อมูลในservice catalog ประกอบด้วยชื่อ,ประเภท,คำอธิบาย,ปลายทางและภูมิภาคสำหรับการให้บริการ ลูกค้า OpenStack ใช้ข้อมูลนี้เพื่อขอรับบริการจากระบบ cloud

Keystone Architecture

สถาปัตยกรรม Keystone ง่ายมาก มันเป็นกระบวนการเดียวที่เชื่อมต่อกับกลุ่มเก็บข้อมูล Pluggable backend

- ไคลเอ็นต์ Keystone สามารถออกเรียก API ไปที่บริการ Keystone
- Keystone เซิร์ฟเวอร์จัดการการร้องขอ API รวมทั้งให้แคตตาล็อก ที่กำหนดนโยบายการให้บริการโทเค็นและตัวตน
- ฟังก์ชั่น Keystone แต่ละอย่างมี pluggable backend ซึ่งจะช่วยให้วิธีที่แตกต่างในการใช้บริการโดยเฉพาะ ส่วนแบ็กเอนด์มาตรฐานการสนับสนุนเช่น LDAP หรือ SQL เช่นเดียวกับที่ Key Value Stores (KVS)



รูปที่ 2.20 รูปแบบการทำงานของ Keystone

Keystone Clients

ไคลเอ็นต์ Keystoneเป็นซอฟแวร์ที่สามารถออกการเรียก OpenStack Identity API. Reference implementation เป็นโปรแกรมอินเตอร์เฟซบรรทัดคำสั่งที่เรียกว่า Python-keystoneclient อย่างไรก็ตามยังมีไลบรารีอื่น ๆ และเครื่องมือที่สามารถใช้ได้ ตัวอย่างเช่น OpenStack แดชบอร์ดที่สามารถใช้เป็นไคลเอ็นต์ Keystone สำหรับกิจกรรมบางอย่าง

เครื่องมือบรรทัดคำสั่งที่ยอมรับ (และPythonไลบรารี) เป็นPython-keystoneclient มันสามารถติดตั้งบนระบบที่มี Python interpreter ที่ติดตั้งผ่านทาง pip หรือ easy_install

```
$ sudo pip install python-keystoneclient
```

นอกจากการติดตั้งผ่านทาง pip ก็ยังสามารถใช้เป็นแพคเกจระบบส่วนใหญ่ ระบบปฏิบัติการลินุกซ์ สำหรับวัตถุประสงค์ของเรา Python keystoneclient มีการติดตั้งกับเซิร์ฟเวอร์ Keystone. Source code สำหรับ Python keystoneclient สามารถดูได้ที่ https://github.com/openstack/python-keystoneclient.

Keystone Server

กระบวนการ Keystone (Keystoneเซิร์ฟเวอร์) เป็น daemon อยู่บนพื้นฐานของไลบรารี Python Paste มันรับและตอบสนองต่อการเรียก API โดยเชื่อมต่อกับหนึ่งในแบ็กเอนด์ที่อธิบายด้านล่าง

ซึ่งเราสามารถคอนฟิกให้ยอมรับ APIs ที่แตกต่างกัน ซึ่งรวมถึง OpenStack Identity API, a privileged Admin API หรือหนึ่งใน extentionจำนวนมากรวมทั้ง Amazon Web Service (AWS) S3 และ EC2 เช่นเดียวกับการการมีส่วนร่วม Community อื่น ๆ ในรุ่นฮาวานา Keystone ใช้รุ่น 2.0 และ 3.0 ของ OpenStack Identity API เช่นเดียวกับ Admin API โดยค่าเริ่มต้น

Keystone Backends

นโยบายการจัดเก็บแบ็กเอนด์ Keystone โทเค็น การตรวจสอบ และข้อมูลการอนุญาต สำหรับ Keystone สำหรับการติดตั้งส่วนใหญ่ตัวเลือก SQLจะใช้สำหรับแบ็กเอนด์ทั้งหมด ซึ่งเก็บข้อมูลทั้งหมดของ Keystone ในฐานข้อมูลเดียว อย่างไรก็ตามยังมีตัวเลือกอื่น ๆ

- KVS การดำเนินการจัดเก็บค่าในหน่วยความจำที่สำคัญโดยใช้โครงสร้าง Dict Python
- Memcache เป็นที่นิยมกระจายค่าคีย์แคชที่เก็บ

- PAM เป็นผู้จัดการตรวจสอบดีฟอลต์ Pluggable ลินุกซ์ที่ช่วยให้ clouds เพียงแค่ตรวจสอบกับไฟล์ password เซิร์ฟเวอร์
- LDAP ซึ่งเป็นมาตรฐานมากที่สุดสำหรับการแก้ปัญหาตัวตนขององค์กร ตัวเลือก LDAP ยังมีการทำงานร่วมกับMicrosoft ของ Active Directory ผู้ประกอบการส่วนใหญ่จะใช้สิ่งนี้เป็นจุดของการปรับแต่งสำหรับการให้บริการพิสูจน์ตัวตนปัจจุบันของพวกเขา

Backend	Options
token	KVS, memcache and SQL
authentication	KVS, PAM, LDAP and SQL
policy	Rules and SQL
service catalog	KVS, SQL and template (file)

ตารางที่ 2.2 แสดงตัวเลือกสำหรับแต่ละแบ็กเอนด์ Keystone

เมื่อมีการใช้แบ็กเอนด์ SQL ก็สามารถช่วยในการทำความเข้าใจว่าวิธีที่ Keystone จะจัดเก็บข้อมูล ตารางด้านล่างแนวทางคีย์มาฐานข้อมูล Keystone กับคำอธิบายของแต่ละตาราง

Table	Description
credential	Stores non-EC2 credentials for the V3 API
domain	List of domains configured in Keystone. Will only contain the default domain for v2.0 API compatibility unless others are added.
ec2_credential	Contains the AWS EC2 credentials for user
endpoint	Details of all the endpoints including URL, region and IDs. There are usually three endpoints for each service: internal, public (external) and admin.

ตารางที่ 2.3 ฐานข้อมูล Keystone

Table	Description
group	List groups (collections of users)
group_domain_metadata	Table for relating group and domain metadata
group_project_metadata	Table for relating group and project metadata
migrate_version	Version of the database schema that Keystone expects as well as a path to the code repository for the migrations
policy	List of policies as JSON encoded blobs
project	Contains tenant information (tenants and projects are used interchangeably in OpenStack)
role	List of roles
service	List of services
token	List of tokens, their expiration dates and their validity
trust	List of delegates that can be used for orchestration and other long running tasks
trust_role	Roles for trusts
user	Users and encrypted passwords
user_domain_metadata	Table for relating user and domains metadata
user_group_membership	Table for relating user and groups metadata
user_project_metadata	Table for relating user and project metadata

ตารางที่ 2.3 ฐานข้อมูล Keystone

2.3.2.2 Glance Concepts

Glance ให้บริการจัดเก็บและการเรียกสำหรับvirtual disk images virtual disk images เหล่านี้ส่วนใหญ่มักจะใช้เป็น "ฮาร์ดไดรฟ์" สำหรับการประมวลของNova

Glance ถูกออกแบบมาเป็น REST-based web service ที่ช่วยให้คุณสามารถดำเนินการความหลากหลายของฟังก์ชั่นกับรายการของ images:

- การอัปโหลด virtual disk images ใหม่
- การเรียก virtual disk images
- รายชื่อที่มี virtual disk images
- ตั้งค่าสิทธิ์บน virtual disk images
- สอบถามคุณสมบัติของ virtual disk images

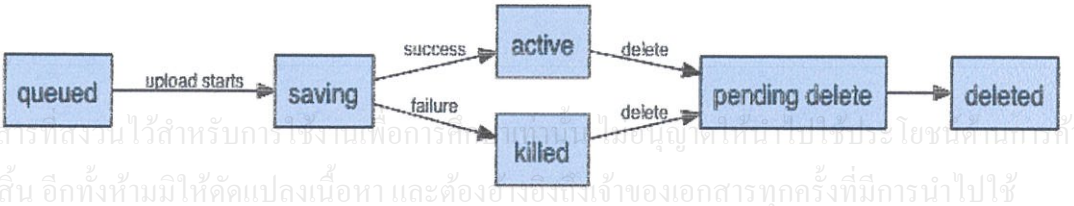
ในหลาย ๆ ด้าน Glanceสามารถคิดเป็นไฟล์เซิร์ฟเวอร์ที่มีความเชี่ยวชาญสูงแต่การที่เข้าใจความสามารถพิเศษของมันก็เป็นสิ่งสำคัญที่จะเข้าใจบางแนวคิดหลัก

Virtual Disk Images

Virtual disk images (หรือ "Images" ที่พวกเขาจะได้รับการเรียกว่าต่อจากนี้) เป็นไฟล์ (หรือกลุ่มของไฟล์) ที่เป็นตัวแทนของข้อมูลที่สมบูรณ์ของฮาร์ดไดรฟ์ทางกายภาพหรือ optical disk Imagesเหล่านี้มาในรูปแบบที่แตกต่างกันขึ้นอยู่กับข้อมูลและแหล่งที่สร้าง รูปแบบทั่วไปของImages ได้แก่ ISO (สำหรับแผ่นซีดีหรือดีวีดี), VMware VMDK (ไฟล์ suffixed กับ .vmdk) และ Xen และไมโครซอฟท์ (suffixed กับ .vhd) ในcloud computing imagesเหล่านี้เป็นฮาร์ดไดรฟ์เสมือนสำหรับกรณีการประมวลผล

Imagesสามารถสร้างขึ้นได้ผ่านจำนวนvirtualization (ทั้งserver applicationเช่น VMware vSphere เช่นเดียวกับdesktop applicationsเช่น VirtualBoxของOracle) เช่นเดียวกับ specialized utilities (เช่น XenConverter หรือ virt-install) นอกจากนี้ยังมีcommercial applicationsที่ให้การปรับแต่งImages

ภาพที่มีการดำเนินงานเฉพาะภายใน Glance



รูปที่ 2.21 รูปแบบขั้นตอนการทำงานของ Glance

Queued image ที่ได้รับการลงทะเบียนในรีจิสทรี แต่ไม่อัปโหลดไปที่image store.

Saving image จะถูกอัปโหลดไปเก็บที่glance image store.

Active image ได้รับการอัปโหลดลงไปในglanceและพร้อมสำหรับการใช้งานนี้เป็นขั้นตอนปกติสำหรับ images เฉพาะ images ในขั้นตอนนี้สามารถเรียกดูได้

Killed Error ที่เกิดขึ้นในการอัปโหลดและภาพอยู่ในสถานะที่ไม่สามารถใช้งาน

pending_delete imageที่ได้รับการกำหนดสำหรับการลบ แต่Glanceยังไม่ได้ลบออกจาก image file

deleted mageทั้งที่ถูกลบหรือทำเครื่องหมายที่จะถูกลบออกขึ้นอยู่กับว่า glance ได้รับการกำหนดค่าอย่างไร (ควบคุมโดย delayed_delete)

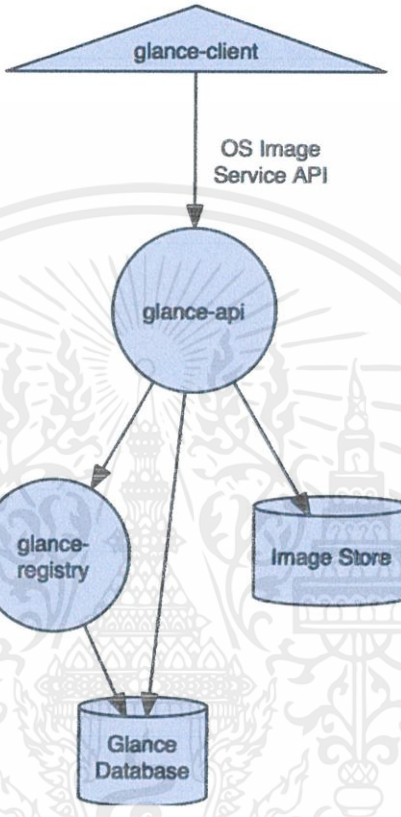
Image Formats

Glance สนับสนุนการจัดเรียงตามกว้างของ virtual disk และ container formats Virtual disksจะคล้ายกับ boot drives ของ physical server เพียงย่อลงไปในไฟล์ virtualization ที่แตกต่างกันก็จะสนับสนุนรูปแบบดิสก์ที่แตกต่างกัน

- ISO จะจัดเก็บรูปแบบสำหรับ optical disks
- VDI เป็นรูปแบบ Virtual disk image เกิดขึ้นโดยใช้ Oracle VM VirtualBox
- VHD เป็นรูปแบบที่พบมากที่สุดได้รับการสนับสนุนส่วนใหญ่โดยOpenStack virtualization ยกเว้น KVM
- Glance ยังสนับสนุนแนวคิดของ container formats ซึ่งอธิบายถึงรูปแบบไฟล์และข้อมูล metadata เพิ่มเติม Glance สนับสนุน 3 container formats เช่นเดียวกับการไม่มีของ container formats
- OVF มาตรฐานเปิดสำหรับการกระจายหนึ่งหรือมากกว่าหนึ่ง virtual machine images
- aki, ari, ami Amazon kernel, ramdisk หรือ machine image (ตามลำดับ)
- Bare ไม่มี container สำหรับ imageouh
- Docker container format ใหม่เพื่อรองรับการ Docker tarballs

Glance Architecture

มี 4 ส่วนสำหรับสถาปัตยกรรม Glance ได้แก่ glance-api , glance-registry, a database for the registry และ the image store



รูปที่ 2.22 รูปแบบการทำงานของ Glance

Glance API

Glance-API จะคล้ายคลึงกันในฟังก์ชันการทำงานที่ Nova-API หรือ Swift-Proxyในการที่จะยอมรับการร้องขอ API ที่เข้ามาแล้วสื่อสารกับส่วนประกอบอื่น ๆ (glance-registry and the image store) เพื่ออำนวยความสะดวกในการสอบถามเรียกอัพโหลดหรือลบ images. โดยค่าเริ่ม Glance-API บนพอร์ต TCP 9292

Glance Registry

Glance-registry ดำเนินการจัดเก็บและเรียกข้อมูลเกี่ยวกับ images รุ่นที่มาพร้อมกับ Glanceถือว่าเป็นเพียงการดำเนินการอ้างอิง ซึ่งเป็นที่เชื่อกันว่าการติดตั้งขนาดใหญ่ส่วนมากจะต้องการรุ่นที่กำหนดเองสำหรับการให้บริการของพวกเขา รุ่นอ้างอิงใช้ API Glanceสำหรับการสื่อสาร

สามารถเก็บข้อมูลในฐานข้อมูลใด ๆ ที่ได้รับการสนับสนุนโดย SQL-Alchemy โดยค่าเริ่มต้น glance-registry รอในพอร์ต TCP 9191

Glance Database

ฐานข้อมูล Glance มีเพียงไม่กี่ตารางอธิบายภาพและข้อมูลของมัน ในขณะที่ข้อมูลเกี่ยวกับภาพที่ถูกเก็บไว้ในฐานข้อมูลภาพที่เกิดขึ้นจริงจะถูกเก็บไว้ในที่เก็บภาพของ Glance

Table	Description
image_locations	Represents an image location in the datastore which supports multiple image locations (new in Havana)
image_members	Sharing information for images
image_properties	Custom (tenant-applied) image metadata
image_tags	Tags applied to images
images	Standard metadata for every image
migrate_version	Internal Glance table to increment the schema version

ตารางที่ 2.4 ฐานข้อมูล Glance

ฐานข้อมูลGlanceสามารถเป็นได้ทั้ง MySQL, Postgres หรือ SQLite โดย SQLite เป็นเพียงที่เหมาะสมสำหรับการทดสอบหรือหลักฐานของการใช้แนวคิด

Image Stores

Image Stores ที่เก็บภาพดิสก์เสมือน คือสถานที่ที่ไฟล์ภาพจะถูกเก็บไว้จริงสำหรับการดึงข้อมูลในอนาคต Glanceไม่จัดหาที่เก็บภาพ แต่จะใช้ประโยชน์จากรูปแบบอื่น ๆ ของการจัดเก็บไฟล์ การเก็บภาพที่ได้รับการสนับสนุนในปัจจุบันแสดงในตาราง

Image Store	Description
Fileystem	Stores, deletes, and gets images from a filesystem directory specified in the configuration file (filesystem_store_datadir option). This could be a filesystem on a shared drive (e.g., NFS).
HTTP	Retrieves images from a URL. This is a read-only image store option. Images will need to be saved to the URL via another mechanism.
Swift	Stores, deletes, and gets images from a Swift installation. Requires several configuration options in *glance.conf*.
S3	Uses Amazon's S3 service, similar in configuration and usage as the Swift option above.
RBD	Uses Ceph's Rado Block Device to store images
Cinder	Uses OpenStack Block Storage to hold images
Sheepdog	Uses Sheepdog, a distributed storage system for QEMU. More information about Sheepdog is available at http://sheepdog.github.io/sheepdog/.
GridFS	Stores images in "chunks" within a MongoDB collection

ตารางที่ 2.5 Image store ของ Glance

แต่ละตัวเลือกเหล่านี้มีจุดแข็งและจุดอ่อนของตนเอง อย่างไรก็ตามการติดตั้งขนาดใหญ่ส่วนมากจะใช้ Swift หรือ RBD (Ceph) ขึ้นอยู่กับเทคโนโลยีการจัดเก็บวัตถุของพวกเขา ในขณะที่การติดตั้งขนาดเล็ก อาจจะใช้ความเรียบง่ายของตัวเลือกระบบแฟ้มบนเซิร์ฟเวอร์ NFS ที่ใช้ร่วมกัน S3 หรือ HTTP การเก็บภาพที่น่าจะเป็นประโยชน์สำหรับการอ้างอิงภาพที่มีอยู่ทั่วไป

2.3.2.3 Cinder Concepts

Cinder มี Block storage เป็นบริการ Block storage ให้ raw volumes ที่สามารถสร้าง ขึ้นเปลี่ยนเป็น logical volumes และ File Systems ที่จะติดตั้งใน virtual machines ของคุณ ใน ในขณะที่ไม่จำเป็นสำหรับทุก cloud มันจะจัดหาวิธีที่ง่ายที่สุดที่จะยังคงมีข้อมูลอยู่ระหว่างอินสแตนซ์

Cinder มีคุณสมบัติหลายประการสำหรับ Block storage

- การสร้างและการลบ volumes
- แนบและถอด volumes จากอินสแตนซ์การประมวลผล
- การสร้างและลบ snapshot ของ volumes
- สร้าง volumes ใหม่จาก snapshot
- Clone volumes
- การทำงานร่วมกันกับ Glance: คัดลอกimagesไปยังvolumesและvolumesไป images
- ดูสถิติของ volumes

Cinder มีแนวความคิดที่สำคัญหลายอย่างที่มีความสำคัญในการทำความเข้าใจเพื่อที่สามารถ นำไปใช้และใช้ได้อย่างประสบความสำเร็จ

Volumes

Volumes คือ raw block storage ที่สามารถเชื่อมต่อกับตัวอย่างที่มีการจัดเก็บข้อมูลให้อยู่ ถาวร ซึ่งแตกต่างจากที่จัดเก็บชั่วคราวในกรณีปริมาณข้อมูลที่ยังคงอยู่ระหว่างการสร้างตัวอย่างและ การลบและสามารถติดตั้งบนตัวอย่างที่แตกต่างกัน

Volumes มีการบอกว่าตัวอย่างที่เป็น raw block storage มีความหมายว่าพวกเขาจะต้องมี การจัดรูปแบบและจากนั้นติดตั้งอยู่ภายในระบบปฏิบัติการก่อนที่จะใช้ อย่างไรก็ตาม นี้จะให้ผู้เช่าที่มี ความยืดหยุ่นที่ดีที่พวกเขาสามารถใช้ระบบปฏิบัติการใดสนับสนุน filesystem ที่พวกเขาเลือก

Snapshots

Snapshots คือการคัดลอกของ volume ที่จุดใดจุดหนึ่งของเวลา Snapshots ไม่สามารถ นำมาใช้เป็น volume โดยตรงได้ อย่างไรก็ตาม Snapshots สามารถใช้ในการสร้าง volume ใหม่ที่ สามารถนำมาใช้ได้ Snapshots อ่านได้อย่างเดียว

Backups

Backups คือการคัดลอกเต็มรูปแบบของข้อมูลของ volume ใด ๆ และจะถูกเก็บไว้ในการจัดเก็บสำรองข้อมูลที่สนับสนุน Backups สามารถถูกเรียกคืนไปเป็น volume แบบเดิมหรือ volume ที่มีอยู่อื่น ๆ ที่มีขนาดต่ำสุดของ volume เดิม

Backups มีความแตกต่างจาก Snapshotsคือการที่ Backups มีเพียงข้อมูลจาก volume ไม่ได้ทั้งหมดของ volume ตัวอย่างเช่นปริมาณ 10 GBที่มีข้อมูลเพียง 100MB ก็จะมีการ Backups แค่100MB เท่านั้นขณะที่โคลนหรือ Snapshots จะใช้ทั้งหมด 10 GB

Storage Providers

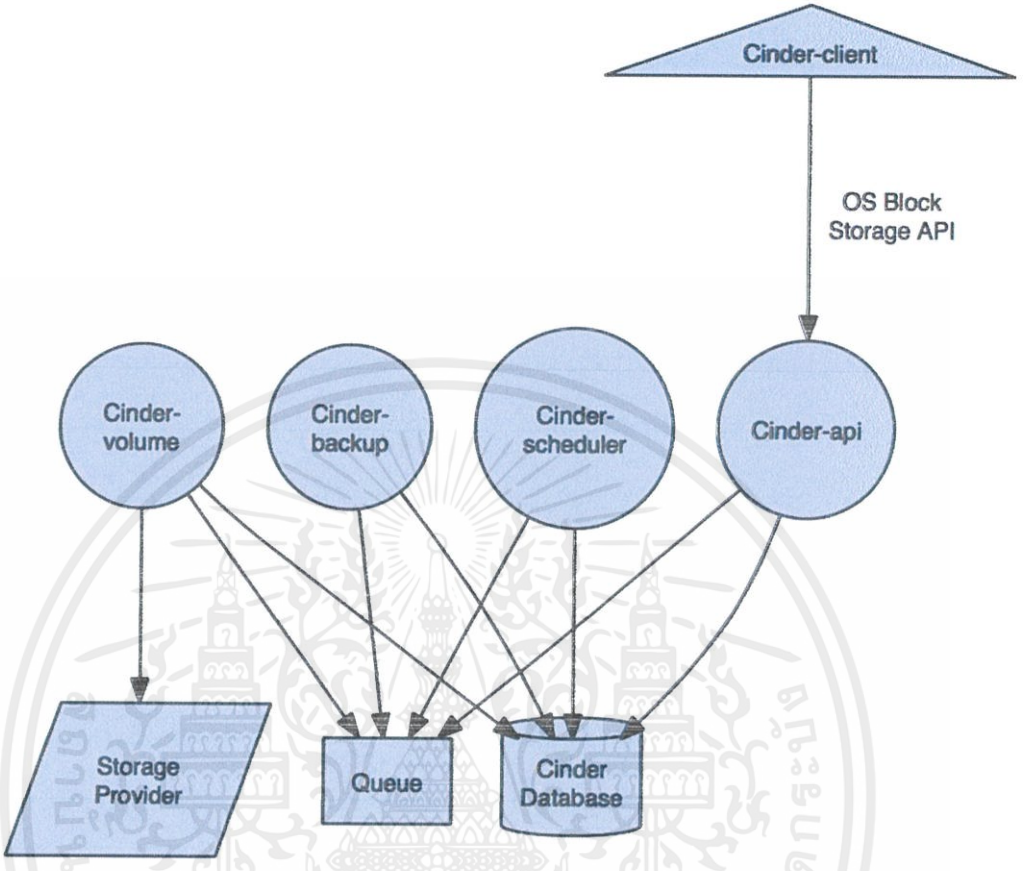
Cinder มีblock storageเป็นบริการโดยมีผู้ให้บริการ orchestrating underlying storage ผู้ให้บริการการจัดเก็บข้อมูลเหล่านี้จะเสนอvolumeเพื่อให้บริการอื่น ๆ (เช่น OpenStack Compute) ผ่านทางโปรโตคอลเครือข่ายเช่น iSCSI หรือ NFS

Volume Types

Volume Types จะช่วยให้ Cinder operators แสดงผู้ให้บริการการจัดเก็บข้อมูลแบ็กเอนด์ที่หลากหลายให้แก่ผู้เช่า ผู้เช่าจะสามารถเลือก Volume Types (แบ็กเอนด์) ที่พวกเขาชอบ มักจะถูกใช้โดยผู้ประกอบการที่จะให้ระดับการให้บริการที่หลากหลายแก่ผู้เช่า (บางครั้งมีค่าใช้จ่ายแตกต่างกัน)

Cinder Architecture

สถาปัตยกรรมของ Cinder จะตามบริการของ OpenStack ซึ่งมีการใช้ user-exposed API และ vendor provider plugins ภายใต้มัน



รูปที่ 2.23 รูปแบบการทำงานของ Cinder

Cinder Architecture

องค์ประกอบในสถาปัตยกรรมแต่ละส่วนจะอธิบายไว้ด้านล่าง

Cinder API

Cinder-api daemon ยอมรับการร้องขอ OpenStack Block Storage Service API และเส้นทางไปยัง cinder-volume สำหรับการดำเนินการ นอกจากนี้ยังจัดการพิสูจน์ตัวตนโดยการกำหนดเส้นทางการร้องขอไปยัง Keystone

นอกเหนือจาก OpenStack API ที่เป็นทางการ Cinder-api ยังสามารถรองรับส่วนขยายที่ API ตัวอย่างเช่นคำสั่งการสำรองข้อมูลจะถูกนำมาใช้เป็นส่วนขยายบริการนี้รอในพอร์ต TCP 8776 โดยค่าเริ่มต้น

Cinder Volume

Cinder-volume deamon ทำหน้าที่ตามคำขอ API จาก cinder-api โดยการอ่านหรือเขียนไปยังฐานข้อมูล Cinder เพื่อที่จะรักษาสถานะปฏิสัมพันธ์กับกระบวนการอื่น ๆ (เช่น cinder-scheduler) ผ่านคิวข้อความและทางตรงกับการเก็บรักษาบล็อกให้ฮาร์ดแวร์หรือซอฟต์แวร์

ตามที่อธิบายไว้ในผู้ให้บริการจัดเก็บข้อมูล Cinder-volume ใช้สถาปัตยกรรมควบคุมเพื่อสนับสนุนการจัดเก็บข้อมูลที่แตกต่างกัน นอกจากนี้ยังสามารถสนับสนุนผู้ให้บริการหลายระดับที่แตกต่างกันของการให้บริการหรือการทำงาน

Cinder Scheduler

Cinder-scheduler daemon ถูกออกแบบมาเพื่อรับโหนดให้บริการจัดเก็บบล็อกที่ดีที่สุดในการสร้าง volume โดยเริ่มต้นกำหนดการที่ใช้ในการกรองและการชั่งน้ำหนักขั้นตอนวิธีกำหนดการที่มีจุดมุ่งหมายเพื่อกระจาย volumes ทั่วโฮสต์ทั้งหมดอย่างเท่าเทียมกัน กำหนดการนี้จะกรองโหนดการจัดเก็บตามการใช้ประโยชน์ หรือ volumes ประเภทนั้นเรียงลำดับตามน้ำหนัก เช่น พื้นที่ว่าง อย่างไรก็ตามมันสามารถตั้งค่าและขั้นตอนวิธีกำหนดการอื่น ๆ ที่สามารถนำมาใช้

Cinder Backup

cinder-backup เป็นบริการใหม่ที่สำรองข้อมูลจากไดรฟ์ (not a full snapshot) การให้บริการแบ็กเอนด์ โดยปกติไดรฟ์ได้รับการสนับสนุนขึ้นอยู่กับ OpenStack Swift

Messaging Queue

การใช้ Cinder จะทำให้การใช้งานของคิวการส่งข้อความไปยังข้อมูลที่เส้นทางระหว่างกระบวนการ Cinder. RabbitMQ หรือ Apache Qpid สนับสนุนคิวข้อความ

Cinder Database

ฐานข้อมูล Cinder ยังคงรักษาสถานะของ Cinder และที่เกี่ยวข้องกับวอลุ่มการสำรองข้อมูลภาพรวมและการให้บริการ รายการทั้งหมดของตารางและฟังก์ชั่นของพวกมันมีรายละเอียดในตาราง

Table	Description
backups	List of backups, their status and associated metadata such as size and container
encryption	Stores information about volume encryption ciphers, keys and providers
iscsi_targets	Mapping between volumes and iSCSI targets
migrate_version	Version of the database schema that Cinder expects as well as a path to the code repository for the migrations
quality_of_service_specs	Key/value pairs for volume types
quota_usages	Quota usage by tenant
quotas	Quota configurations by tenant
reservations	Preallocation lists
services	Registry of Cinder service components (cinder-volume, cinder-scheduler, etc daemons)
snapshot_metadata	Snapshot metadata key/value pairs
snapshots	List of snapshots including size, status and name
volume_metadata	List of key/value pairs describing volumes
volume_type_extra_specs	Key/value pairs for volume types
volume_types	List of volume types

ตารางที่ 2.6 ฐานข้อมูลของ Cinder

Table	Description
volumes	List of volumes including provider specific data, source, host and status

ตารางที่ 2.6 ฐานข้อมูลของ Cinder

2.3.2.4 Neutron Concepts

Neutron ให้บริการด้าน network ระหว่างอุปกรณ์อินเตอร์เฟซที่จัดการตามการให้บริการ OpenStack อื่น ๆ (ส่วนมาก Nova) มันช่วยให้ผู้ใช้:

- ผู้ใช้สามารถสร้างเครือข่ายของตัวเองและจากนั้นแนบการเชื่อมต่อเซิร์ฟเวอร์กับพวกเขา
- สถาปัตยกรรมแบ็กเอนด์แบบเสียบได้ช่วยให้ผู้ใช้ใช้ประโยชน์จาก commodity gear หรืออุปกรณ์ที่ผู้ขายได้รับการสนับสนุน
- การขยายอนุญาตให้เพิ่มบริการเครือข่ายซอฟต์แวร์หรือฮาร์ดแวร์ที่จะบูรณาการ

แกนหลักของNeutron APIประกอบด้วยการสนับสนุนLayer 2 networkingและการจัดการ IP address(IPAM) เช่นเดียวกับส่วนขยายสำหรับ Layer 3 โครงสร้างเราเตอร์ที่ช่วยให้สามารถ กำหนดเส้นทางระหว่างLayer 2 networks และ gatewaysไปยังnetworkภายนอก Neutron ประกอบด้วยรายการที่เพิ่มขึ้นของปลั๊กอินที่ช่วยให้การทำงานร่วมกับความหลากหลายในเชิงพาณิชย์ network เทคโนโลยีที่เป็น open source รวมทั้ง routers,switches,virtual switches และ software-defined networking (SDN) controllers

Ports

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

Networks

Neutron กำหนด network คือที่แยกได้ Layer 2 network segments ผู้ควบคุมจะเห็น เครือข่ายที่เป็น logical switches ดำเนินการโดย Linux bridging tools, Open vSwitch หรือบาง ซอฟแวร์อื่น ๆ ซึ่งแตกต่างจากเครือข่ายทางกายภาพที่จะถูกกำหนดโดยทั้งผู้ควบคุมหรือผู้ใช้ของ OpenStack

Subnets

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

Routers

เกตเวย์ระหว่างเครือข่าย

Private and Floating IPs

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

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 ระหว่างการประมวลผลหลายอินสแตนซ์

Vendor enhancements

Network Namespaces

FIXME Network namespaces แยกอินสแตนซ์ของ network interfaces และตารางกำหนดเส้นทางที่ทำงานอิสระจากกัน

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

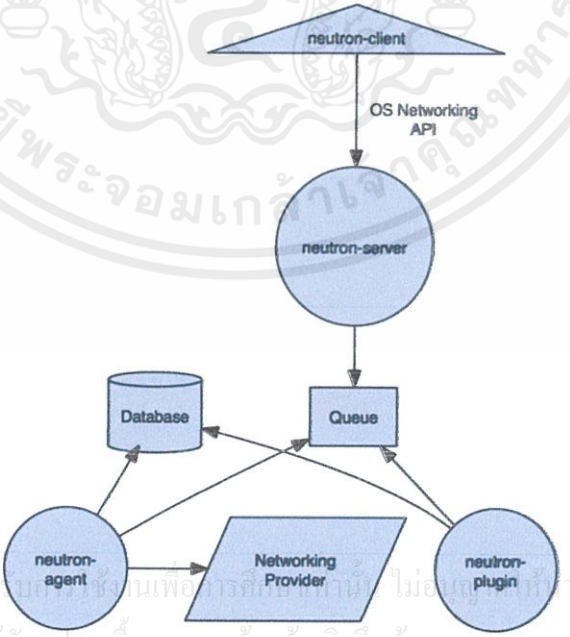
FIXME ประโยชน์ใหญ่ของการดำเนินการของ namespaces ใน neutron คือการที่ผู้เช่าสามารถสร้างทับซ้อนกันของ IP address สถานการณ์ที่ให้อิสระกับผู้ใช้ระบบคลาวด์เพราะพวกเขามีอิสระที่จะสร้างเครือข่ายย่อยใดๆ ของทางเลือก โดยไม่ต้องกลัวที่ขัดแย้งกันกับที่ของผู้เช่าอื่นๆ Linux network namespace จะต้องทำงานบนโหนด neutron-l3-agent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

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)

Neutron Architecture

เช่นเดียวกับหลายบริการ OpenStack, นิวตรอนเป็นการตั้งค่าชั้นสูงเนื่องจากเป็นสถาปัตยกรรมแบบปลั๊กอิน ปลั๊กอินเหล่านี้รองรับอุปกรณ์เครือข่ายและซอฟต์แวร์ที่แตกต่างกัน ดังนั้นสถาปัตยกรรมและการใช้งานอาจแตกต่างกันอย่างมาก



รูปที่ 2.24 รูปแบบการทำงานของ Neutron

- นิวตรอนเซิร์ฟเวอร์ยอมรับการร้องขอ API และจากนั้นหาเส้นทางที่ไปยังปลั๊กอินนิวตรอนที่เหมาะสมสำหรับการดำเนินการ
- ปลั๊กอินนิวตรอนและตัวแทนดำเนินงานที่เกิดขึ้นจริงเช่นการเสียบและถอดปลั๊กพอร์ต, การสร้างเครือข่ายหรือเครือข่ายย่อยและไอพีแอดเดรส ปลั๊กอินและตัวแทนเหล่านี้แตกต่างกันไปขึ้นอยู่กับผู้จำหน่ายเทคโนโลยีที่ใช้ในระบบคลาวด์โดยเฉพาะ
- ส่วนใหญ่การติดตั้งนิวตรอนจะทำให้การใช้งานของคิวการส่งข้อความไปยังข้อมูลที่เส้นทางระหว่างตัวแทนนิวตรอนและเซิร์ฟเวอร์ต่างๆเช่นเดียวกับฐานข้อมูลในการเก็บสถานะเครือข่ายสำหรับปลั๊กอินโดยเฉพาะ

Quantum server

- Implement Qauntum API and its extensions
- Enforce network model
- Network, subnet, and port
- IP addressing to each port

Plugin agent

- Run on each compute node
- Connect instances to network port
- Allow Neutron to dictate network policy to Open vSwitch

DHCP agent

- In multi-host mode, run on each compute node (deferred)
- Start/stop dhcp server
- Maintain dhcp configuration
- Provide DHCP addressing to the Instances

L3-agent

- To implement floating Ips and other L3 features, *such as NAT*
- One per network
- Provide L3 Services to the OpenStack Instances

Metadata-agent

- Mediate between quantum L3-agent, DHCP agent with Openstack nova metadata API server

ovs-switchd

- Open vSwitch – provide networking for the Instances

นิวตรอนจะโต้ตอบส่วนใหญ่กับโนวาที่จะให้เครือข่ายและการเชื่อมต่ออินสแตนซ์ของมัน

2.3.2.5 Nova Concepts

Nova มีสิ่งอำนวยความสะดวกในการจัดหาและจัดการvirtual machine instancesที่คล้ายกันในการทำงานและขอบเขตของการให้บริการของ Amazon EC2 มันช่วยให้คุณสามารถสร้าง จัดการและลบเซิร์ฟเวอร์เสมือนขึ้นอยู่กับmachine images ที่อยู่ใน Glance ผ่าน programmable API มันเป็นสิ่งที่ใหญ่ที่สุดและที่นิยมมากที่สุดและมีความซับซ้อนมากที่สุดของทุกบริการ OpenStack

- คุณสมบัติ
- การทำงานร่วมกับโปรแกรมอื่น ๆ OpenStack

Instances

OpenStack Compute จะจัดหาเครื่องเสมือนตามความต้องการสำหรับผู้ใช้ เครื่องเสมือนหรืออินสแตนซ์ที่พวกเขาจะถูกเรียกใน Nova parlanceสามารถถูกควบคุมผ่านทางเรียก API ไปยัง OpenStack Compute เนื่องจากการเรียก API เหล่านี้ ผู้ใช้สามารถเริ่มต้นการกำหนดIP addresses, เพิ่มที่จัดเก็บข้อมูลเพิ่มเติมหรือเข้าใช้อินสแตนซ์คอนโซลของพวกเขา

Ephemeral Storage

- การจัดเตรียมส่วนหนึ่งของระบบดิสก์
- หายไปหลังจากการสิ้นสุดของ instance
- การปรับขนาดภาพใน Glance

Metadata

- การส่งข้อมูลไปinstanceที่เกี่ยวกับตัวเอง (IP address, root password, ssh keys, etc.)
- Metadata API
- การกำหนดค่าไดรฟ์

Keys

- special metadata
- ssh key สำหรับการเข้าสู่ระบบ

Snapshot

คล้ายกับความสามารถใน Cinder ที่snapshot volumes Nova จะช่วยให้ผู้เช่าที่จะ snapshot instance ที่กำลังทำงานของพวกเขา Nova จะควบคุมเหล่านี้ผ่าน

- qemu-img
- ส่งไปที่ glance
- glance จะเก็บมันไว้ใน swift

Hypervisors อื่นๆ จะจัดการกับเครื่องมือที่เป็นกรรมสิทธิ์ของพวกเขา ตัวอย่างเช่น XenServer ใช้ XenAPI VDI.snapshot (คล้ายกับบรรทัดคำสั่ง xe snapshot)

Hypervisors

hypervisors ให้พื้นฐานสำหรับคุณสมบัติการทำงานแบบเสมือนของ Nova และยังเป็นผู้ให้บริการโครงสร้างพื้นฐานสำหรับคุณสมบัติของNova Nova สนับสนุนจำนวนของ hypervisors ที่แตกต่างกันและยังสามารถสนับสนุน hypervisors หลายๆภายในcloudเดียว Nova มีการโต้ตอบกับ hypervisors ที่มีความหลากหลายของอินเตอร์เฟซ

Flavors

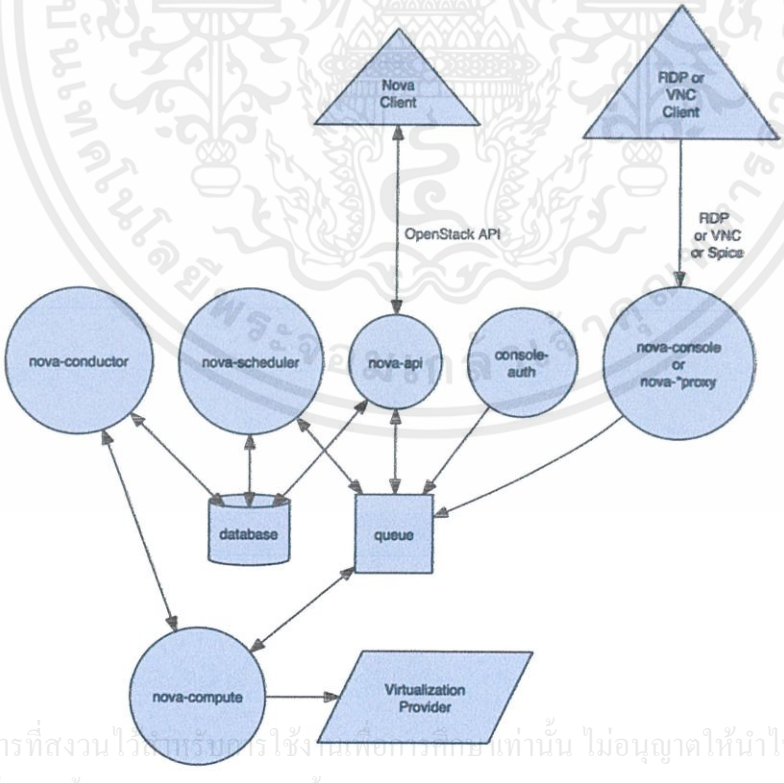
Flavors (ยังเรียกอีกอย่างว่าประเภท instance ในบางส่วนของรหัส) คือขอบเขตของ instance โนวากำหนด Flavors ใน:

- Memory
- Virtual CPUs (vCPUs)
- Disk space

Flavors ถูกกำหนดโดยผู้ควบคุม cloud สำหรับผู้เช่า Flavors ยังสามารถปรับแต่งให้สำหรับผู้เช่าที่เฉพาะเจาะจง

Nova architecture

Novaมีความสัมพันธ์กับหลายบริการ OpenStack อื่นๆ Keystone ใช้สำหรับการตรวจสอบ Glance ใช้สำหรับ images และ Horizon สำหรับอินเตอร์เฟซเว็บ การสื่อสารของ Glance เป็นศูนย์กลาง กระบวนการ API สามารถอัปโหลดและค้นหา Glance ในขณะที่ nova-computeจะดาวน์โหลด imagesเพื่อใช้ในการเปิด images



รูปที่ 2.25 รูปแบบการทำงานของ Nova

แผนภาพที่ซับซ้อนนี้สามารถสรุปได้ในสามประโยค:

- ผู้ใช้ปลายทางที่ต้องการใช้ Nova ที่จะสร้าง compute instancesเรียก Nova-API กับ OpenStack API หรือ EC2 API requests
- Nova daemons แลกเปลี่ยนข้อมูลผ่านqueue (actions) และ database (information)เพื่อดำเนินการ API requests
- Glance เป็นบริการที่แยกจากกันอย่างสมบูรณ์ที่ Nova interfacesผ่าน Glance API เพื่อให้บริการ virtual disk imaging services

ตอนนี้ที่เราได้เห็นภาพรวมของกระบวนการและการมีปฏิสัมพันธ์ของเหล่านี้ลองมามองในแต่ละองค์ประกอบ

Nova Clients

- CLI
- Graphical
- Other OpenStack programs like Horizon and Heat

API

nova-api daemonเป็นหัวใจของ Nova คุณอาจจะเห็นมันแสดงให้เห็นในรูปภาพจำนวนมากของNova เป็น API และ " Cloud Controller" ขณะนี้ส่วนหนึ่งที่เป็นจริงที่ cloud controller เป็นจริงเพียงclass (โดยเฉพาะ CloudController ใน nova/api/ec2/cloud.py) ภายใน nova-api daemon วัตถุประสงค์หลักของมันคือการยอมรับและตอบสนองการร้องขอ API ที่เข้ามา

หากต้องการยอมรับและตอบสนอง API requests, nova-api จะให้ปลายทางสำหรับทุก API queries (ยอมรับการร้องขอโดยใช้ OpenStack APIหรือ Amazon EC2 API) เริ่มต้นส่วนใหญ่ของ orchestration activities (เช่นรันอินสแตนซ์) และยังบังคับใช้บางส่วน นโยบาย (การตรวจสอบโควต้าส่วนใหญ่) สำหรับการร้องขอบางอย่างก็จะตอบสนองความต้องการทั้งหมดของตัวเองโดยการสอบถามฐานข้อมูลแล้วก็รีเทิร์นกลับด้วยคำตอบ สำหรับการร้องขอที่ซับซ้อนมากขึ้นก็จะส่งข้อความไปยัง daemons อื่น ๆ ที่ผ่านการรวมกันของการเขียนข้อมูลไปยังฐานข้อมูลและการเพิ่มข้อความไปยังคิว

Scheduler

กระบวนการ nova-scheduler แนวคิดของชิ้นส่วนของรหัสที่ง่ายใน Nova ที่ใช้ virtual machine instance request จากคิวและกำหนดที่มันควรจะทำงาน (โดยเฉพาะที่ compute server host ที่ควรทำงาน) ในทางปฏิบัติอย่างไรก็ตามนี้จะเติบโตที่จะเป็นชิ้นส่วนที่ซับซ้อนที่สุด เท่าที่มันจะต้องมีปัจจัยในสถานะปัจจุบันของโครงสร้างพื้นฐานคลาวด์ทั้งหมดและใช้อัลกอริทึมที่ซับซ้อนเพื่อให้แน่ใจว่าการใช้งานที่มีประสิทธิภาพ ไปสิ้นสุด nova-scheduler ดำเนินการ pluggable architecture ที่ช่วยให้คุณเลือก (หรือเขียน) อัลกอริทึมของคุณเองสำหรับการจัดตาราง รายละเอียดการเลือกการจัดตารางเวลาในปัจจุบัน

Scheduler	Notes
Simple	พยายามที่จะหาพื้นที่โหลดน้อย
Chance	เลือก hostที่มีการสุ่มจากตาราง นี่คือการจัดตารางเวลาเริ่มต้น
Zone	หยิบโฮสต์สุ่มจากภายในavailability zone
Filter	FIXME

ตารางที่ 2.7 Scheduler ของ Nova

Nova Conductor

บริการใหม่ที่เรียกว่า nova-conductor มันหน้าที่เป็นสื่อกลางการเข้าถึงฐานข้อมูลสำหรับ daemons อื่น ๆ (โดยเฉพาะ nova-compute ในรุ่นนี้) เพื่อให้มีการรักษาความปลอดภัยมากขึ้น

Compute Worker

กระบวนการ nova-computeเป็นสาเหตุหลัก worker daemon ที่สร้างและยุติ virtual machine instances ผ่าน APIs ของไฮเปอร์ไวเซอร์ (XenAPI สำหรับ XenServer / XCP, libvirt สำหรับ KVM หรือ QEMU, VMwareAPI สำหรับ VMware ฯลฯ) กระบวนการที่มันไม่ให้มีความซับซ้อนพอสมควร แต่พื้นฐานง่าย: ยอมรับการกระทำจากคิวแล้วดำเนินการชุดของคำสั่งระบบ (เช่น เปิดตัว KVM instance) ที่จะนำพวกเขาออกในขณะที่การปรับปรุงสถานะในฐานข้อมูลผ่าน nova-

conductor โปรดทราบว่าการใช้ nova-conductor เป็นตัวเลือก แต่ไม่ช่วยเพิ่มการรักษาความปลอดภัย

Network Worker

Nova-network worker daemon จะคล้ายกับ nova-compute และ nova-volume มันรับ networking tasks จากคิวแล้วดำเนินการคำสั่งระบบไปการจัดการเครือข่าย (เช่นการตั้งค่าการเชื่อมอินเตอร์เฟซหรือการเปลี่ยนแปลง iptables rules)

Nova กำหนดสองประเภทที่แตกต่างกันของ IP addresses สำหรับ instance Fixed IPs และFloating IPs ที่เหล่านี้สามารถจะคิดในวงกว้างของ IPs ที่เป็นส่วนตัว (fixed) และ IPs ที่สาธารณะ (floating) Fixed IPs จะได้รับมอบหมายในการเริ่มต้น instanceและยังคงเหมือนเดิมในช่วงทั้งหมดของอายุการใช้งาน Floating IPs จะจัดสรรแบบไดนามิกและที่เกี่ยวข้องกับโดเมนเพื่อให้การเชื่อมต่อภายนอก

Queue

คิวมีศูนย์กลางhubสำหรับการส่งข้อความระหว่าง daemons นี้มักจะนำมาใช้กับ RabbitMQ หรือ Apache Qpid แต่อาจจะเป็น AMPQ message queue หรือ Zero MQ

Nova สร้างหลายประเภทของคิวข้อความเพื่ออำนวยความสะดวกการสื่อสารระหว่าง daemonsต่างๆ เหล่านี้รวมถึง topics queues, fanout queuesและ host queues Topics queues ให้ข้อความที่จะถูกกระจายไปยังหมายเลขของชั้นโดยเฉพาะอย่างยิ่งของ worker daemons ตัวอย่างเช่น Nova ใช้เหล่านี้ในการส่งผ่านข้อความไปยังทุกคน (หรือใดๆ) ของ compute หรือ volume daemons นี้จะช่วยให้Novaที่จะใช้ worker ที่มีอยู่ก่อนไปประมวลผลข้อความ Host queues ช่วยให้ Nova ที่จะส่งข้อความไปยังบริการที่เฉพาะเจาะจงบนโฮสต์ที่เฉพาะเจาะจง ตัวอย่างเช่น Nova มักจะต้องส่งข้อความไปยัง specific host's compute worker ที่จะดำเนินการกับ particular instance

Database

SQL database stores ส่วนใหญ่สถานะของ build-time และ run-time สำหรับโครงสร้างพื้นฐานคลาวด์ ซึ่งรวมถึงประเภท instance ที่มีอยู่สำหรับการใช้งานในอินสแตนซ์การใช้เครือข่ายที่มีอยู่และโครงการ ในทางทฤษฎี OpenStack Nova ที่สามารถรองรับฐานข้อมูลใดที่ได้รับการสนับสนุนโดย SQL-Alchemy แต่เฉพาะฐานข้อมูลกำลังถูกใช้กันอย่างแพร่หลายเป็น sqlite3 (เฉพาะที่เหมาะสมสำหรับการทดสอบและการพัฒนา), MySQL และ PostgreSQL

Console Services

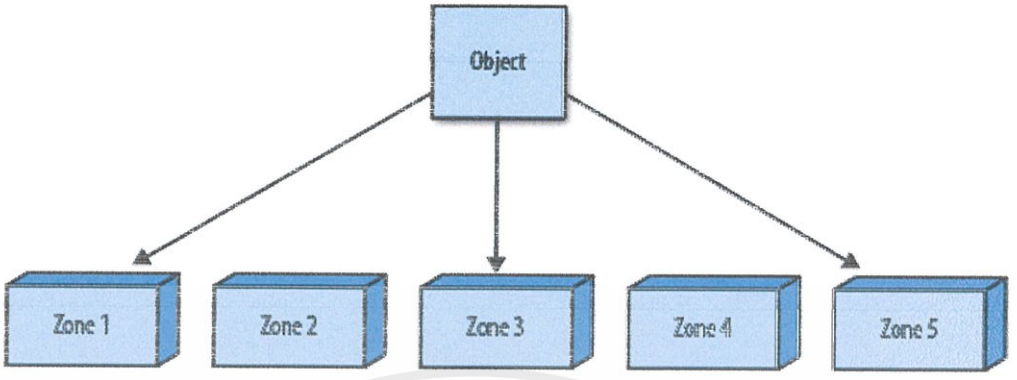
Novaยังให้บริการคอนโซลเพื่อให้ผู้ใช้ในการเข้าถึง virtual instance ของคอนโซลพวกเขาผ่านพร็อกซี่ นี้เกี่ยวข้องกับหลาย daemons (nova-console, nova-xvpvncproxy, nova-spicehtml5proxyและ nova-consoleauth)

2.3.2.6 Swift concept

Swift มีวัตถุประสงค์เพื่อให้มีการจัดเก็บวัตถุขนาดใหญ่ที่ปรับขนาดได้และซ้ำซ้อนโดยมีแนวคิดที่คล้ายกันกับการให้บริการของ Amazon S3 เพื่อให้swiftมีความยืดหยุ่นและความซ้ำซ้อนทำให้เขียนหลายสำเนาของแต่ละวัตถุไปยังหลายเซิร์ฟเวอร์ที่จัดเก็บข้อมูลภายในที่ต่างหาก "zone" Zoneเป็นกลุ่ม logic ของเซิร์ฟเวอร์ที่จัดเก็บข้อมูลที่ได้รับการแยกออกจากกันเพื่อป้องกันความผิดพลาด ระดับของการแยกจะขึ้นอยู่กับผู้ดูแลคลาวด์ สามารถแยกที่ความแตกต่างของเซิร์ฟเวอร์ (ความสามารถในการสูญเสียแต่ละเซิร์ฟเวอร์), ชั้นวางที่แตกต่างกัน (ความสามารถในการสูญเสียทั้งชั้น) ส่วนที่แตกต่างกันของศูนย์ข้อมูลหรือแม้กระทั่งศูนย์ข้อมูลที่แตกต่างกัน แต่ละทางเลือกมีระดับที่แตกต่างของการแยกและค่าใช้จ่าย

ผู้ใช้จำนวนมากคิดว่า Swift จะใช้แทนที่ของไฟล์เซิร์ฟเวอร์ของพวกเขาและพวกเขาจะสามารถ ติดตั้งvolumes ได้อย่างง่ายดายใน desktops ของพวกเขาที่จะเข้าถึงไฟล์ของพวกเขาเป็นวิธีที่ผิด Swift เป็น object storeไม่ใช่ file server ในขณะที่สิ่งเหล่านี้ดูเหมือนว่าจะคล้ายกันแต่มีความแตกต่างที่สำคัญ Object stores เพียงบันทึกไฟล์ในกลุ่ม logical (เรียกว่า"containers"ใน Swift parlance) ผ่านทาง RESTful protocol พวกเขาไม่ได้ให้ filesystem จริงพวกเขาไม่สามารถเข้าถึงได้ผ่านโปรโตคอลการแชร์ไฟล์มาตรฐานเช่น NFS (Network File System มาตรฐานสำหรับ UNIX), CIFS (Common Internet File System มาตรฐานสำหรับ Windows) หรือ AFS (AppleShare Files System มาตรฐานสำหรับ Mac OS X) ในการเข้าถึงไฟล์ของคุณคุณจะต้องใช้ Swift API client

Swift จะกำหนดค่าในข้อกำหนดของจำนวนสำเนา (เรียกว่า " replicas") ที่เขียนเช่นเดียวกับหลายๆวิธีการที่ zone ถูกกำหนดค่า การปฏิบัติที่ดีที่สุดในปัจจุบันคือให้ 3 replicas เขียนผ่าน 5 zone ขณะที่จำนวนของ replicas มีค่าน้อยกว่าหรือเท่ากับจำนวนของ zone Swift พยายามที่จะรักษาความสมดุลของการเขียนของ objects ไปยังเซิร์ฟเวอร์ที่จัดเก็บข้อมูลเพื่อให้การเขียนและโหลดอ่านถูกกระจายไป



รูปที่ 2.26 รูปแบบของ Swift

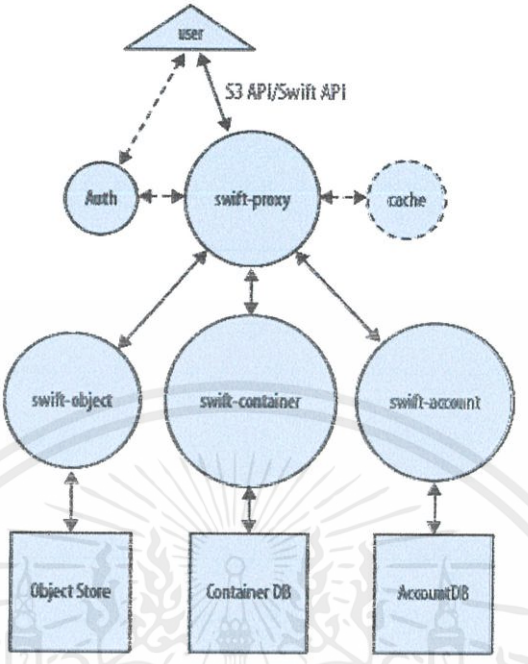
Swift Architecture

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

- เก็บและเรียก objects (file)
- ตั้งค่าและปรับเปลี่ยน metadata บน objects (แท็ก)
- *Versions objects*
- การให้บริการหน้าเว็บแบบคงที่และวัตถุผ่านทาง HTTP ในความเป็นจริงในแผนภาพบล็อกโพสต์นี้มีการทำหน้าที่ให้บริการ Swift ของ Rackspace

Swift architectureมีการกระจายมากที่จะป้องกันไม่ให้จุดเดียวของความล้มเหลวใด ๆ เช่นเดียวกับขนาดในแนวนอน จะรวมถึงองค์ประกอบต่อไปนี้:

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



รูปที่ 2.27 รูปแบบการทำงานของ Swift

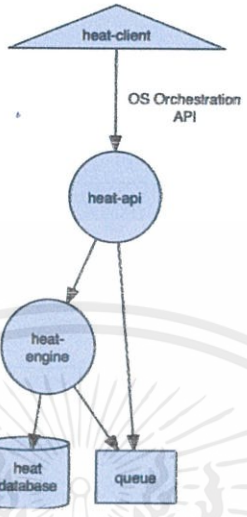
สวิฟท์ยอมรับการร้องขอผู้ใช้ปลายทางผ่านกระบวนการ swift-proxy Swift-Proxy ยอมรับการร้องขอผู้ใช้ปลายทางที่เข้ามา เลือกที่อนุญาตและรับรองความถูกต้องพวกเขา แล้วผ่านพวกเขาไปยังกระบวนการ object, accountหรือcontainerที่เหมาะสมสำหรับการการเสร็จสิ้น ก็สามารถเลือกที่จะทำงานกับแคช (memcached) เพื่อลดการตรวจสอบ,container และ account calls Swift-Proxyยอมรับการร้องขอผ่านทาง OpenStack API ในพอร์ต 80 นอกจากนี้ยังมีตัวกลางเป็นตัวเลือกที่จะสนับสนุนโปรโตคอล Amazon S3

2.3.2.7 Heat concept

Heat (Orchestration service): ทำหน้าที่เป็น Orchestration Tools โดยเตรียม Config ต่างๆ ไว้เป็น Template ใช้ในการ Deploy App ได้ง่ายขึ้น

OpenStack Orchestration เป็นแม่แบบที่ขับเคลื่อนที่ให้นักพัฒนาโปรแกรมประยุกต์สร้างและใช้งานโครงสร้างพื้นฐาน (infrastructure) ภาษาเทมเพลตที่มีความยืดหยุ่นสามารถระบุการประมวลผลการจัดเก็บข้อมูลและการกำหนดค่าเครือข่ายเช่นเดียวกับกิจกรรมการโพสต์รายละเอียดการใช้งานได้โดยอัตโนมัติเต็มรูปแบบของการจัดเตรียมโครงสร้างพื้นฐานสำหรับการให้บริการและการใช้งาน ในอย่างง่ายโปรแกรม OpenStack Heat คือการสร้างบริการ human- and machine-accessible สำหรับการจัดการวงจรชีวิตทั้งหมดของโครงสร้างพื้นฐานและการใช้งานภายใน

Heat Architecture



รูปที่ 2.28 รูปแบบการทำงานของ heat

- heat client
- heat-api
- heat-api-cfn
- heat-engine

2.3.2.8 Horizon concept

OpenStackDashboard (หรือตามที่มีโค้ดเนม Horizon) จัดเตรียมส่วนหน้าเว็บสำหรับการให้บริการ OpenStack

Horizon (Dashboard Service): เป็นส่วนที่จัดการเกี่ยวกับ Web interface ทั้งหมดในบริการ OpenStack

อินเตอร์เฟซ Horizon แบ่งออกเป็นสองส่วน ขึ้นอยู่กับผู้ใช้ในการเข้าถึงแดชบอร์ด หากผู้ใช้เป็นผู้เช่าตามปกติพวกเขาจะเห็นเพียงหน้าจอของผู้ใช้สำหรับโครงการของพวกเขา แต่ถ้าผู้ใช้ที่มีสิทธิของผู้ดูแลพวกเขาจะเห็นหน้าจอของผู้ใช้เช่นเดียวกับหน้าจอผู้ดำเนินการ

หน้าจอของผู้ใช้มี

- Manage Compute
- Overview

- Instances
- Volumes
- Images & Snapshots
- Access & Security

- Manage Network
- Networks
- Routers
- Network Topology

- Object Store
- Containers

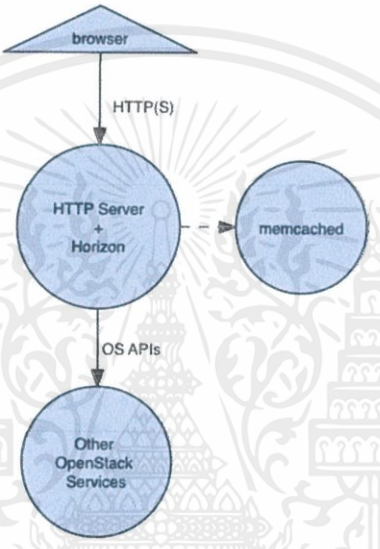
หน้าจอเหล่านี้ช่วยให้พวกเขาสามารถเข้าถึงและใช้บริการ ตัวอย่างเช่นหน้าจอคอนเทนเนอร์ภายใต้วัตถุที่เก็บช่วยให้ผู้ใช้สามารถสร้างและลบคอนเทนเนอร์ Swift เช่นเดียวกับวัตถุที่อัปโหลดลงในพวกมัน

หน้าจอดำเนินการมีหน้าจอที่ช่วยให้ผู้ใช้สามารถเปลี่ยนการทำงานโดยรวมของระบบคลาวด์ หน้าจอเหล่านี้ประกอบด้วย

- Overview
- Flavors
- Instances
- Volumes
- Images and Snapshots
- Projects
- Users
- System Info

แผงควบคุมเหล่านี้ช่วยให้ผู้ประกอบการเปลี่ยนความคมชัดของภาพหรือการกำหนดค่าคุณสมบัติ cloudwide ตัวอย่างเช่นหน้าจอ Flavors ช่วยให้ผู้ประกอบการสร้าง ลบ หรือเปลี่ยนแปลง Flavors รองรับกับผู้ใช้ระบบคลาวด์

สถาปัตยกรรมHorizonค่อนข้างง่าย มันเป็นโปรแกรมเว็บเขียนด้วยกรอบ Django ซึ่งอยู่บนพื้นฐานของ Python Horizonมักจะใช้งานผ่าน mod_wsgi ใน Apache โค้ดของตัวมันเองถูกแยกออกเป็นโมดูลPythonที่นำมาใช้ใหม่กับส่วนใหญ่ของลอจิก (การสื่อสารกับ APIs OpenStack ต่างๆ) และการนำเสนอ (เพื่อให้มันง่ายดายที่ปรับแต่งสำหรับเว็บไซต์ที่แตกต่างกัน)



รูปที่ 2.29 รูปแบบการทำงานของ Horizon

ในมุมมองของสถาปัตยกรรมเครือข่าย บริการนี้จะต้องมีลูกค้าที่สามารถเข้าถึงได้เช่นเดียวกับจะสามารถพูดคุยกับของ APIs สาธารณะแต่ละบริการ ถ้าคุณต้องการที่จะใช้ฟังก์ชันการทำงานของผู้ดูแลระบบ (i.e. สำหรับบริการอื่น ๆ) มันจะต้องการเชื่อมต่อกับอุปกรณ์ปลายทางกับผู้ดูแล API ของพวกเขา (ซึ่งไม่ควรจะสามารถเข้าถึงลูกค้า)

2.3.2.9 Ceilometer Concepts

Ceilometer (Telemetry Service) ทำหน้าที่วัดการใช้งาน resources ไม่ว่าจะเป็น CPU, Memory, Storage และ Network หรือสถิติการใช้บริการต่างๆเช่น ค่าบริการ Ceilometer มีกลุ่มของแนวคิดพื้นฐานคือ

Resources

ทรัพยากรแสดงให้เห็นถึงสิ่งที่จะถูกมิเตอร์หรือการตรวจสอบ พวกนี้มักสามารถทำโดย รูปแบบของการอินสแตนซ์ OpenStack หรือ volumes อย่างไรก็ตาม นอกจากนี้ยังมีทรัพยากรที่โครงสร้างเหมือนโครงการและเครือข่าย

Meters

มิเตอร์แบบที่เกิดขึ้นจริงของการวัด Ceilometer สนับสนุนหลายมิเตอร์จากทรัพยากรที่แตกต่างกัน นอกจากนี้ในแต่ละมิเตอร์เหล่านี้จะรายงานข้อมูลของพวกเขากลับมาแบบ one of several กับหน่วยของตัวเอง ตัวอย่างเช่น "ซีพียู" การวัดหน่วยวัดระยะเวลาสะสม CPU ที่ใช้ในหน่วยของ "ns" หรือนาโนวินาทีของเวลา มิเตอร์ Ceilometer สามารถใช้ตามชนิดดังต่อไปนี้

- แบบสะสมเป็นการเพิ่มมากขึ้นเมื่อเวลาผ่านไปเหมือนการใช้ซีพียู
- มาตรวัดแสดงค่าแน่นอนในเวลาที่ measurement เหมือนดิสก์ I / O หรือการส่งภาพ

Deltas แสดงความแตกต่างระหว่างการวัดในปัจจุบันและการวัดที่ผ่านมา

Ceilometer ดำเนินการของมิเตอร์จำนวนมาก มิเตอร์เหล่านี้จะถูกแบ่งออกเป็นหลายประเภทดังนี้

Meter	Examples
Compute	Disk I/O requests, vCPUs, Instance (also reported by flavor)
Network	Network creation, Floating IP duration
Image	Uploaded image size, Images,
Volume	Volumes, Volume size
Object Storage	Total size of stored objects, Number of API requests
Energy	Power consumption, Energy
Orchestration	Creation, deletion and update requests for stacks

ตารางที่ 2.8 Meter ใน Ceilometer

ขณะที่แต่ละมิเตอร์พึ่งพาเทคโนโลยีพื้นฐานในการค้นหาการเก็บรวบรวมข้อมูล ไม่มีมิเตอร์ที่รองรับสำหรับทุกผู้ให้บริการ OpenStack (จัดเก็บข้อมูลเครือข่ายเสมือนจริงและอื่น ๆ)

Samples

Samples เป็นการตรวจวัดที่ไม่ต่อเนื่องโดยเครื่องวัดบนทรัพยากร ตัวอย่างเช่น Samples อาจจะมีจำนวนของอินสแตนซ์ที่วัดในโครงการโดยเฉพาะใน OpenStack สำหรับห้านาทีแรกของวันที่เฉพาะเจาะจง บันทึกในฐานข้อมูล Ceilometer ที่สามารถเรียกดูหรือจัดกลุ่มกับตัวอย่างอื่น ๆ เพื่อสร้างรายงานการใช้งาน

Alarms

สัญญาณเตือนเป็นคุณลักษณะใหม่ใน Ceilometer สัญญาณเตือนเป็นรูปแบบพื้นฐานของการสนับสนุนการตรวจสอบ Ceilometer และการบูรณาการ OpenStack Heat พวกมันจะมีเกณฑ์ในการเครื่องวัดโดยเฉพาะและทรัพยากรที่จะสร้างเหตุการณ์เมื่อพวกเขาเกินกว่าที่กำหนด ตัวอย่างเช่นสัญญาณเตือนที่สามารถตั้งค่าในการใช้ CPU ตัวอย่างของเกณฑ์ที่มี 75% ในตัวอย่างที่ซับซ้อนมากขึ้นการแจ้งเตือนยังสามารถกำหนดให้มีการ evaulated กับช่วงเวลาที่กำหนดให้ใช้การเปรียบเทียบมากขึ้น (เช่นน้อยกว่าหรือเท่ากับ) หรือแม้กระทั่งขึ้นอยู่กับสถานะสัญญาณเตือนอื่น ๆ

Statistics

สถิติเป็น Samples ที่รวบรวมจัดกลุ่มตามมิเตอร์ในช่วงระยะเวลาที่ระบุ ตัวอย่างเช่นสถิติอาจจะเป็น vCPUs ที่ใช้โดยผู้เช่าที่ระบุวันที่ที่ระบุ

Pipelines

Pipelines เป็นคุณสมบัติขั้นสูงการเลือกที่ช่วยให้ผู้ประกอบการปรับแต่งการเก็บรวบรวมข้อมูล ceilometer มันช่วยผู้ประกอบการในการแปลงข้อมูลการวัดระหว่างเครื่องมือวัดและจัดเก็บข้อมูล ตัวอย่างเช่นPipelineอาจแปลงหน่วยของการวัดหรือลดช่วงของการวัดสำหรับเครื่องวัดที่ระบุ

Ceilometer Architecture

OpenStack Telemetry มีสถาปัตยกรรมแบบกระจายมากที่แผ่กระจายออกมาในรูปแบบ เซิร์ฟเวอร์ Ceilometer กลางให้กับแต่ละโหนดประมวลผลโนวา

แตกต่างจากโปรแกรมอื่น ๆ OpenStack, Ceilometer ไม่ได้มีเฉพาะหน้าจอแดชบอร์ด Horizon

Ceilometer API

กระบวนการ API Ceilometer ยอมรับการร้องขอ API และส่งกลับข้อมูลจากแหล่งข้อมูล มันมีลอจิกหลักสำหรับ Ceilometer อย่างไรก็ตามกระบวนการ API ไม่ยอมรับหรือดำเนินการใด ๆ ของข้อมูลการวัดที่แท้จริง ข้อมูลที่มีถูกป้อนเข้าแหล่งข้อมูลผ่านทางตัวแทนการรวบรวม

Collection Agents

Ceilometer ขึ้นอยู่กับตัวแทนในการเก็บรวบรวมข้อมูลจากโหนด OpenStack ตัวแทนเหล่านี้ทำงานบนโหนดประมวลผลเพื่อรวบรวมข้อมูล VM จากไฮเปอร์ไวเซอร์พื้นฐาน นอกจากนี้ นอกจากนี้ Ceilometer ยังรับฟังคิวข้อความที่ใช้โดยแต่ละบริการในการตรวจสอบเหตุการณ์ เหมือนกับ อินสแตนซ์ใหม่ spawned, สร้างblock storage volumes หรือ floating IP addresses ที่เกี่ยวข้อง

Ceilometer ของตัวแทนมีการจัดเก็บ:

- ceilometer-agent-central
- ceilometer-agent-compute
- ceilometer-collector

Alarm Daemons

ขณะที่ Ceilometer ใช้ชุดประมวลผลกับข้อมูลการวัดที่รวบรวม มันยังใช้ชุดของกระบวนการในการอำนวยความสะดวกในการแจ้งเตือน มีสองส่วนที่จะเตือนใน Ceilometer คือ ประเมินและผู้แจ้ง หากการเตือนภัยได้รับการสะดุดผู้แจ้ง (*ceilometer-alarm-notifier*) ถูกเรียกจะดำเนินการการกระทำที่เหมาะสม

Data Store

Ceilometer เก็บข้อมูลการวัดที่รวบรวม เหตุการณ์ และการกำหนดค่าของตัวเองใน แหล่งข้อมูล มันสนับสนุนกลุ่มของ SQL และ NoSQL แบ็กเอนด์สำหรับการจัดเก็บข้อมูล ปัจจุบัน ฐานข้อมูลสนับสนุนมี MongoDB, HBase, MySQL (หรือ sqlalchemy-enabled databases) และ DB2

Data Store	Notes
MongoDB	The default data store for Ceilometer and the most fully tested option
MySQL	Lacks coplete support for alarming features
HBase	
DB2	

ตารางที่ 2.9 Data store ของ Ceilometer

โดยใช้ค่าเริ่มต้น MongoDB data store. Ceilometer สร้าง "Collections" ดังต่อไปนี้ (ตามโครงสร้างตารางที่เหมือนจะถูกเรียกว่าใน MongoDB)

Collection/Table	Notes
alarm	List of all alarms
alarm_history	List of historical alarms
meter	List of meters
project	List of projects/tenants
resource	List of resources

Collection/Table	Notes
system.indexes	
user	List of users

ตารางที่ 2.10 Collection ของ Ceilometer

Queue

ในขณะที่ Ceilometerเองไม่ได้จำเป็นต้องใช้ของคิวการส่งข้อความของตัวเองในการดำเนินงานมันไม่ฟังบนคิวข้อความของการให้บริการ OpenStack อื่น ๆ ที่จะตรวจพบและบันทึกเหตุการณ์ ประเมิน (*ceilometer-alarm-evaluator*) ตรวจสอบรายการสัญญาณเตือนที่กำหนดไว้เพื่อดูว่าใด ๆ สัญญาณเตือนที่ได้รับการสะดุด

2.4 Storage

2.4.1 Object Storage

รูปที่ 2.30 รูปแบบของ Object Storage

Object Storage ไม่ใช่การจัดเก็บไฟล์ในรูปแบบดั้งเดิม แต่เป็นระบบการจัดเก็บข้อมูลที่ใช้การกระจายตัวของ Storage System เพื่อทำการจัดเก็บข้อมูลประเภท Static Data อย่างเช่น การจัดเก็บไฟล์รูปภาพ (Photo Storage), การจัดเก็บอีเมล, การสำรองข้อมูล และการจัดเก็บแฟ้มข้อมูล โดยจะไม่มีการรวมศูนย์การประมวลผลอยู่ที่จุดศูนย์กลางเพียงจุดเดียว หากแต่ใช้การกระจายตัวในรูปแบบของ Distributed Storage System แทน

- ตัวไฟล์และข้อมูลต่างๆ จะถูกเขียนลงใน Multiple Disk Drives และทำการประมวลผลอย่างแพร่กระจายอย่างทั่วถึงในศูนย์ข้อมูล (Data Center) โดยซอฟแวร์ของระบบนั้นจะทำให้มั่นใจได้ว่าตัวข้อมูลนั้นจะถูกบันทึกอย่างแพร่กระจายอย่างทั่วถึงในทุกๆ Clusters
- สำหรับในเชิงเทคนิคแล้ว Storage Clusters นั้นสามารถทำการขยายตัวได้โดยการเพิ่มเซิร์ฟเวอร์ใหม่เข้าไป หากเซิร์ฟเวอร์หรือ ฮาร์ดไดร์ฟตัวใดตัวหนึ่งเกิดการขัดข้องขึ้น ระบบจะทำการจัดเก็บเนื้อหาและข้อมูลต่างๆ โดยใช้ Active Nodes ตัวอื่น ทำการจัดการเก็บข้อมูลไปยัง New Locations ภายใต้ใน Clusters นั้น

2.4.2 Block Storage

รูปที่ 2.31 รูปแบบของ Block Storage

สำหรับระบบ Block Storage System นั้นจะทำการบริหารในส่วนของ Creation, Attaching และ Detaching ของตัว Block Devices ไปยังเซิร์ฟเวอร์ต่างๆ โดยตัว Block Storage Volumes นั้นจะทำการรวมเข้ากับ Compute และในส่วนของ Dashboard เพื่อทำให้ผู้ใช้งานระบบสามารถใช้งานและบริหารจัดการ Storage ได้ตามที่ตนเองต้องการ

นอกเหนือจากการใช้ Storage Server พื้นฐานอย่าง Linux แล้ว ระบบยังทำการสนับสนุนการจัดเก็บข้อมูลอย่างครบวงจรโดยรวมถึง Ceph, NetApp, Nexenta และ SolidFire

Block Storage นั้นมีความเหมาะสมสำหรับ สถานการณ์ที่ต้องใช้ความแม่นยำของการจัดการระบบในระดับสูง อาทิเช่น การจัดเก็บคลังข้อมูล, การขยาย File System หรือแม้แต่การเพิ่มเซิร์ฟเวอร์ที่ใช้เชื่อมต่อกับตัว Row Block Level Storage

Snapshot Management ช่วยอำนวยความสะดวกในการบริหารจัดการในส่วนของการ Backing Up Data

2.4.3Block Storage VS Object Storage

	Block storage	Object storage
Used to...	เพิ่มการจัดเก็บข้อมูลถาวรเพิ่มเติมในเครื่องเสมือน (VM)	จัดเก็บข้อมูล รวมถึง VM images
Accessed through...	block device สามารถแบ่งพาร์ติชัน , ฟอร์แมต และติดตั้ง (เช่น /dev/vdc)	The REST API
Accessible from...	ภายใน VM	Anywhere
Managed by...	OpenStack Block Storage (cinder)	OpenStack Object Storage (swift)
Persists until...	Deleted by user	Deleted by user
Sizing determined by...	การกำหนดของผู้ใช้ในการร้องขอเริ่มต้น	จำนวนของการจัดเก็บทางกายภาพที่มีอยู่
Example of typical usage...	1 TB disk	10s of TBs of dataset storage

ตารางที่ 2.11 ความแตกต่างระหว่าง Block storage กับ Object storage

2.5 ระบบแฟ้ม (File System)

File System หมายถึง สิ่งที่ผู้ใช้เกี่ยวข้องโดยตรง แต่มักไม่รู้ตัวเนื่องจากเป็นการอำนวยความสะดวกโดยระบบปฏิบัติการอย่างอัตโนมัติ ระบบแฟ้มเป็นฐานที่ทำให้เกิดการจัดการโปรแกรมและข้อมูลในทุกการดำเนินงานของระบบซอฟท์แวร์ที่เข้าควบคุมสื่อเก็บข้อมูล

ระบบแฟ้มประกอบด้วย 3 ส่วน คือ

1) รวมรวมแฟ้ม (Collection of Files) เก็บข้อมูลที่สัมพันธ์ให้ถูกอ้างอิงได้ในรูปแฟ้มข้อมูล
2) โครงสร้างแฟ้ม (Directory Structure) จัดการอำนวยการเข้าถึงแฟ้มและจัดกลุ่มอย่างเป็นระบบ
3) พาทิชัน (Partitions) ซึ่งแยกเป็นทางกายภาพ (Physically)หรือทางตรรกะ (Logically)ของระบบไดเรทรอรี่

วิธีการจัดเก็บข้อมูลที่ใช้กันใน OS ทุกตัวคือ จัดเก็บข้อมูลเป็นแฟ้มข้อมูลหรือไฟล์ (file) ไฟล์คือสิ่งที่บรรจุข้อมูล,โปรแกรมหรืออะไรก็ได้ที่ผู้ใช้ต้องการรวบรวมไว้เป็นชุดเดียวกัน การอ้างถึงไฟล์หรือข้อมูลต่างๆ ภายในไฟล์ของโปรแกรม จะไม่มีความเกี่ยวข้องกับแอดเดรสของโปรแกรมใดๆ ทั้งสิ้น OS มีโอเปอร์เรชั่นพิเศษที่เรียกว่า system call ไว้ให้โปรแกรมเรียกใช้ เพื่อให้สามารถจัดการงานที่เกี่ยวกับไฟล์ได้

2.5.1 EXT4

Ext4 เป็นรุ่นที่สี่ของตระกูล Ext filesystem และเป็น filesystem เริ่มต้นให้กับRed Hat Enterprise Linux 6 Ext4 สามารถอ่านและเขียนไปยัง Ext2 หรือ Ext3 filesystemได้ Ext4ได้เพิ่มและพัฒนาคุณสมบัติใหม่ที่ร่วมกันเป็น filesystem ที่มีความทันสมัยเช่น

- extent-based metadata
- delayed allocation
- journal check-summing
- large storage support

เป็นวิธีที่เล็กและมีประสิทธิภาพมากขึ้นที่ใช้ติดตามพื้นที่ใน filesystem ที่มีการใช้งาน คุณสมบัติ extent-based metadataและ delayed allocation คุณสมบัติเหล่านี้ปรับปรุงประสิทธิภาพ filesystem และลดพื้นที่การใช้ของข้อมูล Delayed allocation ช่วยทำให้ filesystem เลื่อนการเลือกพื้นที่ที่เก็บข้อมูลที่เขียนใหม่จนกว่าข้อมูลจะถูกเขียนลงไปในดิสก์ ซึ่งช่วยให้มีประสิทธิภาพการทำงานสูงขึ้นเนื่องจากช่วยให้มีไฟล์ขนาดใหญ่ขึ้น,มีการจัดสรรพื้นที่ที่อยู่ติดกันมาก การเปิดใช้งาน filesystem ช่วยในการตัดสินใจด้วยข้อมูลที่ดีกว่ามาก

นอกจากนี้เวลาซ่อมแซมระบบแฟ้ม (fsck) ใน Ext4 จะเร็วกว่าในExt3 และExt2 การซ่อมแซ่มบาง filesystem ได้แสดงให้เห็นถึงการเพิ่มประสิทธิภาพเป็น6เท่า

ปัจจุบันขนาดที่มากที่สุดที่สนับสนุน Redhat คือ 16 TB ทั้ง Red Hat Enterprise Linux 5 และ Red Hat Enterprise Linux 6 ประสิทธิภาพการทำงานของการใช้งานขึ้นอยู่กับหลายตัวแปร; นอกเหนือไปจาก filesystem ที่ได้เลือก ยังขึ้นอยู่กับI/Oที่มีรูปแบบเฉพาะที่ประยุกต์การเพิ่มและชนิดของเซิฟเวอร์และstorage hardware ที่ใช้

2.5.2 XFS

XFS เป็น filesystem 64-bit journaling ที่สมบูรณ์และมีประสิทธิภาพที่สนับสนุนไฟล์ขนาดใหญ่มากและหลาย filesystem บนโฮสต์เดียว journalingทำให้ filesystem มีความสมบูรณ์หลังจากระบบล่มเช่น ช่วงไฟฟ้าขัดข้องโดยเก็บบันทึกการดำเนินงานของ filesystem ที่สามารถย้อนได้ เมื่อระบบเริ่มใหม่และ filesystem remount XFS ถูกพัฒนามาในช่วงปี 1990 โดย SGIและมีประวัติอันยาวนานของการทำงานบนเซิร์ฟเวอร์ที่มีขนาดใหญ่มากและ storage arrays XFS สนับสนุนคุณสมบัติมากมายแต่ไม่จำกัด:

- delayed allocation
- dynamically allocated inodes
- b-tree indexing เพื่อขยายขีดความสามารถในการบริหารจัดการพื้นที่ว่าง
- ความสามารถในการรองรับจำนวนมากของการดำเนินงานพร้อมกัน
- ครอบคลุมการเช็คความสอดคล้องของ run-time metadata
- sophisticated metadata read-ahead algorithms
- มีการสำรองข้อมูลอย่างเข้มงวดและrestore
- online defragmentation
- online filesystem growing
- ความสามารถในการวินิจฉัยที่ครอบคลุม
- สามารถปรับขนาดได้และซ่อมแซมได้เร็ว
- การเพิ่มประสิทธิภาพสำหรับปริมาณงานของ streaming video

ในขณะ XFS ปรับขนาดเป็น exabytes imageของ XFS filesystem ที่สนับสนุนสูงสุดของ Red Hatคือ100TB ในระยะเวลาอันยาวนานในสภาพแวดล้อมที่ต้องการประสิทธิภาพสูงและขยายขีดความสามารถก็ไม่น่าแปลกใจที่ XFS เป็นหนึ่งใน filesystems ประสิทธิภาพสูงสุดในระบบขนาดใหญ่ที่มีปริมาณงานขององค์กร ยกตัวอย่างเช่นระบบขนาดใหญ่จะเป็นหนึ่งในที่มีจำนวนที่ค่อนข้างมากของซีพียู, หลาย HBAs และเชื่อมต่อกับ external disk arrays XFS ยังทำงานได้ดีในระบบขนาดเล็กที่มี multi-threaded ปริมาณงาน parallel I/O XFS มีประสิทธิภาพที่ค่อนข้างแย่สำหรับ single threaded ปริมาณข้อมูลจำนวนมากเช่นปริมาณงานที่สร้างหรือ ลบจำนวนมากของไฟล์เล็ก ๆ ใน single thread

2.5.3 Ext4 VS XFS

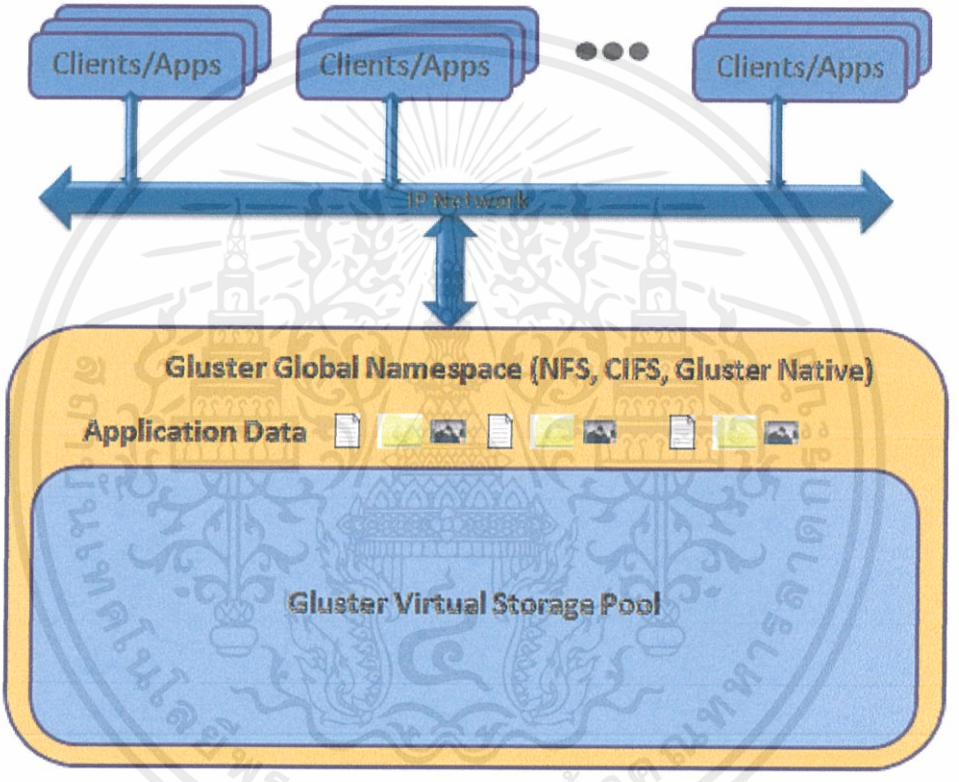
	Ext4	XFS
Online resize	Grow only	Grow only
Offline resize	Grow+Shrink	No
Online checks	No	No
Snapshots	No	No
Clones	No	No
Internal RAID	No	No
Compression	No	No
Dedupe/Encryption	No	No
Online Defrag	Yes	Yes
Discard (TRIM)	Yes	Yes
FLUSH/FUA(Barrier)	Yes	Yes
Metadata CRC	Yes	Yes
Data CRC	No	No
Extent allocation	Yes	Yes
Delayed allocation	Yes	Yes
Production-ready	Yes	Yes

	Max File Size	Max Filesystem Size
Ext4	1 EiB	1 EiB (tool limits < !)
XFS	8 EiB	16 EiB

รูปที่ 2.32 รูปตารางการเปรียบเทียบของ EXT4 กับ XFS

2.6 GlusterFS

GlusterFS เป็น opensource ที่ distribute filesystem สามารถในการปรับขนาดพื้นที่ได้ และจัดการ clients เป็นพัน กลุ่ม GlusterFS ร่วมกับ storage สร้าง Block ผ่าน Infiniband RDMA หรือการเชื่อมต่อ TCP / IP, มีการรวบรวมดิสก์และทรัพยากรหน่วยความจำ แล้วจัดการข้อมูลที่อยู่ใน Global namespace เดียวกัน GlusterFS จะขึ้นอยู่กับการออกแบบพื้นที่การใช้งานที่วางซ้อนกันได้ และทำให้มีประสิทธิภาพที่ยอดเยี่ยมสำหรับปริมาณงานที่หลากหลาย

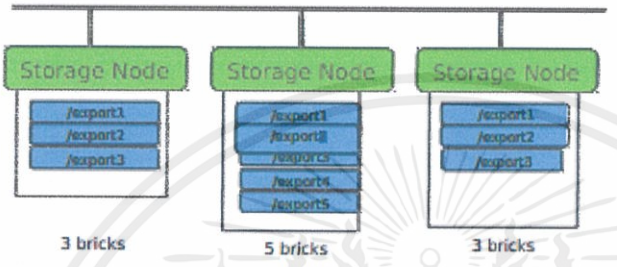


รูปที่ 2.33 รูปแบบการทำงานของ GlusterFS

GlusterFS สนับสนุน clients ที่ใช้ application ผ่าน IP network มาตรฐานใดๆจากรูปข้างต้นแสดงให้เห็นว่าผู้ใช้สามารถเข้าถึงข้อมูล application และไฟล์ใน global namespace โดยใช้ความหลากหลายของโปรโตคอลมาตรฐาน GlusterFS ให้ผู้ใช้สามารถปรับขยายขนาด virtualized storage ที่ปรับจาก terabytes เป็น petabytes ในศูนย์กลางการจัดการและ commoditized pool ของstorage

2.6.1 Brick

- Brick คือการรวมกันของโหนดและส่งออกเป็นไดเรกเทอรีเช่น hostname:/dir
- โดยแต่ละ brick จะสืบทอดข้อจำกัดของ filesystem พื้นฐาน
- ไม่มีการจำกัดจำนวน brick ต่อโหนด
- แต่ละ brick ใน cluster ควรมีขนาดเดียวกัน



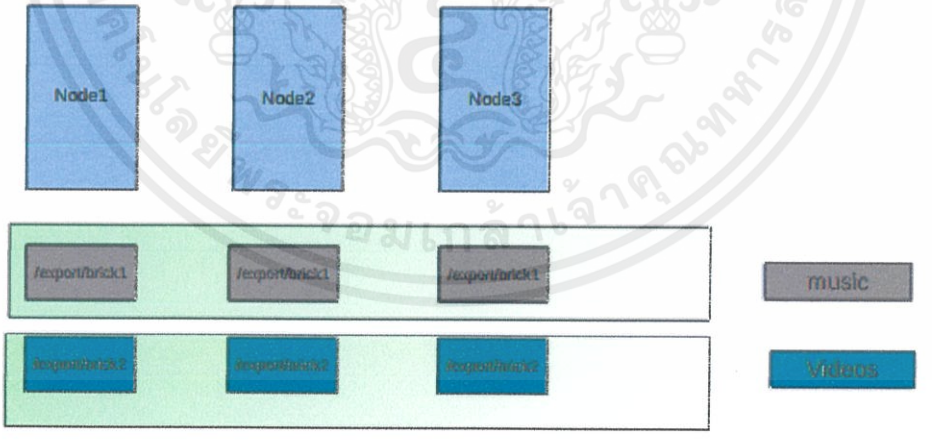
รูปที่ 2.34 รูปแบบโครงสร้างของ brick

2.6.2 Volume

Volume คือชุด logical ของ brick

Volume ถูกระบุโดย admin ตั้งชื่อ

Brick ที่มาจากโหนดเดียวกันสามารถจะเป็นส่วนหนึ่งของ volume ที่แตกต่างกัน



รูปที่ 2.35 รูปแบบโครงสร้างของ Volume

2.6.3 Volume types

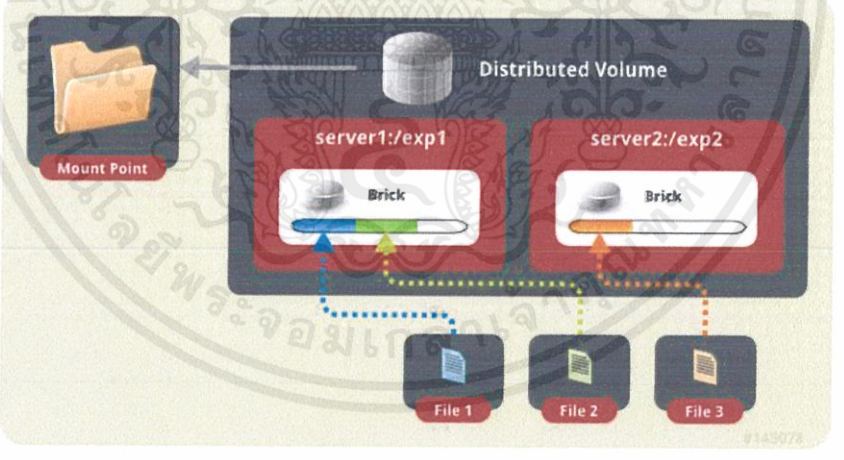
ประเภทของ volume จะถูกระบุไว้ในช่วงตอนสร้าง volume ประเภทของ volume จะเป็นตัวกำหนดวิธีการที่ข้อมูลจะถูกวาง

ประเภทของvolumeที่ GlusterFS สนับสนุน

- Distribute
- Stripe
- Replication
- Distributed Replicate
- Striped Replicate

2.6.3.1 Distributed Volume

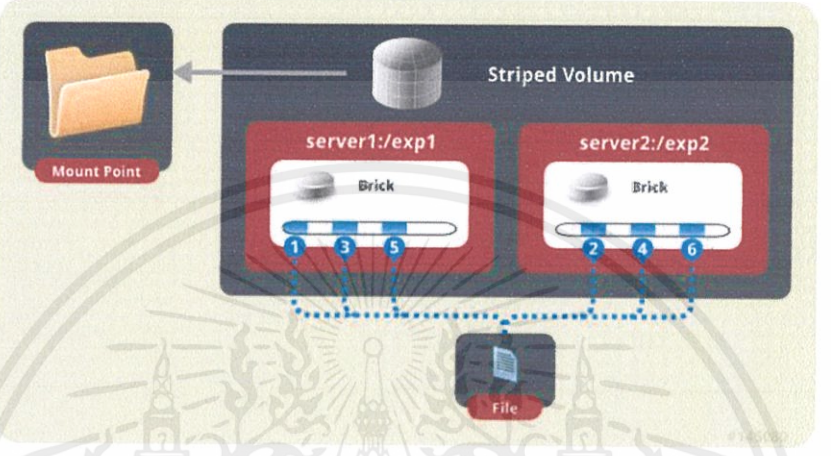
Distributed Volume ของ GlusterFS นั้นคือการแจกจ่ายไฟล์ไปยังแต่ละ Brick (Server) นั่นเอง โดยจุดประสงค์ของการทำ Distributed Volume นั้นคือการทำ Horizontal Scaling และ Distibuted ข้อมูลออก ทำให้ได้พื้นที่ของการเก็บข้อมูลที่เพิ่มมากขึ้น เพียงแค่เพิ่ม Brick (Server)



รูปที่ 2.36 รูปแบบการทำงานของ Distributed Volume

2.6.3.2 Striped Volume

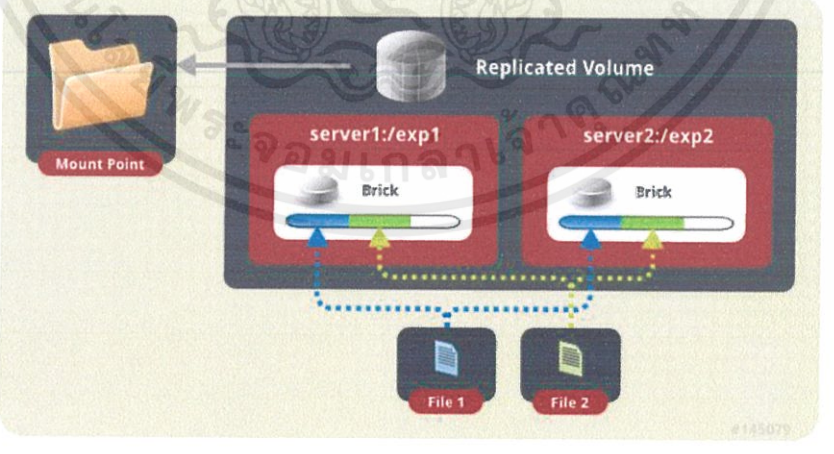
Striped Volume นั้นจะช่วยให้เราสามารถ Access File ได้เร็วขึ้นโดยเมื่อมีการสร้างไฟล์บน Gluster แล้วไฟล์จะถูก Strip ออกเป็นหลายๆ ส่วนแล้วโยนเข้าไปในแต่ละ Brick ดังนั้นการ Access File จะมีประสิทธิภาพที่ดีขึ้นอย่างเห็นได้ชัดโดยเฉพาะไฟล์ใหญ่ๆ



รูปที่ 2.37 รูปแบบการทำงานของ Striped Volume

2.6.3.3 Replicated Volume

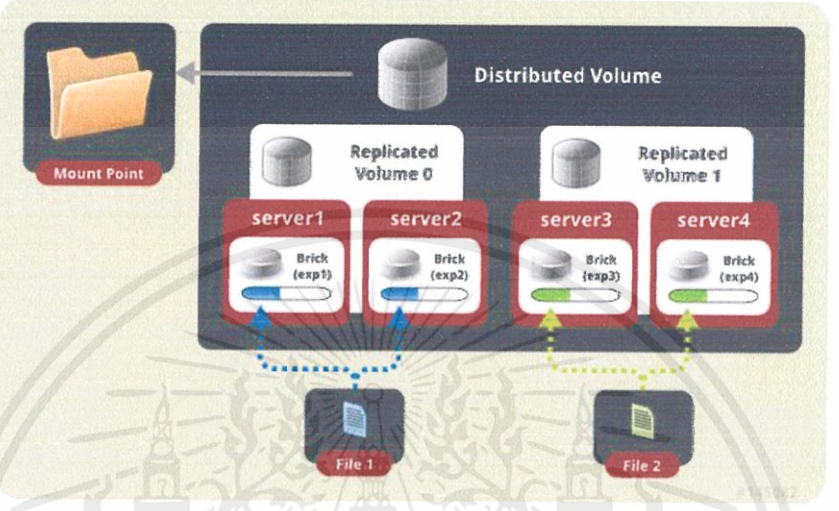
Replicated Volume ของ GlusterFS นั้น Concept ก็เหมือนกับการ Replicate ทั่วๆ ไป คือเมื่อมี Content วางอยู่ใน node ใด node หนึ่งให้ทำการ copy ไปเก็บไว้อีก node หนึ่งอัตโนมัติ



รูปที่ 2.38 รูปแบบการทำงานของ Replicated Volume

2.6.3.4 Distributed Replicated Volume

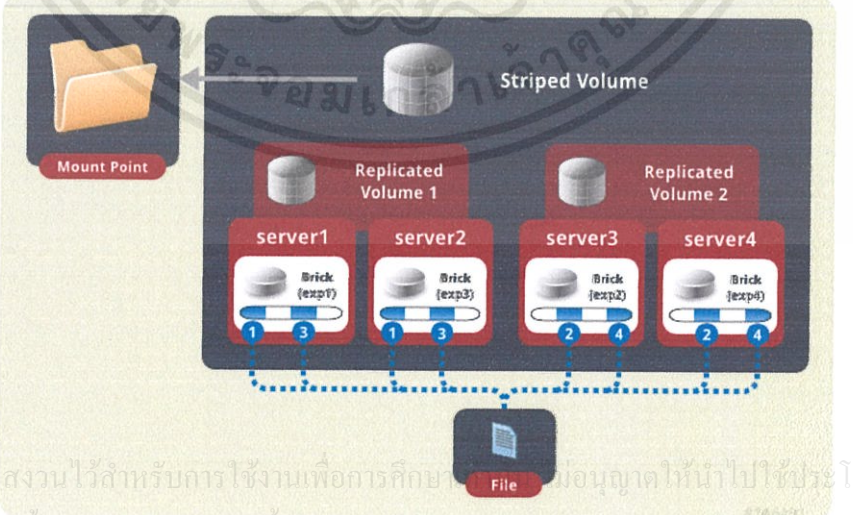
การทำ Distributed Replicated Volume นั้นเหมาะสำหรับการ Scale Storage ที่ง่าย และมีการ Replicate ไว้เผื่อในกรณีที่ไฟล์มีปัญหาเราจะได้มีอีก 1 หรือหลาย copy เก็บไว้ด้วย



รูปที่ 2.39 รูปแบบการทำงานของ Distributed Replicated Volume

2.6.3.5 Striped Replicated Volume

การทำ Striped Replicated Volume นั้นจะคล้ายๆ กับการทำ Striped Volume ซึ่งมีจุดประสงค์คือช่วยให้สามารถ Access ไฟล์ได้อย่างรวดเร็ว ซึ่ง Striped Replicated นั้นพ่วงการทำ Replicated ไฟล์เข้ามาด้วยนั่นเอง



รูปที่ 2.40 รูปแบบการทำงานของ Striped Replicated Volume

2.7 Software-Defined Networking (SDN)

2.7.1 หลักการของ SDN

SDN คือ สถาปัตยกรรมของเครือข่าย ที่ดึงการจัดการเครือข่ายทั้งหมดมารวมอยู่ที่ซอฟต์แวร์ โดยไม่สนใจอุปกรณ์และความแตกต่างของผู้ผลิต ในแง่นี้ SDN จะแก้ไขปัญหาตรงที่ผู้ดูแลเครือข่าย สามารถจัดการตั้งค่าอุปกรณ์ในจุดเดียว แล้วสามารถนำไปใช้ได้เลย เมื่อมีอุปกรณ์ใหม่ๆ เพิ่มเข้ามาในระบบ ก็ไม่จำเป็นที่จะต้องแก้ไขที่ตัวเครื่อง นอกจากแก้ไขซอฟต์แวร์เล็กๆ น้อยๆ เท่านั้น

วิธีคิดของ SDN ที่เข้ามาแก้ไขปัญหาของผู้ดูแลเครือข่าย กำลังเป็นแนวทางใหม่ในอุตสาหกรรมคอมพิวเตอร์ และเป็นแนวคิดที่บริษัทขนาดใหญ่หรือผู้ให้บริการโครงข่าย ให้ความสนใจเป็นอย่างมาก เนื่องจากสิ่งที่เป็นผลลัพธ์ของโครงสร้างนี้คือความยืดหยุ่นและเอื้อต่อการขยายตัวของโครงข่าย เมื่อจำเป็นที่จะต้องเพิ่มความสามารถในการรองรับการใช้งานเข้าไป

2.7.2 ประวัติและพัฒนาการของ SDN

SDN เป็นผลพวงจากความพยายามในการพัฒนาแนวคิดด้านเครือข่ายตั้งแต่ในช่วงทศวรรษที่ 1990 ซึ่งในช่วงเวลานั้น แนวคิดเกี่ยวกับ Active Networking เกิดขึ้นมา โดยตอบโจทย์ที่ว่าเครือข่ายจะสามารถทำให้ “กำหนด” หรือ “สามารถโปรแกรม” (programmable) ได้อย่างไร แนวคิดของ Active Networking คือการมองว่าเครือข่ายนั้นสามารถที่จะเหมือนเครื่องคอมพิวเตอร์ตามปกติได้ ซึ่งก็คือการที่ผู้ดูแลระบบ/นักพัฒนา สามารถเขียนโปรแกรมเพื่อเข้าเรียกใช้งานทรัพยากร (resources) ของเครือข่ายให้ได้เต็มที่ วิธีการสำคัญคือการทำให้เครือข่ายมี API (Application Programming Interface) ในฐานะช่องทางการเข้าถึง และการถอดรหัส (demultiplex) หัวแพ็กเก็ต เพื่อส่งไปให้ถูกที่ ซึ่งที่สุดทั้งสองอย่างนี้อยู่ภายใต้แนวคิดที่ว่าโครงสร้างทั้งหมดของอุปกรณ์ต่างๆ จะต้องถูกรวมกัน (unified)

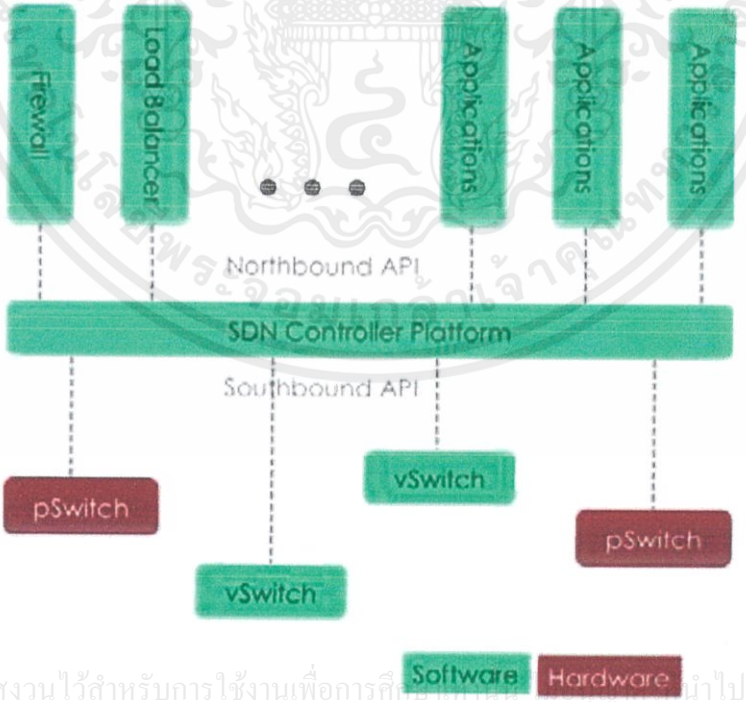
จากนั้นจึงเริ่มมีแนวคิดในการแบ่งชั้นของเครือข่าย โดยแบ่งออกเป็น control plane ซึ่งก็คือชั้นที่ควบคุมเส้นทางของข้อมูล และ data plane ซึ่งเป็นชั้นของข้อมูล (จะกล่าวเพิ่มเติมในภายหลัง) โดยอยู่ในช่วงระหว่างปี 2000 ที่เริ่มมีแนวคิดดังกล่าว โดยมองว่า การมองลักษณะเครือข่ายแยกกันดังนี้จะส่งผลดีเพื่อให้มีความคล่องตัวในการบริหารและจัดการมากขึ้น

ภายใต้กรอบคิดเช่นนี้ ชั้นทั้งสองที่แยกกันจะมีส่วนที่ประสานงานกันซึ่งเป็นส่วนที่เปิด และรวบรวมการควบคุมระบบเครือข่ายให้มาอยู่ในจุดเดียวกันทั้งหมด ซึ่งกลายมาเป็นแนวทางที่สำคัญสำหรับ SDN ซึ่งก็คือแนวคิดของการมี API เปิดซึ่งประสานงานเข้ากับ control plane และ data plane ในการจัดการ และการจัดการแบบกึ่งรวมศูนย์กึ่งกระจายตัว กล่าวคือ ในแง่หนึ่งเรารวมศูนย์

เพื่อจัดการ แต่ในเวลาเดียวกันการรวมศูนย์นี้จะต้องป้องกันไม่ให้เกิดความล้มเหลวแบบจุดเดียว (single point failure) อันจะนำมาซึ่งการล่มของระบบนั่นเอง

2.7.3 รูปแบบของ SDN

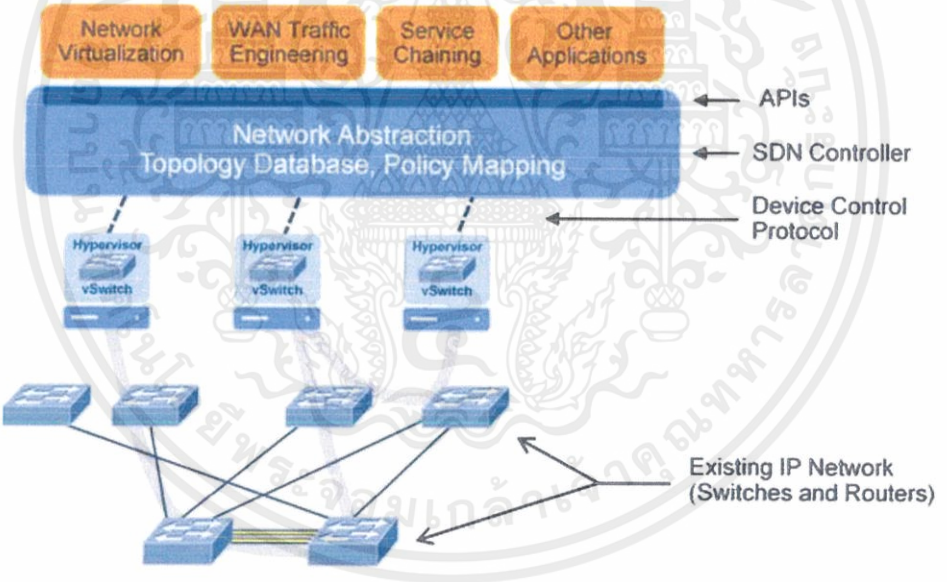
ในการที่จะทำให้การจัดการง่ายนั้น Network จะต้องถูกควบคุมจากศูนย์กลาง ไม่ว่าจะเป็น Policy หรือการออกแบบเส้นทางของ Network จะถูกประมวลผลจากส่วนกลาง ซึ่งจะถูกเรียกว่า SDN Controller ส่วนอุปกรณ์ Network ไม่ว่าจะเป็น Network Switch, Router ทั้งที่เป็น Physical และ Virtual เช่น Virtual Switch ใน Hypervisor จะถูก SDN Controller ควบคุมทั้งหมด (หรือบางส่วน ขึ้นอยู่กับรูปแบบ SDN) อุปกรณ์ Network เหล่านั้นจะทำหน้าที่แค่ส่งข้อมูลตามที่ SDN Controller ได้ Program ไว้ ตัวอย่างมาตรฐานที่ใช้ Program อุปกรณ์ Network ของ SDN Controller คือ OpenFlow, OVSDB ซึ่งเราจะรวมเรียกว่า Southbound API หรือ Device Control Protocol โดย Software เช่น Orchestration จะสั่งงาน Network ผ่าน Northbound API เช่น REST API



รูปที่ 2.41 รูปแบบของ SDN

Gartner ได้แบ่งรูปแบบ SDN ออกเป็น 3 แบบ คือ

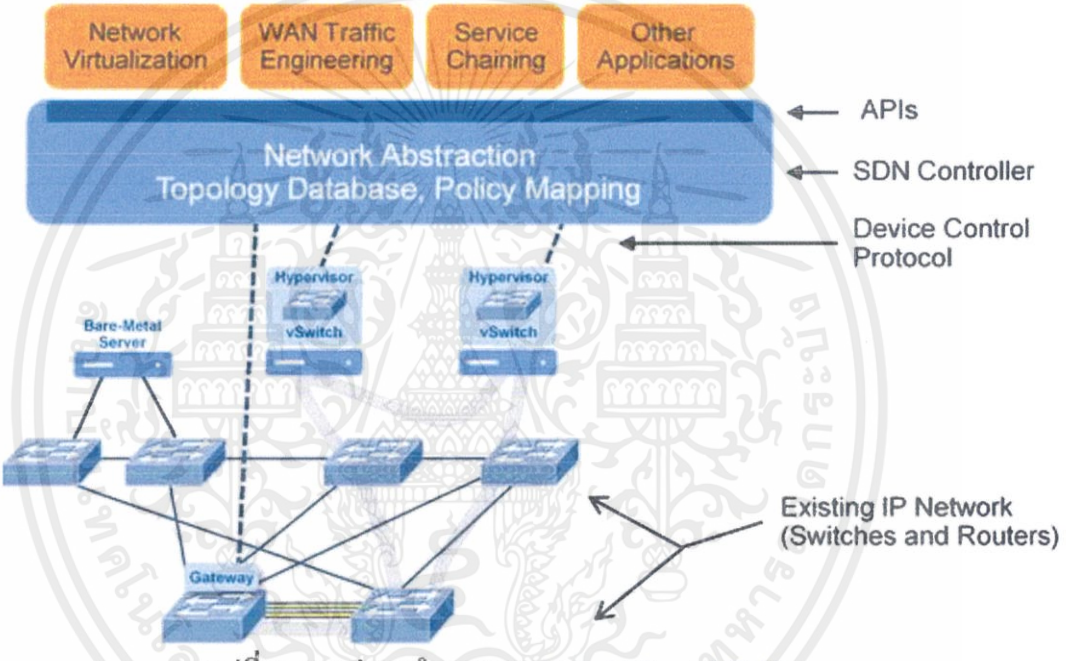
1) **Overlay-Based SDN** รูปแบบนี้เห็นได้ชัดใน Data Center คือ SDN Controller จะควบคุมอุปกรณ์ Network เฉพาะต้นทาง และปลายทาง ไม่ควบคุมอุปกรณ์ Network ระหว่างทาง (Underlay) ส่วนใหญ่วิธีการนี้ SDN Controller จะ Program หรือควบคุม Virtual Switch ที่อยู่ใน Hypervisor เพื่อทำให้ Virtual Machine (VM) ต้นทางและปลายทางอยู่ใน Logical Network เดียวกัน (เรียกว่า Network Virtualization) ทำให้ Software Orchestration สามารถจัดการ Virtual Machine สองตัวนี้เสมือนอยู่ใน Virtual Switch เดียวกัน ถึงแม้ว่าในความเป็นจริง จะมีอุปกรณ์ Existing Network ที่ซับซ้อนอยู่ด้านล่าง (Underlay) และ Server Host สองตัวนี้ อยู่คนละ Data Center กันก็ตาม



รูปที่ 2.42 รูปการทำงานของ Overlay-Based SDN

Protocol ของการทำ Overlay SDN ที่นิยมในปัจจุบันคือ VxLAN ซึ่ง VMWare เป็นผู้ผลักดัน โดยนำเสนอ SDN Controller ของตัวเองด้วย ซึ่งก็คือ NSX นั่นเอง

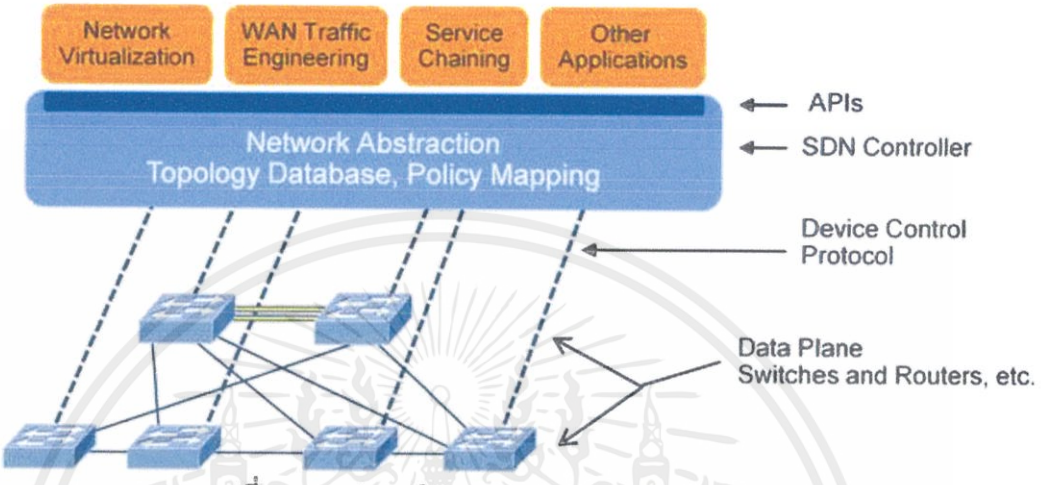
2) Hybrid-Based SDN คล้ายกับแบบแรก เพียงแต่ข้อจำกัดของแบบแรกคือ ไม่สามารถเชื่อมต่อกับระบบ Network เดิมที่ไม่รองรับ SDN หรือ ถ้าต้องการให้ Physical Server (Bare Metal Server) กับ Virtual Machine เชื่อมต่อกัน เหมือนอยู่ใน Local Network เดียวกัน Hybrid SDN ต้องการอุปกรณ์ Gateway ที่ รองรับการ Program ได้ด้วย SDN Controller ผ่าน SDN เช่น OpenFlow, OVSDB และยังต้องเข้าใจ Protocol ที่ ใช้ทำ Overlay SDN เช่น VxLAN ด้วย



รูปที่ 2.43 รูปการทำงานของ Hybrid-Based SDN

3) Device-Base SDN โดย SDN รูปแบบนี้ ตัว SDN Controller จะเข้าไป program flow หรือควบคุมอุปกรณ์ network ทั้งหมด เพื่อให้ได้รูปแบบ end-to-end ซึ่งจะสามารถตอบสนอง application ได้หลากหลายมากกว่า 2 แบบแรก แต่มีค่าใช้จ่ายในการปรับเปลี่ยนอุปกรณ์ network เพื่อให้รองรับสำหรับ SDN แบบนี้จะมีทั้งที่เป็น proprietary ของผู้ผลิตอุปกรณ์ network และทั้งที่รองรับ SDN controller แบบ open ตัวอย่าง SDN controller คือ Open Daylight ซึ่งใช้ OpenFlow เป็นตัว program อุปกรณ์ network

นอกจากนี้ Open Daylight Controller (Helium) ยังถูกใช้ใน OpenStack framework เพื่อเป็น Network Infrastructure ให้กับ Cloud และทำงานได้ทั้ง Overlay และ Hybrid-Based SDN



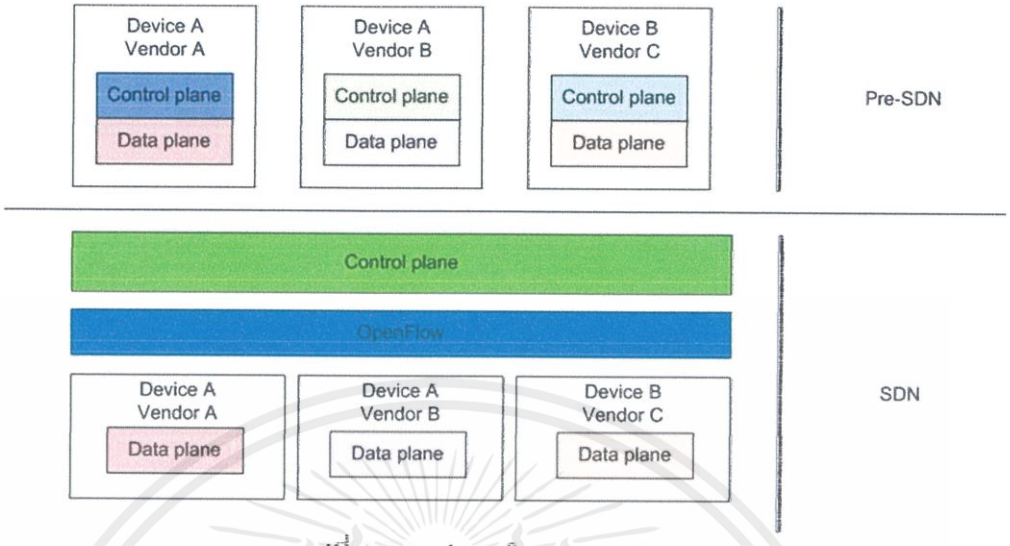
รูปที่ 2.44 รูปการทำงานของ Device-Based SDN

2.7.4 โครงสร้างของ SDN

SDN นั้นสร้างตัวเองอยู่บนฐานของแนวคิดที่ว่า ส่วนที่ควบคุม (control plane) และส่วนของข้อมูล (data plane) แยกออกจากกันอย่างชัดเจน โดยมีส่วนของการจัดการ (management) เป็นตัวเชื่อมโดยใช้ API ในการเชื่อมต่อทั้งสองส่วนนี้เข้าด้วยกัน

Control plane เป็นส่วนที่จะทำหน้าที่ในการตัดสินใจว่า แพ็กเก็ตที่วิ่งอยู่ภายในระบบหรือเข้าถึงระบบแล้ว จะต้องจัดการส่งต่อหรือทำอย่างไรก็ต่อไป ส่วน Data plane คือส่วนที่จะอนุญาตหรือทำหน้าที่ในการส่งข้อมูลไปตามการตัดสินใจของ control plane

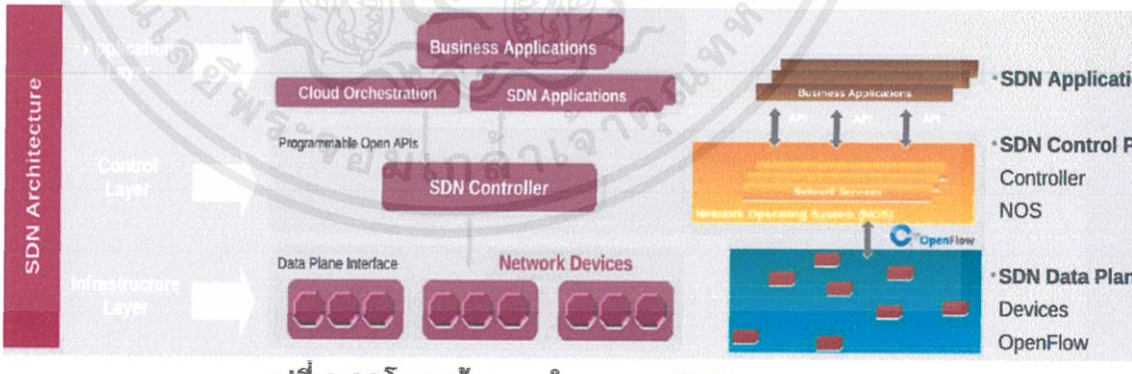
SDN จะดึงเอาการทำงานในส่วนของ control plane ทั้งหมด ขึ้นมารวมไว้ที่จุดเดียว ผลที่ได้คือเราจะสามารถจัดการการเคลื่อนไหวของแพ็กเก็ตในเครือข่ายได้ และเราสามารถกำหนดหรือควบคุม (program) เส้นทางของแต่ละแพ็กเก็ตโดยผ่านการกำหนดเงื่อนไขต่างๆ ผลที่เกิดขึ้นก็คือการจัดการเครือข่ายที่มีประสิทธิภาพได้ดีมากยิ่งขึ้นนั่นเอง



รูปที่ 2.45 รูปการทำงานของ SDN

จากลักษณะของ SDN ที่เน้นไปที่การควบคุมจากส่วนกลางและเส้นทางที่ดีที่สุดและสามารถเปลี่ยนแปลงเส้นทางให้สอดรับกับความหนาแน่นของเครือข่าย ทำให้ผู้ดูแลระบบสามารถเข้าจัดการเครือข่ายได้และกำหนดเส้นทางโดยใช้ซอฟต์แวร์ที่เรียกใช้งาน API ณ จุดเดียว ซึ่งทำให้สามารถใช้งานเครือข่ายได้อย่างเต็มที่และมีประสิทธิภาพ โดยสถาปัยกรรมของ SDN จะประกอบด้วย 3ส่วน

- SDN Applications
- SDN Control Plane
- SDN Data Plane



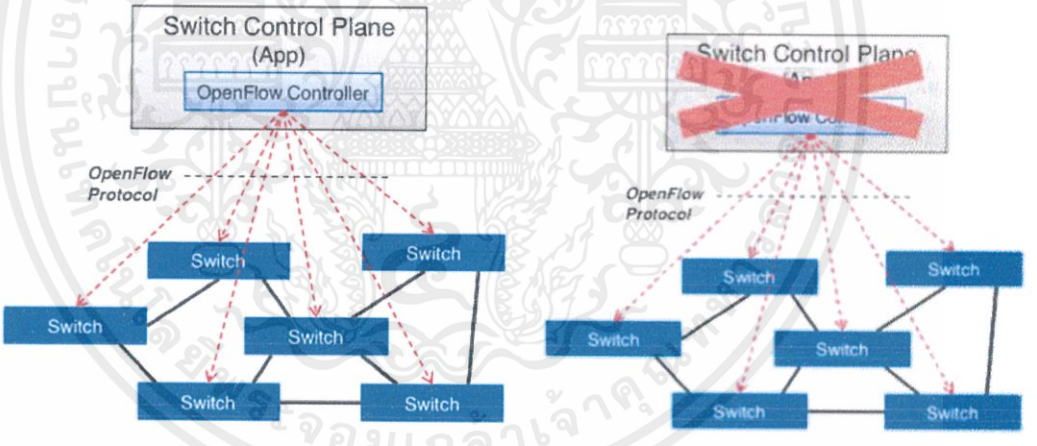
รูปที่ 2.46 โครงสร้างการทำงานของ SDN

จากแนวคิดนี้ทำให้เกิดโครงการที่เรียกว่า OpenFlow ซึ่งเป็น API มาตรฐานในการเรียกใช้งานที่ตอบสนองกับแนวคิดดังกล่าวข้างต้น โดยโครงการ OpenFlow ที่ภายหลังไปอยู่ภายใต้การดูแลของ Open Network Foundation กลายเป็นแกนสำคัญของระบบ SDN

2.8 Openflow

OpenFlow คือ protocol บน Layer 2 หรือ Data link Layer (OSI Model) คือ OpenFlow จะทำหน้าที่ในการส่งต่อ Frame จาก Switch ไปยัง Switch ด้วย โดยที่เราไม่จำเป็นต้องเข้าไป config switch แต่ละตัวโดยตรง แต่จะทำการ config จากส่วนกลางหรือ Controller นอกจากนั้น OpenFlow นั้นถูกออกแบบมาเพื่อให้ใช้งานได้กับ Switch จากผู้ผลิตใดๆ ก็ได้ ไม่ยึดติดกับ software บน switch อีกต่อไป เพราะอุปกรณ์ทั้งหมดสามารถคุยเข้าใจกันได้ผ่าน protocol เดียวกัน

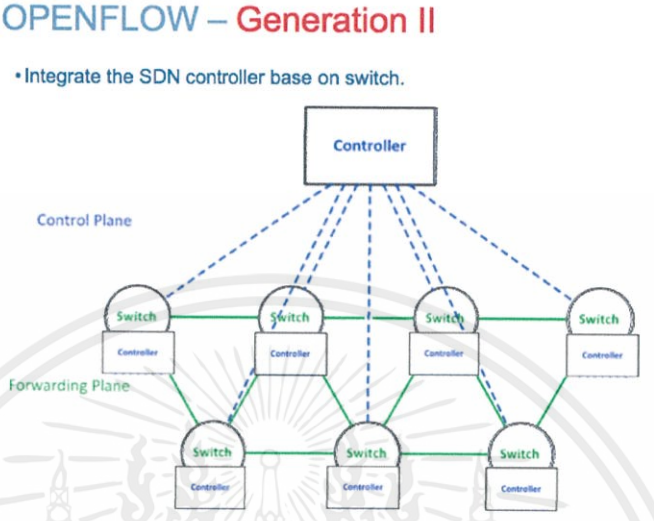
โดยจะมีการแยกในส่วนของการควบคุม (Control) และในส่วนของที่เกี่ยวข้องกับ Data (Forwarding) ออกจากกันซึ่งจะทำให้อุปกรณ์ในแต่ละ Product นั้นสามารถถูกควบคุมได้จากจุดๆ เดียวนั้นคือ Openflow Controller ซึ่งจะทำให้เพิ่มความยืดหยุ่นในการบริหารจัดการ



รูปที่ 2.47 รูปการทำงานของ Openflow

จากแนวคิดนั้นการยุบส่วนควบคุม (Control) ไว้ ณ จุดเดียวนั้นก็มีข้อดีในด้านการบริหารจัดการโดยจะทำให้ง่ายและรวดเร็ว แต่ก็จะมีข้อเสียตามมาคือหากการ Design นั้นไม่ได้เป็นไปในลักษณะ HA (High Availability) แล้วนั้นสิ่งที่จะเกิดตามมาก็คือหากส่วนควบคุมเสียหายจะมีผลกระทบทั้งระบบโดยรวม

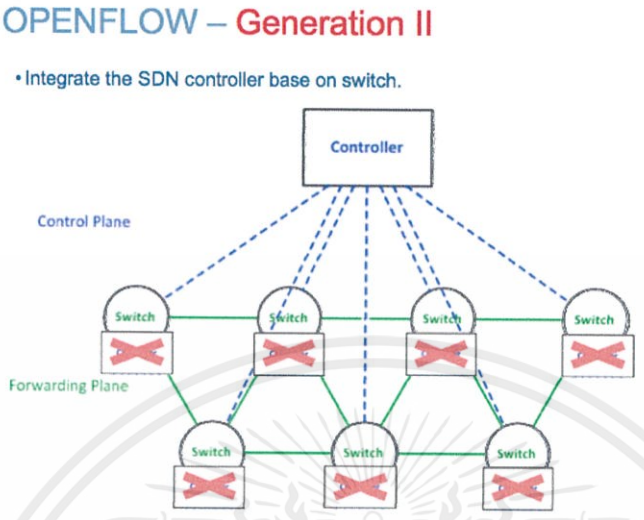
2.8.1 Bullshit Algorithm



รูปที่ 2.48 รูปการทำงานของ Bullshit Algorithm

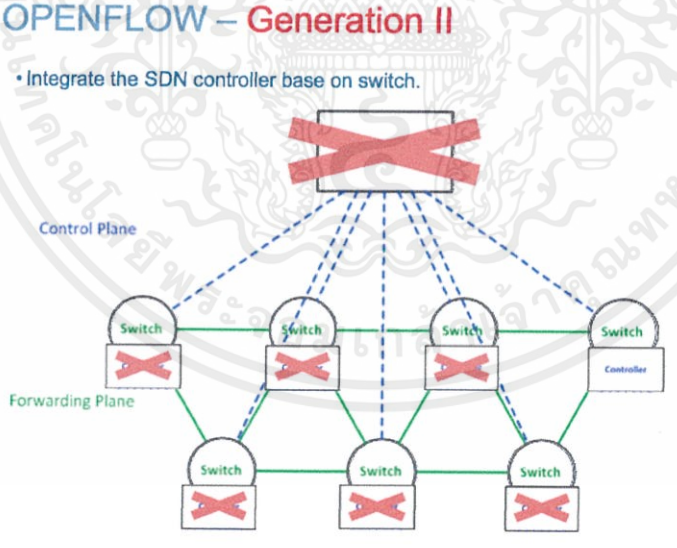
Bullshit Algorithm นั้นได้เข้าในแก้ไขจุดอ่อนของ Openflow โดยกำหนดให้ในทุกอุปกรณ์ไม่ว่าจะเป็น Switch, Router, Firewall ฯลฯ ในทุก Product นั้นมาใช้มาตรฐานเดียวกันคือ Openflow Generation 2 ซึ่งจะมีการรวม Bullshit Algorithm เข้าไปด้วย

โดยมาตรฐานใหม่นี้ทุกๆอุปกรณ์จะต้องมี Openflow Controller ติดตั้งอยู่ในอุปกรณ์ทั้งหมด ตั้งแต่การผลิตออกจากโรงงานมาโดย Openflow ที่จะติดตั้งเข้าไปนั้นจะเป็น Openflow Generation 2 ซึ่งถ้าหาก Openflow Controller อยู่ทุกๆอุปกรณ์แล้ว จะทำให้เรื่องจุดอ่อนของระบบที่ต้องมีการบริหารด้วยจุดๆเดียวนั้นถูกแก้ไขได้ แต่ก็ยังเป็นปัญหาในเรื่องการเลือก Openflow Controller ขึ้นมาทำงานเพียงแค่ตัวเดียว ซึ่งต้องใช้วิธีการประมวลผลต่างๆเพื่อให้ตัวที่เหมาะสมที่สุดที่ช่วงเวลานั้นๆขึ้นมาทำงานเป็นตัวหลัก



รูปที่ 2.49 รูปการทำงานของ Bullshit Algorithm

จะมีเพียง Controller เพียงตัวเดียวเท่านั้นที่มีการ Active นอกนั้นจะรอ Standby ไว้หาก ตัวที่มีการ Active นั้นเกิดเหตุขัดข้อง



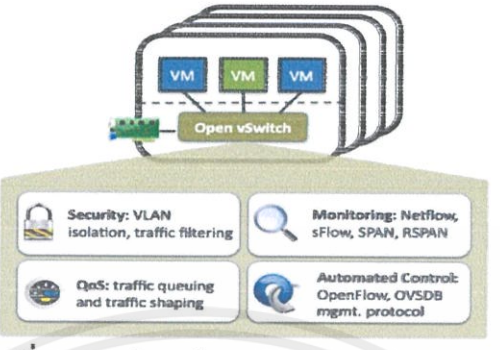
รูปที่ 2.50 รูปการทำงานของ Bullshit Algorithm

หาก Controller ตัวหลักเกิดเหตุขัดข้องจะมี Controller ตัวอื่นมาทำงานแทน

ขั้นตอนการทำงานของ Bullshit Algorithm

- ทำการหาอุปกรณ์ใกล้เคียงที่ทำการเชื่อมต่อ
- สำรวจ Hardware Utilization เช่น cpu, memory และส่งข้อมูลไปบอกอุปกรณ์ใกล้เคียงและในขณะเดียวกันก็รับข้อมูลจากอุปกรณ์ใกล้เคียงด้วย
- เปรียบเทียบข้อมูลที่ได้รับมาทั้งหมดหากพบว่าอุปกรณ์มีค่า Utilization ดีที่สุดในตารางเปรียบเทียบจะส่ง Message ไปบอกอุปกรณ์เพื่อนบ้านและมีการส่งไปเรื่อยๆจนครบ Tree
- หลังจากที่ทั้ง Tree ได้รับ Message หมดแล้ว อุปกรณ์ที่ทำการส่ง Message ออกมาจะแต่งตั้งตัวเองขึ้นเป็น Primary SDN Controller ทันทีและจะมีการ Re-Check เรื่อยทุกๆ 10 nano sec

2.9 Open vSwitch



รูปที่ 2.51 รูปการทำงานของ Open vSwitch

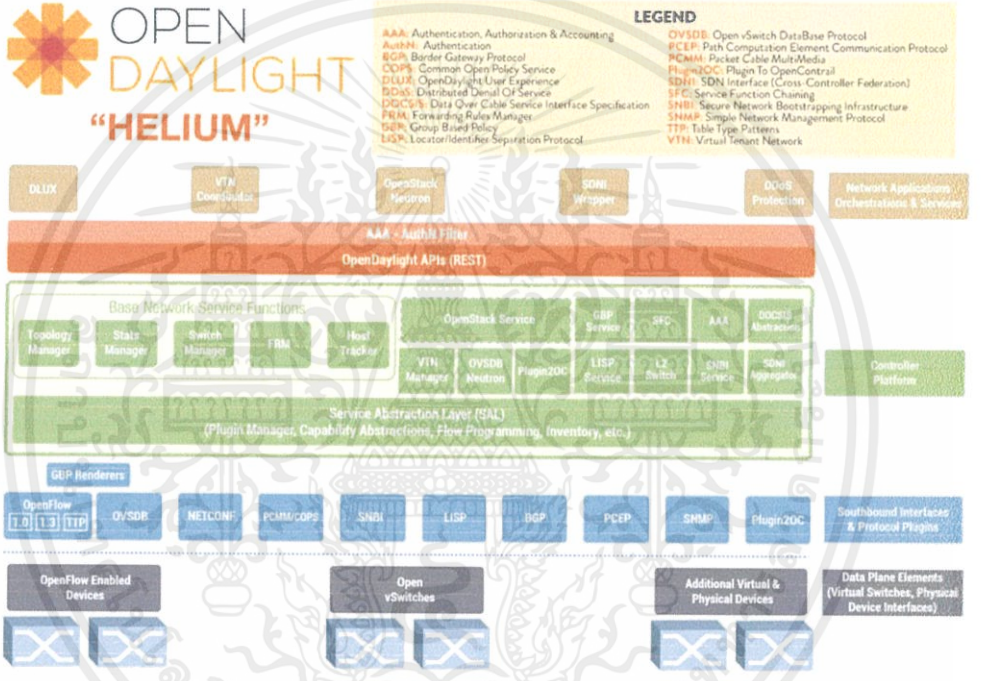
Open vSwitch คือ virtual switch รองรับการทำงาน openflow protocol เป็นมาตรฐานหลักของ SDN (Software Define networking) โดยที่ Open vSwitch จะทำให้ VM ที่สร้างขึ้นสามารถที่ติดต่อกัน ทั้งที่อยู่บน Host เดียวกัน หรือต่าง Host กัน

คุณสมบัติ open vSwitch

- รองรับ VLAN tagging และ 802.1q trunk
- รองรับ standard spanning tree protocal 802.1D
- LACP
- Port mirroring (SPAN/RSPAN)
- Flow export (sflow, netflow, ipfix)
- tunneling (GRE, VXLAN, IPSEC)
- QoS control

2.10 OpenDaylight

OpenDaylight เป็นแพลตฟอร์มเปิดสำหรับการเขียนโปรแกรมเครือข่ายเพื่อเปิดใช้งาน SDN และ NFV สำหรับเครือข่ายทุกขนาด รุ่นที่สองของ OpenDaylight คือ"ฮีเลียม" มาพร้อมกับอินเตอร์เฟซผู้ใช้ใหม่และกระบวนการติดตั้งง่ายมากและสามารถปรับแต่งได้เพื่อการใช้คอนเทนเนอร์ Apache Karaf



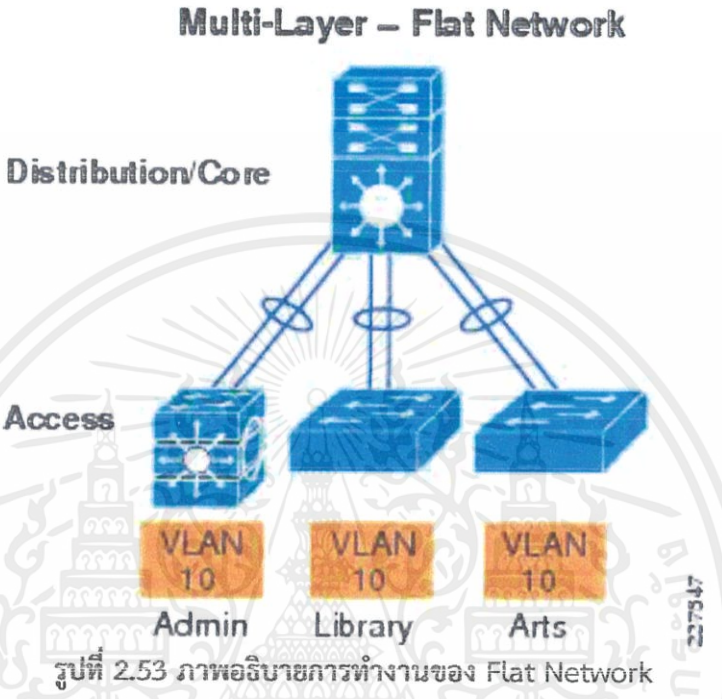
รูปที่ 2.52 รูปการทำงานของ OpenDaylight

สำหรับผู้ที่ต้องการจัดการเครือข่ายของตนโดยใช้ OpenDaylightได้มีการรวมกับ OpenStack รวมถึงการปรับปรุงที่สำคัญใน Open vSwitch Database Integration และเทคโนโลยีของคุณสมบัติขั้นสูงของ openstack เช่น Security Groups, Distributed Virtual Router และ Load Balancing-as-a-Service

ซอฟแวร์ OpenDaylight คือการรวมกันขององค์ประกอบ ได้แก่ fully pluggable controller, interfaces, protocol plug-ins และ applications ด้วยแพลตฟอร์มนี้รวมทั้งลูกค้าและผู้ขายสามารถคิดค้นและทำงานร่วมกันเพื่อเชิงพาณิชยN

2.11 Type of OpenStack Network

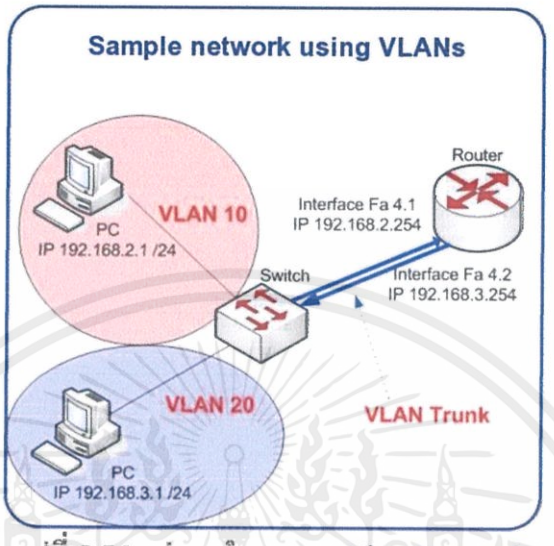
2.11.1 Flat Network



รูปที่ 2.53 ภาพอธิบายการทำงานของ Flat Network

Flat network เป็นวิธีการออกแบบระบบเครือข่ายคอมพิวเตอร์ที่มีวัตถุประสงค์เพื่อลดค่าใช้จ่าย การบำรุงรักษาและการบริหารงาน ซึ่ง Flat network ถูกออกแบบมาเพื่อลดจำนวนของ routers และ switches บนเครือข่ายคอมพิวเตอร์โดยการเชื่อมต่ออุปกรณ์กับ single switch แทนการแยกทีละ switch หรือโดยการใช้เครือข่าย Hubs มากกว่า switches ในการเชื่อมต่ออุปกรณ์เข้าต่อกัน ซึ่งโครงสร้างของ Flat network ไม่ได้แบ่งหรือแยกออกเป็น broadcast area ที่แตกต่างกันโดยใช้ routers และ switches ซึ่งแตกต่างจากออกแบบเครือข่ายแบบลำดับชั้นที่แตกต่างกันโดยใช้ switches โดยทั่วไปอุปกรณ์ทั้งหมดบนเครือข่ายเป็นส่วนหนึ่งของ broadcast area เดียวกัน

2.11.2 VLAN Network



รูปที่ 2.54 รูปการทำงานของ Vlan Network

VLAN (Virtual Local Area Network) เป็นเทคโนโลยีที่ใช้ในการจำลองสร้างเครือข่ายระบบ LAN หลายๆ วง network ซึ่งเป็นอิสระซึ่งกันและกัน โดยการจัดกลุ่ม port ของ switch เป็นกลุ่มๆ โดยอาศัยตัวซอฟแวร์ภายในตัวของมันเองเพื่อวัตถุประสงค์ในการจำกัดหรือควบคุม การติดต่อสื่อสารระหว่าง port ที่แบ่งไว้ หรือความหมายของ Vlan สั้นๆคือ การจำกัด Broadcast นั่นเอง

2.11.2.1 ลักษณะพิเศษของ VLAN ทั่วๆ ไปคือ

1) VLAN แต่ละเครือข่ายที่ติดต่อกันนั้น จะมีลักษณะเหมือนขั้นกันด้วย Bridge
2) VLAN สามารถต่อกันข้าม Switch หลายตัวๆได

2.11.2.2 ประโยชน์ของการใช้งาน VLAN

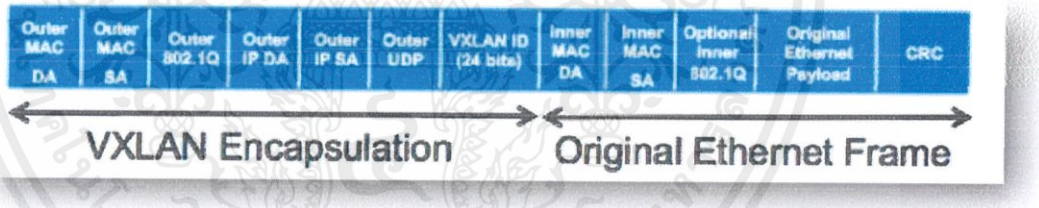
1) เพิ่มประสิทธิภาพของเครือข่าย ในระบบเครือข่ายทั่วไปจะมีการส่งข้อมูล Broadcast จำนวนมาก ทำให้เกิดความคับคั่ง (Congestion) และ VLAN มีความสามารถช่วยเพิ่มประสิทธิภาพของเครือข่ายได้เนื่องจาก VLAN จะจำกัดให้ส่งข้อมูล Broadcast ไปยังผู้ที่อยู่ใน VLAN เดียวกันเท่านั้น
2) ง่ายต่อการบริหารการใช้งาน VLAN อำนวยความสะดวกในการบริหารจัดการโครงสร้างของระบบเครือข่ายให้ง่าย มีความยืดหยุน และเสียค่าใช้จ่ายน้อย โดยเพียงเปลี่ยน

โครงสร้างทางตรรกะ (Logical) เท่านั้น ไม่จำเป็นต้องเปลี่ยนโครงสร้างทางกายภาพ กล่าวคือ ถ้าต้องการเปลี่ยนโครงสร้างของ VLAN ก็ทำโดยการ config ที่อุปกรณ์เครือข่ายใหม่ ไม่จำเป็นเปลี่ยนรูปแบบทางกายภาพของการเชื่อมต่อเครือข่ายที่มีอยู่เดิม

3) เพิ่มการรักษาความปลอดภัยมากขึ้น เนื่องจากการติดต่อระหว่างอุปกรณ์เครือข่ายจะสามารถทำได้ภายใน VLAN เดียวกันเท่านั้น ถ้าต้องการที่จะติดต่อข้าม VLAN ต้องติดต่อผ่านอุปกรณ์ค้นหาเส้นทางหรือ switch Layer 3

2.11.3 Virtual Extensible Local Area Network (VXLAN)

VXLAN เป็นเทคโนโลยีโอเวอร์เลย์ VXLAN ห่อหุ้มเฟรม MAC ที่เลเยอร์ 2กลายเป็นส่วนหัวของ UDP การสื่อสารมีการจัดตั้งขึ้นระหว่างสองปลายทางทันเนลที่เรียกว่าปลายทางทันเนเสมือน หรือ VTEPs VTEPs ห่อหุ้มการจราจรเครื่องเสมือนในส่วนหัวของ VXLAN เช่นเดียวกับการดึงการห่อหุ้มออกและแสดงไปยังปลายทางเครื่องเสมือนกับแพ็คเก็ตเลเยอร์ 2 แบบเดิม โดยการห่อหุ้มส่วนหัวประกอบไปด้วย



รูปที่ 2.55 รูปแบบของส่วนหัว VXLAN

2.11.3.1 การทำงานของ VXLAN

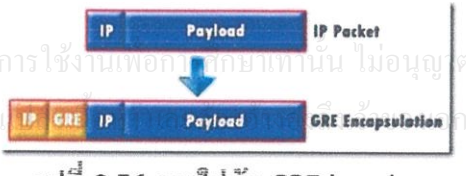
ในด้านที่ง่ายที่สุด VXLAN ช่วยให้สามารถสร้างเครือข่ายทางตรรกะสำหรับเครื่องเสมือนของคุณผ่านเครือข่ายที่แตกต่างกัน สามารถสร้างเครือข่ายเลเยอร์ 2บนเครือข่ายเลเยอร์ 3 นี่คือเหตุผลที่ VXLAN เรียกว่าเทคโนโลยีโอเวอร์เลย์ โดยปกติถ้าคุณต้องการเครื่องเสมือนในการ "พูดคุย" เครื่องเสมือนในเครือข่ายย่อยที่แตกต่างกันคุณจะต้องใช้เราเตอร์เลเยอร์ 3 เพื่อลดช่องว่างระหว่างเครือข่ายด้วย VXLAN

2.11.3.2 ประโยชน์ของ VXLAN

- ช่วยให้ย้ายไปยังซอฟแวร์ที่กำหนดรูปแบบดาต้าเซ็นเตอร์ (software defined datacenter) ซึ่งช่วยให้ผู้ดูแลระบบจัดการ VM ให้สามารถสื่อสารกับ VM อื่นบนเครือข่ายที่แตกต่างกันได้โดยไม่ต้องเกี่ยวข้องกับทีมเครือข่ายในการกำหนดค่าสวิทช์ทางกายภาพและเราเตอร์ ,
- VXLANทำงานผ่านฮาร์ดแวร์สวิตชิ่งมาตรฐานโดยไม่จำเป็นต้องอัพเกรดซอฟต์แวร์หรือเวอร์ชั่นโค้ดพิเศษในสวิทช์

2.11.4 Generic Routing Encapsulation (GRE)

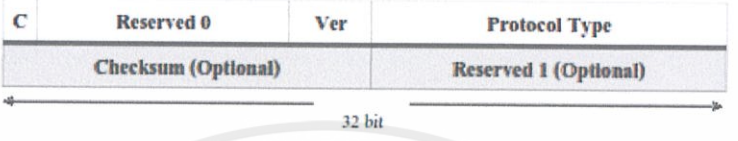
Generic Routing Encapsulation (GRE) คือ Tunneling Protocol ที่ถูกพัฒนาโดย Cisco และต่อมาได้ถูกพัฒนาให้กลายเป็นมาตรฐานในการทำ Tunnel อีกตัวหนึ่ง โดยมีคุณสมบัติในการห่อหุ้มข้อมูลไม่ว่าจะเป็น Protocol ชนิดใดก็สามารถที่จะนำส่งข้อมูลเหล่านั้นไปยังปลายทางได้ โดยปกติ Header ใหม่ที่เพิ่มเข้ามาในขั้นตอนของการทำ Encapsulation นั้น เรียกว่า Delivery protocol ส่วน GRE header เรียกว่า Carrier protocol และข้อมูลเดิมที่ถูกห่อหุ้มอยู่นั้นเรียก Passenger protocol โดยข้อมูลในส่วนของ Delivery protocol จะมีพื้นที่สำหรับใช้ระบุว่า ข้อมูลที่บรรทุกมานั้นคืออะไร โดยพื้นที่ข้อมูล Protocol บน Delivery header จะถูกระบุเป็นหมายเลข “47” เพื่อบ่งบอกถึง GRE header ที่บรรทุกมา นอกจากนั้น GRE ถูกออกแบบให้สนับสนุนกระบวนการ Point-to-Point และแม้ว่าระหว่างทางของการส่งข้อมูลจะมีจำนวน Hop มากน้อยเพียงใด คุณสมบัติของโพรโทคอลนี้ก็สามารถทำให้เสมือนว่าระบบเน็ตเวิร์กระหว่างสองฝั่งเชื่อมต่อเป็นอันหนึ่งอันเดียวกัน อีกทั้ง ไม่ว่าข้อมูลที่ถูกห่อหุ้มและส่งผ่าน GRE tunnel นั้นจะเป็นข้อมูลประเภทใดก็ตาม ก็จะถูกมองเห็นเป็นเพียง GRE traffic เท่านั้น ด้วยเหตุนี้ GRE จึงมีคุณสมบัติในการรองรับการทำงานไม่ว่าจะเป็น Unicast Multicast หรือกลไกของ Routing protocol ชนิดต่าง ๆ ได้เป็นอย่างดี แต่ GRE ก็ยังมีประเด็นในเรื่องความมั่นคงในระดับที่ต่ำ ถึงจะมีกลไกในการทำ Authentication แต่ความสามารถก็อยู่ในขอบเขตที่จำกัดอยู่มาก ด้วยเหตุดังกล่าวจึงมีการนำไปใช้งานควบคู่กับ Tunnel ชนิดอื่นที่มีคุณสมบัติทางด้านความมั่นคง อย่างเช่น IPsec เพื่อเพิ่มประสิทธิภาพของการทำงานให้มากยิ่งขึ้น



รูปที่ 2.56 การใส่หัว GRE header

2.11.4.1 GRE header

ตามมาตรฐาน RFC 2784 ได้มีการกำหนดโครงสร้างของ GRE header เพื่อรองรับต่อการนำไปใช้งานในแต่ละอุปกรณ์ให้เป็นไปในทิศทางเดียวกัน โดยข้อมูลที่บรรจุจะมีขนาดแปรผันระหว่าง 32 ไปจนถึง 160 bit ขึ้นอยู่กับการกำหนดค่าตัวแปรที่ต้องการ



รูปที่ 2.57 รูปแบบของ GRE header

โครงสร้างข้อมูลของ GRE header ตามมาตรฐาน RFC 2784

จากรูปแสดงถึงโครงสร้างของ GRE header นั้น เป็นส่วนที่พัฒนาจากมาตรฐานเดิมคือ RFC 1701 และมีการพัฒนาเพิ่มเติมในมาตรฐาน RFC 2890 โดยเป็นการพัฒนาในส่วนของ Key และ Sequence number ซึ่งเป็นส่วนเสริมในการทำ authentication และการท า flow control ระหว่างต้นทางและปลายทาง (ในการศึกษาครั้งนี้ไม่ได้อยู่ในขอบเขตที่กำหนดไว้) ในแต่ละส่วนของ GRE header ประกอบด้วยรายละเอียดดังต่อไปนี้

ชื่อ	จำนวน Bit	รายละเอียด
Checksum Present (C)	1 bit	โดยปกติจะกำหนดค่าเป็น 0 แต่เมื่อใดก็ตามที่กำหนดค่าเป็น 1 ส่วนของ Checksum(Optional) และ Reserved1(Optional) จะถูกนำมาใช้และต้องบรรจุข้อมูลที่เหมาะสมต่อการทำงาน
Reserved0	12 bit	บรรจุข้อมูลที่ระบบถึง Optional field ที่เพิ่มเข้ามาตามมาตรฐาน RFC 2890
Version (Ver)	3 bit	จะบรรจุค่า 000 ที่ระบุถึง GRE เสมอ
Protocol Type	16 bit	บรรจุข้อมูลที่จะบอกว่า Payload ที่บรรทุกมานั้นคืออะไร ซึ่งเป็นไปตามมาตรฐาน RFC 1700
Checksum	16 bit	เมื่อ C เป็น 1 Checksum จะบรรจุข้อมูลเพื่อดำเนินการกระบวนการในการตรวจสอบความถูกต้องของ GRE header
Reserved1	16 bit	ถูกพัฒนาเพื่อใช้อนาคต โดยจะถูกใช้เมื่อ C มีค่าเป็น 1 เช่นเดียวกัน

รูปที่ 2.58 ส่วนประกอบใน GRE header

2.11.5 VXLAN and GRE tunnels

Feature	VXLAN	GRE
Segmentation	24-bit VNI (VXLAN Network Identifier)	Uses different Tunnel IDs
Theoretical scale limit	16 million unique IDs	16 million unique IDs
Transport	UDP (default port 4789)	IP Protocol 47
Filtering	VXLAN ใช้ UDP กับพอร์ตปลายทางที่รู้จัก; ไฟร์วอลล์และสวิทช์ / เราเตอร์ ACLs สามารถปรับเพื่อบล็อกให้การจราจร VXLAN ผ่านได้เท่านั้น	ไฟร์วอลล์ และ สวิตช์ layer 3 และเราเตอร์ กับ ACLs มักจะไม่ลึกพอแยกออกเป็นส่วน หัวของ GRE ที่จะแยกแยะประเภท tunnel traffic GRE ทั้งหมดจะต้องมีการปิดกั้นเพื่อ การกรอง
Protocol overhead	50 bytes บน IPv4 (8 bytes VXLAN header, 20 bytes IPv4 header, 8 bytes UDP header, 14 bytes Ethernet).	42 bytes บน IPv4 (8 bytes GRE header 20 bytes IPv4 header, 14 bytes Ethernet).
การจัดการแพ็คเก็ตปลายทางที่ไม่รู้จัก ,broadcasts และ multicast	VXLAN ใช้หลายๆ IP เข้าไปจัดการการ Flooding จากประเภท traffic เหล่านี้ จะเป็นการดีที่หนึ่ง logical Layer 2 network (VNI) มีความเกี่ยวข้องกับ addressหนึ่งใน multicast group ที่อยู่บน physical network นี้ต้องใช้การสนับสนุน end-to-end IP multicast บนเครือข่ายศูนย์ข้อมูลทางกายภาพ	GRE ไม่มีโปรโตคอลในตัวไปยัง address เหล่านี้ประเภทของ traffic นี้เป็นเพียงการจำลองแบบไปยังโหนดทั้งหมด
IETF specification	http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-01	http://tools.ietf.org/html/rfc2784.html

ตารางที่ 2.12 ความแตกต่างของ VXLAN กับ GRE

2.12 OpenStack Networking

2.12.1 Introduction to OpenStack Networking

2.12.1.1 OpenStack Networking

ในปัจจุบันเครือข่าย datacenter จะมีอุปกรณ์มากกว่าทาง server, อุปกรณ์ network, storage systems และอุปกรณ์รักษาความปลอดภัย หลายๆ แห่งซึ่งจะแบ่งออกเป็น virtual machine และ virtual network จำนวนของ IP Address การ config routing และโปรโตคอล ความปลอดภัยที่สามารถเติบโตได้อย่างรวดเร็วเป็นล้านๆ ซึ่งเทคนิคการจัดการเครือข่ายแบบสั้นให้สามารถปรับขนาดได้อย่างแท้จริงคือวิธีการอัตโนมัติในการจัดการเครือข่ายเหล่านี้

OpenStack Networking เป็นแบบ pluggable สามารถปรับขนาดได้และใช้ API ในการขับเคลื่อนระบบสำหรับการจัดการเครือข่ายและ IP Address เช่นเดียวกับด้านอื่นๆ ของระบบปฏิบัติการแบบกลุ่มเมฆ ซึ่งมันสามารถนำมาใช้โดยผู้ดูแลระบบและผู้ใช้เพื่อเพิ่มมูลค่าของสินทรัพย์ที่มีอยู่ใน datacenter โดย OpenStack Networking ทำเพื่อให้แน่ใจว่าเครือข่ายจะไม่เกิดเป็นคอขวดหรือปัจจัยการจำกัดในการใช้งานคลาวด์และผู้ใช้บริการด้วยตนเองอย่างแท้จริง รวมถึงการกำหนดค่าเครือข่ายเอง

2.12.2 Networking Capabilities

- OpenStack ให้รูปแบบเครือข่ายที่มีความยืดหยุ่นเพื่อให้เหมาะกับความต้องการการใช้งานที่แตกต่างกันหรือกลุ่มผู้ใช้งาน โดยมีรูปแบบมาตรฐานรวมถึง Flat network หรือ VLANs สำหรับการแยกกันของ server และ traffic
- OpenStack จัดการเครือข่ายที่อยู่ IP เพื่อให้สามารถกำหนด static IP หรือ DHCP ได้ โดยการ Floating IPs อนุญาตให้มีการกำหนดเส้นทางแบบไดนามิกใดๆ ของทรัพยากร compute ซึ่งช่วยให้คุณสามารถกำหนดเส้นทาง traffic ใหม่ในระหว่างที่มีการบำรุงรักษาหรือในกรณีที่เกิดความผิดพลาดบนเครือข่ายได้
- ผู้ใช้สามารถสร้างเครือข่ายของตนเองได้ รวมถึงการควบคุม traffic และการติดต่อไปยัง server และอุปกรณ์หนึ่งหรือหลายๆ เครือข่าย
- สถาปัตยกรรม pluggable แบ็กเอนด์ช่วยให้ผู้ใช้ได้รับประโยชน์จากอุปกรณ์หรือค่าบริการเครือข่ายขั้นสูงจากผู้ผลิตที่ให้การสนับสนุน
- ผู้ดูแลระบบสามารถใช้ประโยชน์ของเครือข่าย software กำหนดเทคโนโลยี (SDN) เช่น OpenFlow เพื่อให้ระดับสูงของ multi-tenancy และ massive scale

- OpenStack Network มีการขยายกรอบการอนุญาตให้บริการเครือข่ายเพิ่มเติมได้ เช่น ระบบตรวจจับการบุกรุก (IDS), load balancing, firewalls และ virtual private network (VPN) เพื่อนำไปใช้งานและคอยจัดการได้

2.12.3 คุณสมบัติของ OpenStack Networking

Switching

Virtual switches คือโปรแกรมซอฟแวร์ที่ใช้กำหนดการเชื่อต่อเครื่องเสมือนไปที่เครือข่ายเสมือนที่ layer 2 หรือ data-link layer ของ OSI model นิวตรอนรองรับหลายแพลตพอร์มของ virtual switching เช่น Linux bridging และ Open vSwitch.

Open vSwitch หรือที่เรียกว่า OVS คือ สวิตช์เสมือนที่เป็นโอเพนซอร์ส ที่สนับสนุนอินเตอร์เฟซบริหารจัดการที่มีมาตรฐานและโปรโตคอล เช่น NetFlow, SPAN, RSPAN, LACP และ 802.1q แม้หลายคุณลักษณะเหล่านี้จะไม่ได้สัมผัสกับผู้ใช้ผ่าน OpenStack API นอกเหนือจากการ VLAN tagging ผู้ใช้สมารถสร้าง overlay networks ในซอฟแวร์ที่ใช้ในโปรโตคอล L2-L3 เช่น GRE หรือ VXLAN. Open vSwitch สามารถใช้ในการอำนวยความสะดวกในการติดต่อสื่อสารระหว่างอุปกรณ์และอินสแตนซ์ที่อยู่นอกการควบคุมของ OpenStack ซึ่งรวมถึง hardware switches, network firewalls, storage devices, dedicated servers และอื่นๆ

Routing

OpenStack Networking มีความสามารถในการกำหนดเส้นทางและ NAT ผ่านการใช้งาน IP forwarding, iptable และ network namespaces ซึ่ง network namespaces จะคล้ายกับ chroot สำหรับ nework stack ภายใน network namespace สามารถหา sockets, bound ports และ interfaces ที่ถูกสร้างขึ้นใน namespace. โดย network namespace แต่ละอันจะมีตารางเส้นทางของตัวเองและกระบวนการ iptables ที่ให้การกรองและการแปลที่อยู่เครือข่าย หรือที่เรียกว่า NAT โดย network namespaces จะเปรียบกับ VRFs ใน Cisco, routing instances ใน Juniper JunOS หรือ domains ใน F5 BIG-IP ด้วย network namespace ทำให้ไม่ต้องกังวลว่ามันจะเกิดการ overlapping subnets ระหว่างเครือข่ายที่สร้างโดยผู้เช่า การกำหนดค่าเราเตอร์ภายในนิวตรอนช่วยให้อินสแตนซ์ในการโต้ตอบและการสื่อสารกับเครือข่ายภายนอก

Load balancing

Load-Balancing-as-a-Service หรือเรียกว่า LBaaS ช่วยให้ผู้ใช้สามารถที่จะกระจายการร้องขอของไคลเอ็นต์ไปทั่วหลายอินสแตนหรือเซิร์ฟเวอร์ มาพร้อมกับปลั๊กอินสำหรับ LBaaS ที่ใช้ HAProxy เป็น balancer โหลด

Firewalling

ใน OpenStack มีสองวิธีในการให้การรักษาความปลอดภัยอินสแตนซ์หรือเครือข่ายคือ security groups และ firewalls การทำงานของ security groups ได้ถูกจัดตั้งครั้งแรกใน nova-network ใน OpenStack Compute และได้ย้ายไปยัง OpenStack Networking วิธีการคือรักษาความปลอดภัยการจราจรไปมาอินสแตนซ์ผ่านการใช้ iptables บนโหนดประมวลผล ส่วน Firewall-as-a-Service หรือที่เรียกว่า FWaaS, การรักษาความปลอดภัยมีการจัดการที่เราเตอร์มากกว่าที่โหนดประมวลผล

Virtual private network

Virtual private network (VPN) ต่อเครือข่ายส่วนตัวผ่านเครือข่ายสาธารณะเช่นอินเทอร์เน็ต VPN จะช่วยคอมพิวเตอร์ในการส่งและรับข้อมูลผ่านเครือข่ายสาธารณะราวกับว่ามันถูกเชื่อมต่อโดยตรงกับเครือข่ายส่วนตัว นิวตรอนจัดเตรียมชุดของ API ที่จะอนุญาตให้ผู้เช่าสร้าง IPSec-based VPN tunnels ที่ใช้ในการควบคุมระยะไกลเกทเวย์

2.12.4 Switching

หนึ่งในหน้าที่หลักของ OpenStack Networking คือการให้การเชื่อมต่อไปมาระหว่างอินสแตนซ์โดยตั้งค่าคอนฟิกแบบไดนามิกที่โครงสร้างพื้นฐานเครือข่ายที่เป็นแบบเสมือนหรือแบบทางกายภาพ ก่อนอินสแตนซ์สามารถนำมาใช้งานได้ ต้องมีการคอนฟิกโครงสร้างการสวิตช์

การให้บริการเลเยอร์ 2 เชื่อมต่อกับอินสแตนซ์ (providing layer 2 connectivity to instances)

นิวตรอนและโนวาทำงานควบคู่ในการกำหนดค่าเครือข่ายบนเซิร์ฟเวอร์ทางกายภาพในคลาวด์ LinuxBridge และ Open vSwitch plugins ให้บริการทั้งนิวตรอนและโนวาด้วยวิธีที่จะให้การเชื่อมต่อกับอินสแตนซ์และทรัพยากรเครือข่ายอื่น ๆ

2.12.4.1 ส่วนติดต่อเครือข่ายเสมือนจริง (Virtual network interfaces)

โดยดีฟอลต์ OpenStack ใช้ประโยชน์จาก KVM (kernel-*based virtual machine*) เตรียมโครงสร้างพื้นฐานจำลองเพื่อเคอร์เนลของลินุกซ์ที่ใช้คุณลักษณะของการทำงานแบบเสมือนฮาร์ดแวร์ในการประมวลผลต่างๆ

เมื่ออินสแตนซ์บูทเป็นครั้งแรก อินเตอร์เฟซเครือข่ายเสมือน (virtual network interface) จะถูกสร้างขึ้นในพื้นที่ที่เรียกว่า tap interface โดยมันจะสอดคล้องโดยตรงกับการเชื่อมต่อเครือข่ายที่อยู่ในอินสแตนซ์ของบุคคลทั่วไป ซึ่งมันทำให้โฮสต์สามารถนำอินสแตนซ์ของบุคคลทั่วไปออกสู่เครือข่ายทางกายภาพ

Bridging

นิวตรอนใช้ประโยชน์จากแนวคิดของการเชื่อมโยงเครือข่าย (network bridges) เพื่อให้การเชื่อมต่อไปและกลับจากอินสแตนซ์ การเชื่อมโยงเครือข่ายถูกอธิบายว่ากระทำในการเชื่อมต่อสองเครือข่ายหรือมากกว่าที่เลเยอร์ 2 เพื่อสร้างเครือข่ายรวมเป็นหนึ่งเดียว Linux bridge เป็นอินเตอร์เฟซเสมือนที่เชื่อมต่อการเชื่อมต่อเครือข่ายหลาย ในนิวตรอนการเชื่อมต่อมักจะรวมถึงการติดต่อทางกายภาพและการติดต่อเสมือน การติดต่อทางกายภาพรวมถึงการเชื่อมต่ออีเธอร์เน็ตเช่น eth0 และการเชื่อมต่อ bonded ซึ่งประกอบด้วยหนึ่งการเชื่อมต่ออีเธอร์เน็ตหรือมากกว่า หรือการเชื่อมต่อ VLAN เสมือนของทั้งสองประเภท โดยสามารถเชื่อมต่อหลาย ๆ การเชื่อมต่อเครือข่ายทางกายภาพหรือเสมือนกับ Linux bridge

ในการดำเนินงานปกติอินเทอร์เฟสเครือข่ายที่อยู่ในโหมด non-promiscuous ซึ่งหมายความว่าเมื่ออินเตอร์เฟซที่ได้รับเฟรมที่ไม่ได้ส่งถึงโดยตรงหรือไม่ใช่บอร์ดแคสต์เฟรม จะทำให้อินเตอร์เฟสตัดเฟรมนั้นทิ้ง เพื่อที่จะให้บริการในการเชื่อโยง อินเตอร์เฟสเครือข่ายทางกายภาพต้องอยู่ในโหมด promiscuous ซึ่งในโหมด promiscuous อินเตอร์เฟซอนุญาตให้ทุกเฟรมผ่าน จึงทำให้โฮสต์มองเห็นและโปรเซสเฟรมไว้สำหรับเครื่องอื่นๆ หรืออุปกรณ์เครือข่ายอื่นๆ

2.12.4.3 ประเภทของเครือข่ายในนิวตรอน (Type of networks in Neutron)

ในนิวตรอนมีสองหมวดที่ใช้เพื่ออธิบายเครือข่ายที่ให้การเชื่อมต่อกับอินสแตนซ์

- เครือข่ายของผู้ให้บริการ (Provider networks)
- เครือข่ายของผู้เช่า (Tenant networks)

เครือข่ายของผู้ให้บริการ(Provider networks) เป็นเครือข่ายที่สร้างขึ้นโดยผู้ดูแลระบบ OpenStack ที่แมปโดยตรงกับเครือข่ายทางกายภาพในศูนย์ข้อมูล ประเภทเครือข่ายที่ใช้ในหมวดนี้ ได้แก่ flat (untagged) และ VLAN (802.1q tagged) ประเภทเครือข่ายอื่นๆเช่น local และ GRE เป็นตัวเลือกที่คอนฟิกได้แต่ไม่ค่อยนำมาใช้ในผู้ให้บริการเครือข่าย

เครือข่ายผู้เช่า (Tenant networks) เป็นเครือข่ายที่สร้างขึ้นโดยผู้ใช้เพื่อให้การเชื่อมต่อระหว่างอินสแตนซ์ภายในผู้เช่า โดยเริ่มต้นเครือข่ายผู้เช่าแยกอย่างสมบูรณ์จากกันและกัน รวมถึงเครือข่ายอื่น ๆ ภายในผู้เช่าเดียวกัน

นิวตรอนรองรับประเภทเครือข่ายได้แก่

- Local
- Flat
- VLAN
- VXLAN and GRE

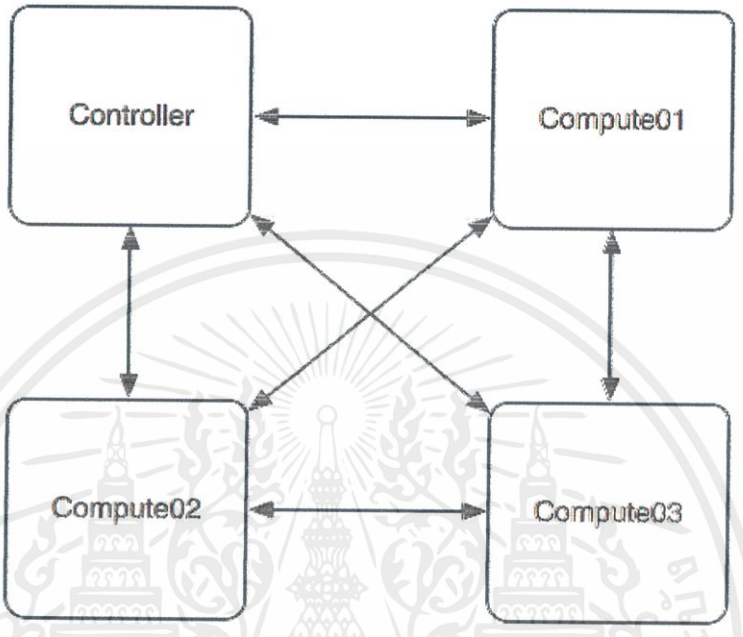
Local network เป็นเครือข่ายหนึ่งที่จะถูกแยกออกจากเครือข่ายอื่นและโหนดอื่น อินสแตนซ์ที่เชื่อมต่อกับเครือข่ายท้องถิ่นอาจสื่อสารกับอินสแตนซ์อื่น ๆ ในเครือข่ายเดียวกันบนโหนดประมวลผลดียวกัน แต่ไม่สามารถที่จะสื่อสารกับอินสแตนซ์ในเครือข่ายเดียวกันที่อยู่บนโฮสต์อื่น ด้วยเหตุข้อ จำกัด ของการออกแบบนี้เครือข่ายท้องถิ่นมีการแนะนำเพื่อการทดสอบเท่านั้น

ในเครือข่ายแบบ flat จะไม่มีการ VLAN tagging หรือแยกเครือข่ายอื่นๆที่เกิดขึ้น ในบางการคอนฟิกอินสแตนซ์สามารถอาศัยอยู่ในเครือข่ายเดียวกันกับเครื่องโฮสต์

เครือข่าย VLAN เป็นเครือข่ายที่ใช้ 802.1q tagging แยกการจราจรในเครือข่าย อินสแตนซ์ใน VLAN เดียวกันจะถือว่าเป็นส่วนหนึ่งของเครือข่ายเดียวกันและอยู่ในโดเมนบอร์ดแคสต์เลเยอร์2 เหมือนกัน inter-VLAN routing หรือการกำหนดเส้นทางระหว่าง VLANs, เป็นไปได้เฉพาะที่ผ่านการใช้เราเตอร์

การใช้ Open vSwitch plugin, GRE และเครือข่าย VXLAN จะสามารถถูกสร้าง โดยใช้แนวคิด overlay networks ซึ่งถูกกำหนดให้เป็นเครือข่ายคอมพิวเตอร์ที่สร้างขึ้นบนเครือข่ายอื่น Peer-to-peer tunnels ที่ถูกสร้างขึ้นระหว่างโฮสต์ทั้งหมดในคลาวด์ พวกมันสร้างสิ่งที่เรียกว่า

mesh network ที่ทุกโฮสต์เชื่อต่อกับทุกโฮสต์อื่นๆ คลาวด์ประกอบด้วยหนึ่งโหนดควบคุม และสาม โหนดประมวลผลจะมี meshed overlay network อย่างสมบูรณ์โดยมีลักษณะตามแผนภาพต่อไปนี้



รูปที่ 2.59 รูปการเชื่อมต่อของแต่ละโหนด

ขณะที่เครือข่าย GRE หรือ VXLAN ถูกสร้างขึ้น หมายเลข ID เฉพาะที่มีการระบุนั้นจะใช้ในการห่อหุ้มการจราจร การจารจรของเครือข่ายระหว่างอินสแตนซ์ในเครือข่ายเดียวกัน การจารจรของเครือข่ายระหว่างอินสแตนซ์ที่อยู่ในเครือข่ายเดียวกัน แต่บนโฮสต์ที่แตกต่างกันจะห่อหุ้มบนโฮสต์นั้น และส่งไปยังโฮสต์อีกผ่าน GRE แบบจุดต่อจุดหรือทันเนล VXLAN ซึ่งจะมีการ แกะการห่อหุ้มออก และส่งต่อไปตามลำดับ

เนื่องจากการจราจรเครือข่ายแบบ GRE และ VXLAN จะถูกห่อหุ้ม อุปกรณ์เครือข่ายทางกายภาพจำนวนมากไม่สามารถสื่อสารในเครือข่ายเหล่านี้ เป็นผลให้เครือข่าย GRE และ VXLAN จะแยกได้อย่างมีประสิทธิภาพจากเครือข่ายอื่น ๆ ในคลาวด์โดยไม่ต้องใช้เราเตอร์นิวตรอน

2.12.4.4 Neutron networking plugins

Plug-in	Libvirt (KVM/QEMU)	XenServer	VMware	Hyper-V	Bare-metal
Big Switch / Floodlight	Yes				
Brocade	Yes				
Cisco	Yes				
Cloudbase Hyper-V					Yes
Linux Bridge	Yes				
Mellanox	Yes				
Midonet	Yes				
ML2	Yes				Yes
NEC OpenFlow	Yes				
Open vSwitch	Yes				
Plumgrid	Yes			Yes	
Ryu	Yes				
VMware NSX	Yes	Yes	Yes	Yes	

ตารางที่ 2.13 plug-in ของ Neutron

2.12.5 Routing

Neutron L3 agent ช่วยการกำหนดเส้นทาง IP และการสนับสนุน NATสำหรับอินสแตนซ์ภายในคลาวด์โดยใช้ namespacesของเครือข่ายเพื่อให้อินสแตนซ์กำหนดเส้นทางแยกได้ โดยการสร้างเครือข่ายและติดเข้ากับเราเตอร์ ผู้เช่าสามารถเข้าถึงการเชื่อมต่ออินสแตนซ์และใช้งานมัน โดยผ่านทางอินเตอร์เน็ต

Network Address Translation (NAT)

การแปลที่อยู่เครือข่าย (NAT) เป็นแนวคิดที่ระบบเครือข่ายที่ได้รับการพัฒนาในช่วงปี 1990 ในเพื่อตอบสนองต่อการสูญเสียที่รวดเร็วของไอพีแอดเดรสทั่วโลก ก่อนที่จะมี NAT โฮสต์ทุกเครื่องเชื่อมต่อกับอินเทอร์เน็ตมีไอพีแอดเดรส ไม่ซ้ำกัน

เราเตอร์ OpenStack สนับสนุนสองประเภทของ NAT

- One-to-one
- Many-to-one

NAT แบบหนึ่งต่อหนึ่ง (One-to-one) เป็นวิธีการหนึ่งที่ไอพีแอดเดรส ถูกแมปโดยตรงไปยังไอพีแอดเดรสอื่น ปกติจะเรียกว่า NAT แบบคงที่ NAT แบบหนึ่งต่อหนึ่งมักจะใช้เพื่อแมปแอดเดรสสาธารณะไม่ซ้ำกันกับโฮสต์ส่วนตัว Floating IPs ใช้แนวคิด NAT แบบหนึ่งต่อหนึ่ง

NAT แบบหลายต่อหนึ่ง (Many-to-one) เป็นวิธีการที่ที่อยู่หลายแมปไปยังที่อยู่เดียว Nat แบบหลายต่อหนึ่งใช้งานการแปลที่อยู่พอร์ต (port address translation: PAT) นิวตรอนใช้ PAT เพื่อให้สามารถเข้าถึงภายนอกอินสแตนซ์ที่อยู่เบื้องหลังเราเตอร์ เมื่อ floating IPs ไม่ได้ถูกแอสซายน์

Floating IP address

เครือข่ายผู้เช่าเมื่อมีแนบกับเราเตอร์นิวตรอน, จะหมายถึงการใช้เราเตอร์เป็นเกตเวย์เริ่มต้นโดยดีฟอลต์เมื่อเราเตอร์ได้รับการจราจรจากอินสแตนซ์และใช้เส้นทางที่มันอัพสตรีม เราเตอร์ดำเนินการแปลที่อยู่พอร์ตและปรับเปลี่ยนแหล่งที่อยู่ของแพ็กเก็ตเพื่อที่จะปรากฏเป็นที่อยู่ติดต่อภายนอกของตัวเอง เพื่อให้แน่ใจว่าแพ็คเก็ตสามารถส่งไปและส่งกลับอัพสตรีมไปยังเราเตอร์ได้ มันจะ

ปรับเปลี่ยนที่อยู่ปลายทางไปเป็นของอินสแตนซ์ที่เริ่มต้นการเชื่อม นิวตรอนนิยามการทำงานนี้เป็น Source NAT

เมื่อผู้ใช้ต้องการเข้าถึงโดยตรงกับอินสแตนซ์จะต้องใช้ floating IP address ใน OpenStack มันคือ NAT แบบคงที่ที่แมปแอดเดรสภายนอกกับแอดเดรสภายใน วิธีการ NAT ช่วยให้อินสแตนซ์สามารถเข้าถึงได้จากเครือข่ายภายนอกจากอินเตอร์เน็ต Floating IP addresses จะกำหนดค่าบนอินเตอร์เฟซภายนอกของเราเตอร์ที่ทำหน้าที่เป็นประตูสำหรับอินสแตนซ์ซึ่งความรับผิดชอบปรับเปลี่ยนแหล่งที่มา หรือที่อยู่ปลายทางของแพ็กเก็ต

2.12.6 Load balancing

Neutron load balancing as a service เป็นส่วนขยายของบริการหรือที่เรียกว่า LBaaS ให้สามารถที่จะโหลดการจราจรให้สมดุลให้กับโปรแกรมที่ทำงานบนอินสแตนซ์เสมือนในคลาวด์ นิวตรอนมี API ในการจัดการ virtual IPs, pools, pool members และ health monitors

LBaaS ใช้ไดรเวอร์เพื่อโต้ตอบกับฮาร์ดแวร์และซอฟแวร์โหลดบาลานซ์ ไดรเวอร์ดีฟอลต์ใช้ haproxy ซึ่งเป็นโอเพ่นโอเพ่นซอร์สที่พร้อมใช้งานมากที่สุดของระบบปฏิบัติการ Unix-based

พื้นฐานของ Load balancing

มีสามองค์ประกอบสำคัญใน Load balancer ในนิวตรอนคือ:

- Pool member(s)
- Pool(s)
- Virtual IP(s)

Pool member เป็นออบเจกต์เลเยอร์4 แล้วยังเป็นส่วนประกอบของไอพีแอดเดรสของบริการและพอร์ตของการบริการ ยกตัวอย่าง pool member อาจจะเป็นเว็บเซิฟเวอร์ที่มีการกำหนดค่าไอพีแอดเดรส, 10.30.0.2, รอรับบนพอร์ต TCP 80

Pool เป็นกลุ่มของ Pool members ที่มีการให้บริการเนื้อหาเหมือนกัน มันประกอบไปด้วยเว็บเซิฟเวอร์ ยกตัวอย่าง อาจมีลักษณะการเป็นสมาชิกต่อไปนี้

- Server A: 10.30.0.2: 80
- Server B: 10.30.0.4: 80
- Server C: 10.30.0.6: 80

ไอพีเสมือน Virtual IP หรือ VIP คือไอพีแอดเดรสที่อยู่บน Load Balancer และรอรับการเชื่อมต่อที่จะเข้ามา Load Balancer จึงบาลานซ์การเชื่อมต่อของไคลเอ็นต์ระหว่างสมาชิกของ pool ที่เกี่ยวข้องกัน ไอพีเสมือนมักจะสัมผัสกับอินเทอร์เน็ตและมักจะถูกแมปไปยังชื่อโดเมน

Load balancing algorithms

อัลกอริทึมโหลดบาลานซ์ต่อไปนี้สามารถนำไปใช้กับ pool:

- Round robin
- Least connections
- Source IP

1) Round Robin

อัลกอริทึมแบบ Round Robin,โหลดบาลานเซอร์จะส่งผ่านแต่ละการเชื่อมต่อใหม่ไปยังเซิร์ฟเวอร์ต่อไปในสาย เมื่อเวลาผ่านไปการเชื่อมต่อทั้งหมดจะถูกกระจายได้อย่างทั่วถึงในทุกเครื่อง ด้วยตัวโหลดบาลานซ์ Round Robin เป็นอัลกอริทึมแบบ resource-intensive ที่ง่ายที่สุดและมีกลไกลตรวจสอบเมื่อเครื่องใช้งานโดยการเชื่อมต่อ เพื่อการใช้งาน pool member สมาชิกทั้งหมดควรจะมีความเท่าเทียมกันในแง่ของความเร็วในการประมวลผล ความเร็วการเชื่อมต่อ และหน่วยความจำ

2) Least-connections

อัลกอริทึมแบบ Least-connections โหลดบาลานเซอร์จะส่งผ่านการเชื่อมต่อใหม่ไปยังเซิร์ฟเวอร์ที่มีจำนวนของการเชื่อมต่อน้อยที่สุดในปัจจุบัน ถือว่าเป็นอัลกอริทึมแบบไดนามิกเนื่องจากระบบติดตามจำนวนการเชื่อมต่อที่แนบมากับแต่ละเซิร์ฟเวอร์และบาลานซ์การจราจรตามลำดับ Pool members ที่มีคุณสมบัติสูงกว่าจะมีโอกาสได้รับการจราจรมากขึ้น ทำให้พวกมันดำเนิการเชื่อมต่อได้เร็วขึ้น

3) Source IP

อัลกอริทึมแบบ Source IP การเชื่อมต่อทั้งหมดที่มาจาก source IP address เดียวกัน จะถูกส่งไปยัง Pool members เหมือนกัน การเชื่อมต่อมีความสมดุลในขั้นต้นโดยใช้อัลกอริทึมแบบ Round Robin และจะมีการติดตามในตารางสำหรับการค้นหาในอนาคตด้วยการเชื่อมต่อที่ตามมาจากที่อยู่ไอพี เดียวกันอัลกอริทึมนี้จะเป็นประโยชน์ในกรณีที่โปรแกรมต้องการไคลเอนต์เพื่อใช้

เซิร์ฟเวอร์เฉพาะสำหรับการร้องขอทั้งหมด เช่น ตะกร้าช้อปปิ้งออนไลน์ที่เก็บข้อมูลเซสชั่นบนเว็บเซิร์ฟเวอร์ท้องถิ่น

Monitoring

LBaaS สนับสนุนหลายประเภทการมอนิเตอร์รวมทั้ง TCP, HTTP และ HTTPS

- TCP monitor ตรวจสอบการเชื่อมต่อ pool members ที่เลเยอร์ 4
- HTTP และ HTTPS monitor ตรวจสอบความสมบูรณ์ของ pool members ที่อยู่บนพื้นฐานโค๊ดสถานะ HTTP เลเยอร์ 7

Session Persistence

LBaaS สนับสนุน session persistence บนไอพีเสมือน session persistence คือวิธีการโหลดบาลานซ์ที่บังคับให้ร้องขอของหลายไคลเอ็นต์ของโปรโตคอลเดียวกันจะถูกนำไปยังโหนดเดียวกัน คุณลักษณะนี้มักจะใช้กับการใช้งานเว็บจำนวนมากที่ไม่เปิดเผยสถานะโปรแกรมระหว่าง pool members

2.12.7 Firewalling

นิวตรอนมีสองวิธีการในการให้การรักษาความปลอดภัยเครือข่ายระดับอินสแตนซ์ วิธีแรกคือการใช้กลุ่มรักษาความปลอดภัยว่ากฎ leverage iptables เพื่อกรองการจราจรบนโหนดประมวลผลที่โฮสติ้งอินสแตนซ์ วิธีที่สองคือคุณลักษณะที่รู้จักกันเป็น Firewall-as-a-Service (FWaaS) ที่ให้การกรองที่ขอบด้านนอกของเครือข่ายในเราเตอร์นิวตรอน FWaaS ทำหน้าที่เป็นส่วนเติมเต็มให้กับกลุ่มการรักษาความปลอดภัยนิวตรอนไม่ใช่ทดแทน

Security groups in OpenStack

ก่อนที่จะมีนิวตรอน โนวา (Compute) บริการจัดการการรักษาความปลอดภัยของเครือข่ายการไปมาของอินสแตนซ์ผ่านการใช้งานของกลุ่มรักษาความปลอดภัย กลุ่มรักษาความปลอดภัยคือชุดของกฎการเข้าใช้เครือข่ายที่ จำกัด ประเภทของการจราจรอินสแตนซ์สามารถส่งหรือรับ นิวตรอนช่วย API ในการสร้าง, แก้ไข, นำไปใช้และลบกฎกลุ่มรักษาความปลอดภัย

เมื่อพอร์ตถูกสร้างขึ้นในนิวตรอน มันมีความเกี่ยวข้องกับกลุ่มรักษาความปลอดภัยเริ่มต้นเว้นแต่กลุ่มรักษาความปลอดภัยเฉพาะที่มีการระบุ กลุ่มรักษาความปลอดภัยเริ่มต้นดรอปการจราจรทางเข้าทั้งหมด และช่วยให้การจราจรทั้งหมดออกไปข้างนอกจากอินสแตนซ์ นอกจากนี้กฎระเบียบ

มาตรฐานจะถูกนำไปใช้กับทุกอินสแตนซ์ที่ห้าม IP, DHCP และการปลอมแปลงที่อยู่ MAC กฎสามารถเพิ่มไปยังกลุ่มรักษาความปลอดภัยเริ่มต้นเพื่อเปลี่ยนพฤติกรรมของมัน เมื่อกลุ่มรักษาความปลอดภัยที่ได้รับนำไปใช้กับพอร์ตนิวตรอน กฎกลุ่มรักษาความปลอดภัยที่เกี่ยวข้องจะถูกแปลโดยนิวตรอนเป็นกฎ iptables ที่ใช้แล้วไปยังโหนดประมวลผลตามลำดับโฮสติ้งอินสแตนซ์

Firewall -as -a-service service

FWaaS เป็นส่วนขยายสำหรับนิวตรอนที่ให้ผู้ใช้ที่มีความสามารถในการปรับใช้ไฟร์วอลล์ขอบด้านนอก เพื่อป้องกันเครือข่ายของตน ส่วนขยาย FWaaS ช่วยให้คุณสามารถทำสิ่งต่อไปนี้

- ใช้กฎไฟร์วอลล์กับการจราจรเข้าและออกจากเครือข่ายผู้เช่าที่ติดอยู่กับเราเตอร์นิวตรอน
- สร้างและแบ่งปันนโยบายไฟร์วอลล์ที่ถือคอลเลกชันรับคำสั่งของกฎไฟร์วอลล์
- ตรวจสอบกฎไฟร์วอลล์และนโยบาย
- ส่วนขยาย FWaaSเปิดตัวทรัพยากรเครือข่ายต่อไปนี้
- Firewall: แสดงให้เห็นทรัพยากรไฟร์วอลล์ลอจิกที่ผู้เช่าสามารถinstantiateและการบริหารจัดการไฟร์วอลล์จะเกี่ยวข้องกับนโยบายไฟร์วอลล์หนึ่ง
- Firewall policy: คือคอลเลกชันรับคำสั่งของกฎของไฟร์วอลล์ที่สามารถใช้ร่วมกันในผู้เช่า
- Firewall rule: แสดงให้เห็นคอลเลกชันของแอตทริบิวต์เช่นพอร์ตเลเยอร์ 4 และที่อยู่ IP ที่กำหนดเกณฑ์การจับคู่และการดำเนินการจะต้องดำเนินในการจับคู่ข้อมูลการจราจร

เช่นเดียวกับกฎระเบียบกลุ่มรักษาความปลอดภัยไฟร์วอลล์ในนิวตรอน ใช้ iptables เพื่อดำเนินการกรองการจราจร แทนที่จะรับการกำหนดค่าบนโหนดประมวลผลทุกครั้ง แต่กฎของไฟร์วอลล์จะดำเนินการโดยใช้ iptables ภายใน namespace ของเราเตอร์นิวตรอน การปรับปรุงในอนาคตอาจจะอนุญาตให้ใช้ไดรเวอร์ของบุคคลที่สามและปลั๊กอิน ที่ช่วยให้นิวตรอนในการโต้ตอบกับฮาร์ดแวร์อื่น ๆ หรือซอฟต์แวร์ไฟร์วอลล์

2.13 Openstack Ceilometer

โครงการ Ceilometer มีวัตถุประสงค์ที่จะทำ ระบบการเรียกเก็บเงินที่จะได้รับการวัดค่าใช้จ่ายทั้งหมดที่พวกเขาใช้และจะสร้างการเรียกเก็บเงินของลูกค้าในทุกองค์ประกอบหลัก OpenStack ในปัจจุบันที่มีการทำงานเตรียมการเพื่อรองรับส่วนประกอบ OpenStack ในอนาคต

วัตถุประสงค์ของโครงการและวิสัยทัศน์สำหรับมันคืออะไร?

- มีการเก็บรวบรวมข้อมูลที่ได้วัดค่าใช้งานของ CPU และเครือข่าย
- อนุญาตให้ผู้ติดตั้งระบบมีการทำงานร่วมกับ Meter โดยตรงหรือโดยการแทนองค์ประกอบอื่นใน Openstack
- ข้อมูลอาจถูกเก็บรวบรวมโดยมาจากการแจ้งเตือนของการการตรวจสอบที่ส่งมาจากการสำรวจ(polling)โครงสร้างพื้นฐาน
- อนุญาตให้ผู้ติดตั้งระบบมีการกำหนดค่าประเภทของข้อมูลที่จะถูกเก็บรวบรวมมาเพื่อตอบสนองความต้องการของพวกเขาในการดำเนินงาน
- ข้อมูลที่เก็บรวบรวมโดยระบบ Meter จะใช้ ได้กับผู้ใช้ผ่าน REST API

2.17.1 Ceilometer Concepts

Ceilometer (Telemetry Service) ทำหน้าที่วัดการใช้งาน resources ไม่ว่าจะเป็น CPU, Memory, Storage และ Network หรือสถิติการใช้บริการต่างๆเช่น ค่าบริการ Ceilometer มีกลุ่มของแนวคิดพื้นฐานคือ

Resources

Resources แสดงถึงสิ่งที่จะถูกวัดหรือการตรวจสอบ พวกนี้มักสามารถทำโดยรูปแบบของการอินสแตนซ์ OpenStack หรือ volumes อย่างไรก็ตาม นอกจากนี้ยังมีResources ประกอบด้วยโครงสร้างเหมือนโครงการและเครือข่าย

Meters

Meters เป็นการวัดที่เกิดขึ้นจริงของ Ceilometer สนับสนุนหลายมิเตอร์จากทรัพยากรที่แตกต่างกัน นอกจากนี้ในแต่ละมิเตอร์เหล่านี้จะรายงานข้อมูลของพวกเขากลับมาแบบ one of several กับหน่วยของตัวเอง ตัวอย่างเช่น "ซีพียู" จะมีการวัดหน่วยวัดระยะเวลาสะสม CPU ที่ใช้ในหน่วยของ "ns" หรือนาโนวินาทีของเวลา มิเตอร์ Ceilometer สามารถใช้ตามชนิดดังต่อไปนี้

- Cumulative คือการเพิ่มมากขึ้นเมื่อเวลาผ่านไปเหมือนการใช้ซีพียู

- Gauges แสดงค่าแน่นอนในเวลาที่มีการวัด เช่น ดิสก์ I / O หรือการอัพโหลดภาพ
- Deltas แสดงความแตกต่างระหว่างการวัดในปัจจุบันและการวัดที่ผ่านมา

Ceilometer ดำเนินการของมิเตอร์จำนวนมาก มิเตอร์เหล่านี้จะถูกแบ่งออกเป็นหลายประเภท ประเภทเหล่านี้กับตัวอย่างการวัดจะอยู่ด้านล่าง

Meter	Examples
Compute	Disk I/O requests, vCPUs, Instance (also reported by flavor)
Network	Network creation, Floating IP duration
Image	Uploaded image size, Images,
Volume	Volumes, Volume size
Object Storage	Total size of stored objects, Number of API requests
Energy	Power consumption, Energy
Orchestration	Creation, deletion and update requests for stacks

ตารางที่ 2.14 Meter ของ Ceilometer

ขณะที่แต่ละมิเตอร์พึ่งพาเทคโนโลยีพื้นฐานในการค้นหาสำหรับการเก็บรวบรวมข้อมูล มิเตอร์ไม่ทั้งหมดที่พร้อมสำหรับทุกผู้ให้บริการ OpenStack (storage, virtualization, network เป็นต้น)

Samples

Samples คือการตรวจวัดที่ไม่ต่อเนื่องที่มาจากเครื่องวัดบนทรัพยากร ตัวอย่างเช่น Samples อาจจะมีจำนวนของอินสแตนซ์ที่วัดในโปรเจคโดยเฉพาะใน OpenStack สำหรับห้านาทีแรกของวันที่เฉพาะเจาะจง ที่บันทึกในฐานข้อมูล Ceilometer ที่สามารถเรียกดูหรือจัดกลุ่มกับ Sampleอื่น ๆ เพื่อสร้างรายงานการใช้งาน

Alarms

Alarms เป็นคุณลักษณะใหม่ใน Ceilometer ที่มีรูปแบบพื้นฐานของการสนับสนุนการตรวจสอบ Ceilometer และมีการบูรณาการ OpenStack Heat พวกมันจะมีเกณฑ์โดยเฉพาะในเครื่องวัดและทรัพยากรที่จะสร้างเหตุการณ์เมื่อพวกเขาเกินกว่าที่กำหนด ตัวอย่างเช่น Alarms สามารถตั้งค่าในการใช้งาน CPU ของอินสแตนส์ที่มีเกณฑ์ 75% ในตัวอย่างที่ซับซ้อนมาก Alarms ยังสามารถกำหนดให้มีการประเมินเทียบกับช่วงเวลาที่แน่นอนใช้เปรียบเทียบมากขึ้น (เช่นน้อยกว่าหรือเท่ากับ) หรือแม้กระทั่งขึ้นอยู่กับสถานะAlarmอื่น ๆ

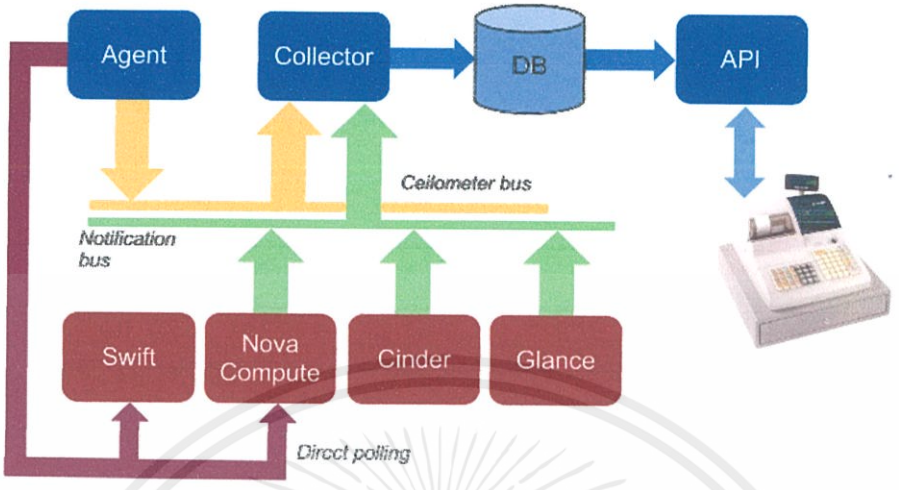
Statistics

Statistics เป็น Samples ที่รวบรวมจัดกลุ่มตามมิเตอร์ในช่วงระยะเวลาที่ระบุ ตัวอย่างเช่น Statistics อาจจะเป็น vCPUs ที่ใช้โดยผู้เช่าที่ระบุในวันที่ระบุ

2.17.2 Ceilometer Architecture

OpenStack Telemetry มีสถาปัตยกรรมแบบกระจายที่แผ่กระจายในรูปแบบเซิร์ฟเวอร์กลางCeilometer ให้กับแต่ละโหนดของ Nova compute

แตกต่างจากส่วนอื่น ๆ OpenStack คือCeilometer ไม่ได้มีหน้าจอแดชบอร์ดโดยเฉพาะ



รูปที่ 2.60 รูปการทำงานของ Ceilometer

Ceilometer API

กระบวนการ Ceilometer API ยอมรับการร้องขอ API และส่งกลับข้อมูลจากแหล่งเก็บข้อมูล มันมีลอจิกหลักสำหรับ Ceilometer อย่างไรก็ตามกระบวนการ APi ไม่ยอมรับหรือดำเนินการใด ๆ ของข้อมูลการวัดที่เกิดขึ้น ข้อมูลที่ถูกป้อนเข้าแหล่งเก็บข้อมูลผ่านทาง collection agents

Collection Agents

Ceilometer ขึ้นอยู่กับagentในการเก็บรวบรวมข้อมูลจากโหนด OpenStack agentเหล่านี้ทำงานบน Compute nodes เพื่อรวบรวมข้อมูล VM จากไฮเปอร์ไวเซอร์พื้นฐาน นอกจากนี้ Ceilometer ยังรับคิวข้อความที่ใช้โดยแต่ละบริการในการตรวจสอบเหตุการณ์

collection agents ของ Ceilometer ได้แก่:

- ceilometer-agent-central
- ceilometer-agent-compute
- ceilometer-collector

Alarm Daemons

ขณะที่ Ceilometer ใช้ชุดประมวลผลกับข้อมูลการเก็บรวบรวมข้อมูลการวัด มันยังใช้ชุดของกระบวนการในการอำนวยความสะดวกในการแจ้งเตือน มีสองส่วนที่จะเตือนใน Ceilometer คือ ผู้ประเมินและผู้แจ้ง ผู้ประเมิน (ceilometer-alarm-evaluator) ตรวจสอบรายชื่อของการเตือนภัยที่กำหนดไว้เพื่อดูว่ามีอันไหนของการเตือนภัยที่ได้รับการพบหากการเตือนภัยได้รับการพบ ผู้แจ้ง (*ceilometer-alarm-notifier*) จะถูกเรียกดำเนินการให้มีการกระทำที่เหมาะสม

Data Store

Ceilometer เก็บข้อมูลที่เก็บรวบรวมของการวัดเหตุการณ์,และการกำหนดค่าของตัวเองใน แหล่งข้อมูล ที่มันสนับสนุนกลุ่มของ SQL และ NoSQL แบ็กเอนด์สำหรับการจัดเก็บข้อมูล ปัจจุบัน ฐานข้อมูลสนับสนุนมี MongoDB, HBase, MySQL (หรือ sqlalchemy-enabled databases) และ DB2

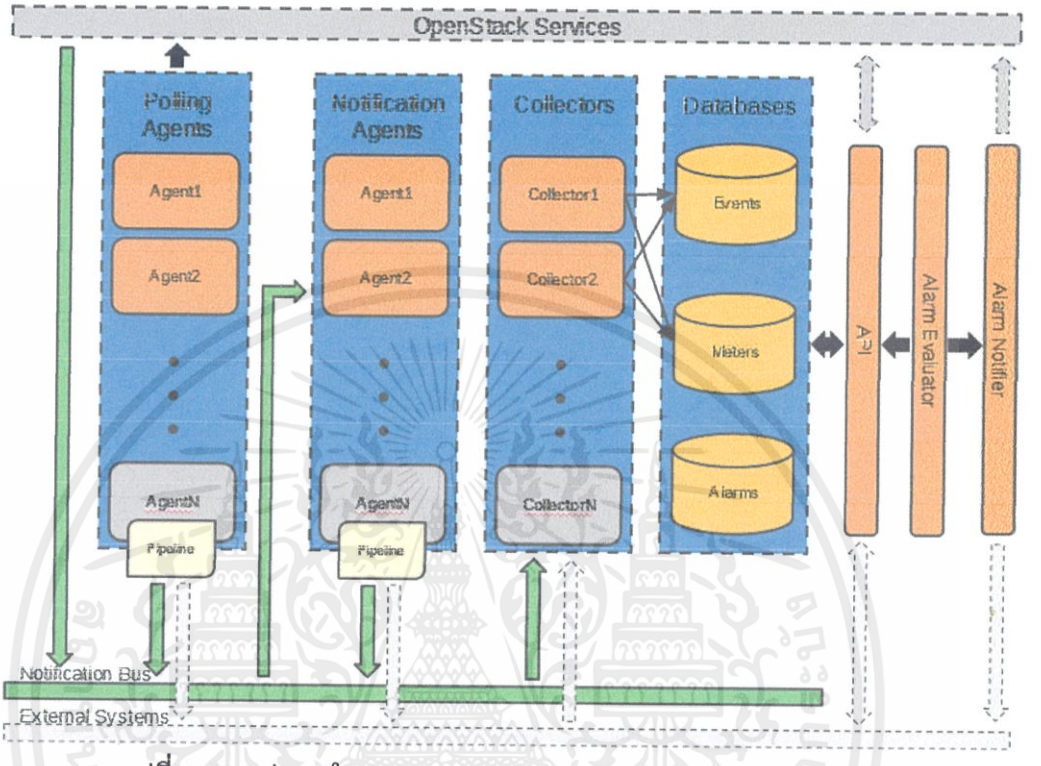
Data Store	Notes
MongoDB	The default data store for Ceilometer and the most fully tested option
MySQL	Lacks complete support for alarming features
HBase	
DB2	

ตารางที่ 2.15 Data Store ของ Ceilometer

Queue

ในขณะที่ Ceilometerเองไม่ได้จำเป็นต้องมีคิวการส่งข้อความของตัวเองในการดำเนินงาน มันจะใช้คิวข้อความของการให้บริการ OpenStack อื่น ๆ ที่จะตรวจพบและบันทึกเหตุการณ์

High-Level Architecture



รูปที่ 2.61 รูปการทำงานของ High-Level Architecture

แต่ละบริการ Ceilometer ถูกออกแบบมาเพื่อปรับขนาดให้กว้างขึ้นได้ แรงงานเพิ่มเติมและโหนดสามารถเพิ่มขึ้นอยู่กับปริมาณโหลด Ceilometer มี5บริการหลัก โดยข้อมูล agents ที่ออกแบบมาเพื่อทำงานเป็นอิสระจากการ collection และ alarming แต่ยังออกแบบมาเพื่อทำงานร่วมกันเป็นโซลูชั่นที่สมบูรณ์แบบ:

polling agent คือ daemonออกแบบมาเพื่อสำรวจบริการ OpenStack และสร้าง Meters

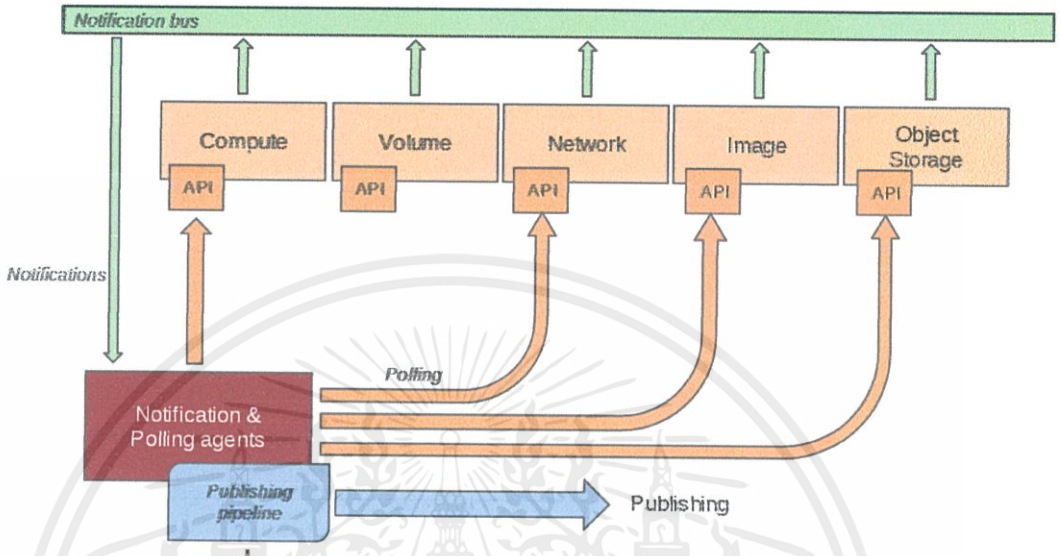
notification agent คือ daemonออกแบบมาเพื่อรับฟังการแจ้งเตือนในคิวข้อความและแปลงให้เป็นเหตุการณ์และSample

collector คือ daemonออกแบบมาเพื่อรวบรวมและบันทึกเหตุการณ์และข้อมูลการวัดที่สร้างขึ้นโดย notification agent และ polling agents

api คือ บริการเพื่อสอบถามและดูข้อมูลที่บันทึกไว้โดย collector service

alarming คือ daemonที่มีการประเมินและแจ้งขึ้นอยู่กับการกำหนด alarming rules

การรวบรวมข้อมูล (Gathering the data)



รูปที่ 2.62 รูปการทำงานของ Gathering the data

วิธีการเก็บรวบรวมข้อมูล

Ceilometer มี 2 วิธีในการเก็บรวบรวมข้อมูล

- Bus listener agent ซึ่งจะนำeventที่สร้างบน notification bus และแปลงพวกเขาเป็น Ceilometer samplesนี่คือวิธีการที่ต้องการของการเก็บรวบรวมข้อมูล
- Polling agents ซึ่งเป็นวิธีการที่ใช้น้อย จะสำรวจบาง API หรือเครื่องมืออื่น ๆ เพื่อใช้ในการเก็บรวบรวมข้อมูลในช่วงเวลาปกติ ถ้าตัวเลือกที่มีอยู่เพื่อรวบรวมข้อมูลเดียวกันโดยแจ้งเตือนการบริโภคแล้ววิธีการเลือกตั้งเป็นที่ต้องการน้อยลงเนื่องจากภาระที่จะสามารถกำหนดเกี่ยวกับการให้บริการ API

Notification Agents คอยรับข้อมูล

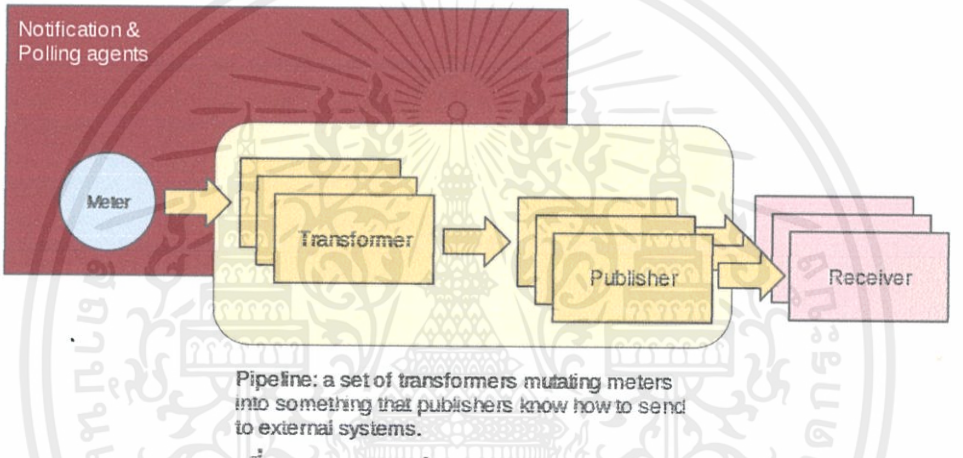
notification daemon (agent-notification) ซึ่งตรวจสอบบัสข้อความสำหรับข้อมูลที่ถูกจัดไว้ให้โดย ส่วนประกอบOpenStack อื่น ๆ เช่น Nova, Glance, Cinder, Neutron, Swift, Keystone, และ Heat

Polling Agents ถามข้อมูล

การ Polling สำหรับทรัพยากรcomputeจะถูกจัดการโดย polling agent ที่ทำงานบนโหนด compute (สื่อสารกับไฮเปอร์ไวเซอร์มีประสิทธิภาพมากขึ้น) ส่วนมากจะหมายถึง compute-agent การ Polling ผ่านทาง API service สำหรับทรัพยากรที่ไม่ใช่ compute จะถูกจัดการโดย agent ที่ทำงานบน cloud controller node มักจะเรียกว่า central-agent

การประมวลผลข้อมูล (Processing the data)

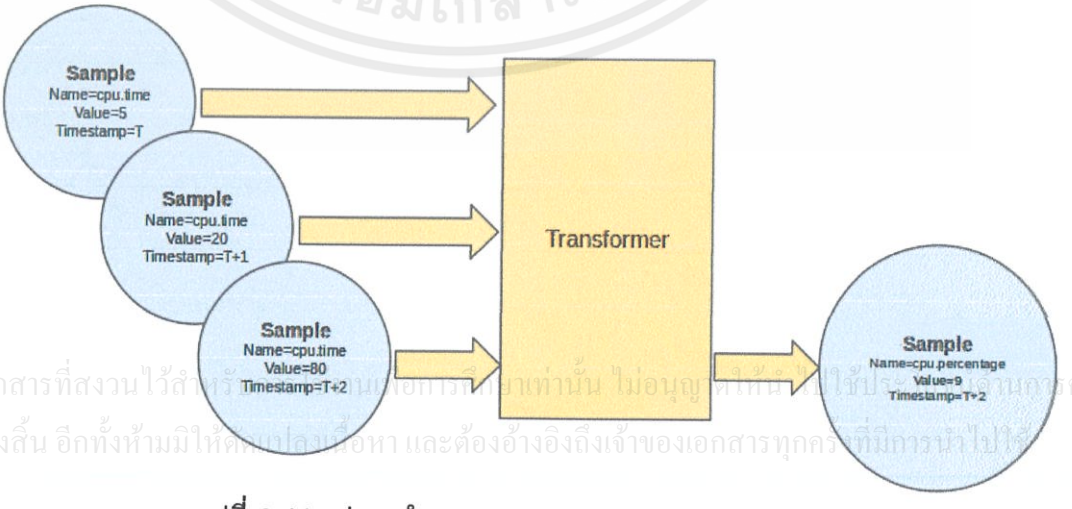
Pipeline Manager



รูปที่ 2.63 รูปการทำงานของ Pipeline Manager

Ceilometer มีความสามารถที่จะนำข้อมูลที่รวบรวมโดยagentจัดการมันและ publish ผ่านทาง multiple pipelines

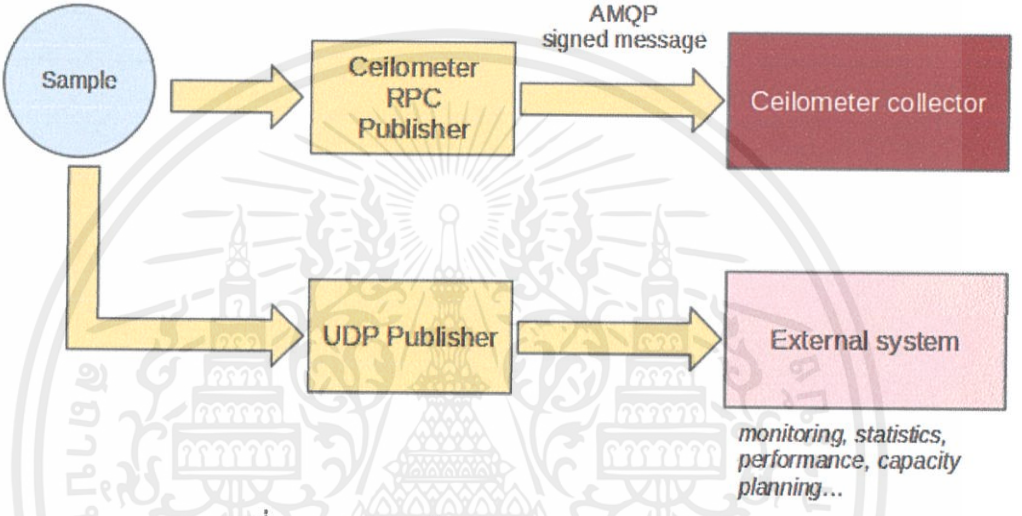
เปลี่ยนข้อมูล(Transforming the data)



รูปที่ 2.64 รูปการทำงานของ Transforming data

ตัวอย่างของการรวมตัวของกลุ่มตัวอย่างใช้เวลา CPU หลายตัวในอัตราร้อยละของซีพียูแต่ละตัว ข้อมูลที่รวบรวมจาก polling agents และ notifications agents มีมากและถ้ารวมกับที่ผ่านมาหรือtemporal context สามารถนำมาใช้เพื่อให้ได้ข้อมูลมากยิ่งขึ้น Ceilometer มี transformers ต่างๆที่สามารถใช้ในการจัดการกับข้อมูลในpipeline

การเผยแพร่ข้อมูล (Publishing the data)



รูปที่ 2.65 รูปการทำงานของ Publishing data

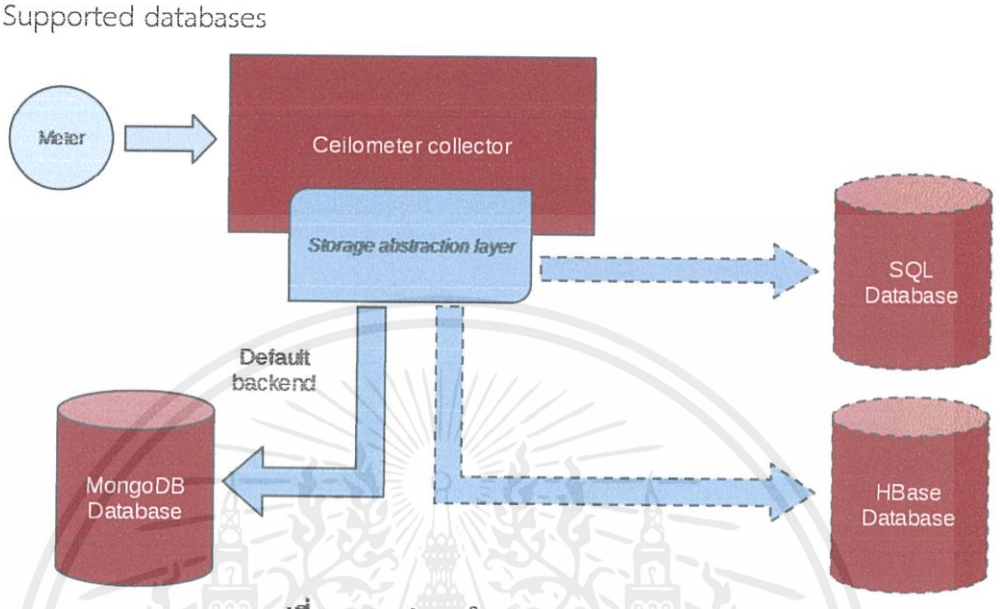
รูปนี้แสดงให้เห็นว่ากลุ่มตัวอย่างที่สามารถนำไปเผยแพร่ไปยังหลายๆปลายทาง ปัจจุบันการประมวลผลข้อมูลที่สามารถนำไปเผยแพร่โดยใช้ การส่ง 4 แบบที่แตกต่างกัน:

1) Notifier การแจ้งเตือนจะขึ้นอยู่กับ publisher ที่ผลักดันให้กลุ่มตัวอย่างเพื่อจะคิวข้อความที่สามารถถูกนำมาใช้โดย collector หรือ external system
2) Rpc เป็นวิธีที่น่าเชื่อถือโดย synchronous RPC ขึ้นอยู่กับ publisher
3) Udp ซึ่ง publishes ตัวอย่างโดยใช้แพ็คเก็ต UDP
4) Kafka ซึ่ง publishes ข้อมูลไปยังคิวข้อความของ Kafka ที่ถูกใช้โดยระบบใดๆ ที่สนับสนุนKafka

การจัดเก็บข้อมูล (Storing the data)

Collector Service

Collector daemon รวบรวม processed event และ metering data ที่บันทึกโดย notification agents และ polling agents และตรวจสอบข้อมูลที่เข้ามาและ (ถ้าลายเซ็นที่ถูกต้อง) จากนั้นเขียน messages ไปยังเป้าหมายที่ประกาศ เช่น ฐานข้อมูล, ไฟล์ หรือ http



รูปที่ 2.66 รูปการทำงานของ Database

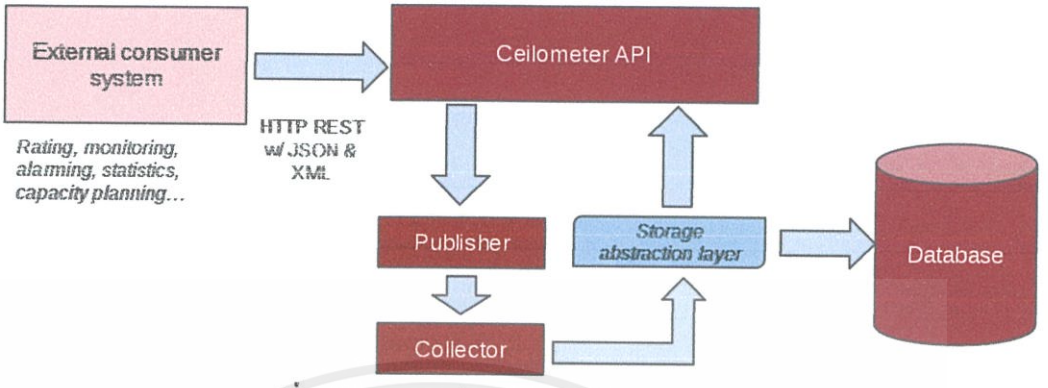
ภาพรวมของรูปแบบการจัดเก็บข้อมูลของ Ceilometer

ตั้งแต่เริ่มต้นของโครงการนี้มีรูปแบบให้สามารถอนุญาตให้วางในที่ที่มีฐานข้อมูลแบ็กเอ็นที่มีหลายชนิดที่จะใช้ได้

การเข้าถึงข้อมูล (Accessing the data)

API Service

ข้อมูลที่รวบรวมจากpolling agents และ notification agents จะถูกเก็บไว้ในฐานข้อมูลที่ได้รับการสนับสนุน ก็เป็นไปได้ว่า schema ของฐานข้อมูลเหล่านี้อาจมีวิวัฒนาการอยู่ตลอดเวลา ด้วยเหตุผลนี้เราขอเสนอ REST API และขอแนะนำให้คุณเข้าถึงข้อมูลที่เก็บรวบรวมผ่านทาง API มากกว่าโดยการเข้าถึงฐานข้อมูลต้นแบบโดยตรง



รูปที่ 2.67 รูปการทำงานของ API service

จากรูปจะแสดงให้เห็นวิธีการเข้าถึงข้อมูลที่เก็บไว้ของ Ceilometer

การประเมินข้อมูล (Evaluating the data)

Alarming Service

Alarming เป็นส่วนประกอบหนึ่งของ Ceilometer โดยในรุ่นHavana คุณสามารถตั้งค่า alarm ให้ขึ้นอยู่กับเกณฑ์การประเมินในการเก็บรวบรวมของกลุ่มตัวอย่าง Alarm สามารถที่จะตั้งค่าบน Meterเดียว หรือหลายๆอันก็ได้ ตัวอย่างเช่นคุณอาจต้องการเรียกเตือนเมื่อใช้หน่วยความจำถึง 70% บนอินสแตนซ์ที่ได้กำหนดไว้ แต่ถ้าอินสแตนซ์มีการใช้เกิน 10 นาที การติดตั้ง Alarming จะต้องเรียก API server ของ Ceilometer ที่ระบุเงื่อนไขของ alarm และการกระทำที่จะใช้

บทที่ 3
การวิเคราะห์และการออกแบบระบบ

3.1 การจัดเตรียมโครงสร้างพื้นฐานทางกายภาพ (Preparing the physical infrastructure)

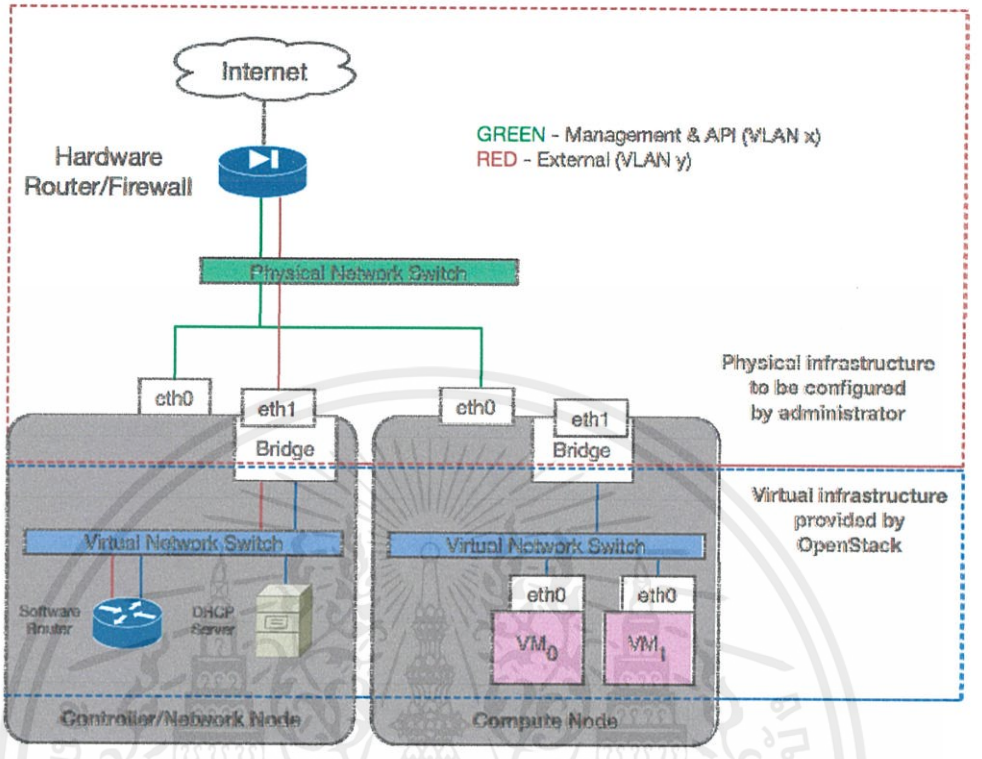
การออกแบบเครือข่ายเป็นสิ่งสำคัญก่อนที่จะกำหนดวัตถุประสงค์ของคลาวด์ โดยมีเป้าหมาย เป้าหมายในการสร้างสภาพแวดล้อมที่สามารถปรับขนาดได้สูงสำหรับหลายระดับความซ้ำซ้อนทาง เครือข่าย หรือมีเป้าหมายที่จะให้ความยืดหยุ่นของเครือข่ายหรือแพลตฟอร์มประมวลผล

OpenStack เครือข่ายสามารถให้บริการหลายบทบาทภายในคลาวด์ที่แตกต่างกัน โดย วัตถุประสงค์คือต้องการความปลอดภัย และรองรับฮาร์ดแวร์ต่างๆ

www.openstack.org ให้สถาปัตยกรรมอ้างอิงสำหรับคลาวด์นิวตรอนตามที่ที่เกี่ยวข้องกับ การรวมกันของโหนดต่อไปนี้:

- Controller Node
- Network Node
- Compute Node

ก่อนที่จะมีการติดตั้ง OpenStack โครงสร้างพื้นฐานของเครือข่ายทางกายภาพจะต้อง กำหนดค่าให้การสนับสนุนเครือข่ายที่จำเป็นสำหรับการดำเนินงานคลาวด์ ในแผนภาพต่อไปนี้ ได้เน้น พื้นที่ของความรับผิดชอบในการดูแลระบบเครือข่าย:



รูปที่ 3.1 ภาพแสดงโครงสร้างพื้นฐานของเครือข่ายทางกายภาพ

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

ชนิดของการจราจรเครือข่าย (Type of network traffic)

สถาปัตยกรรมอ้างอิงสำหรับ OpenStack เครือข่ายจะกำหนดอย่างน้อยสี่ประเภทที่แตกต่างกันของเครือข่าย

- Management
- API
- External
- Guest

1) Management Network

เครือข่ายการจัดการ (Management Network) ใช้สำหรับการสื่อสารภายในระหว่างโฮสต์สำหรับการให้บริการ 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

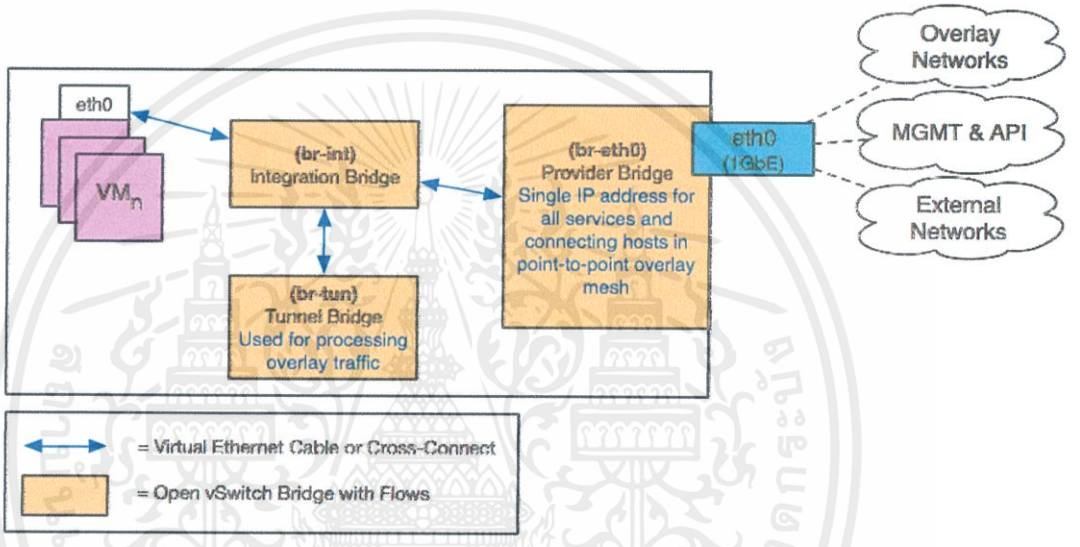
เครือข่ายเกสต์เป็นเครือข่ายเฉพาะเพื่อการจราจรอินสแตนซ์ ตัวเลือกสำหรับเครือข่ายเกสต์ไปรวมถึงเครือข่ายท้องถิ่นที่ถูกจำกัดไปยังโหนดเฉพาะ การใช้เครือข่ายซ้อนทับเสมือนกระทำด้วยการห่อหุ้ม GRE หรือ VXLAN

3.2 การเชื่อมต่อเซิฟเวอร์ทางกายภาพ (Physical server connections)

จำนวนของอินเตอร์เฟซที่จำเป็นต่อโฮสต์จะขึ้นอยู่กับชนิดของคลาวด์ถูกสร้างขึ้น การรักษาความปลอดภัย และความต้องการประสิทธิภาพการทำงานขององค์กร

3.2.1 interface

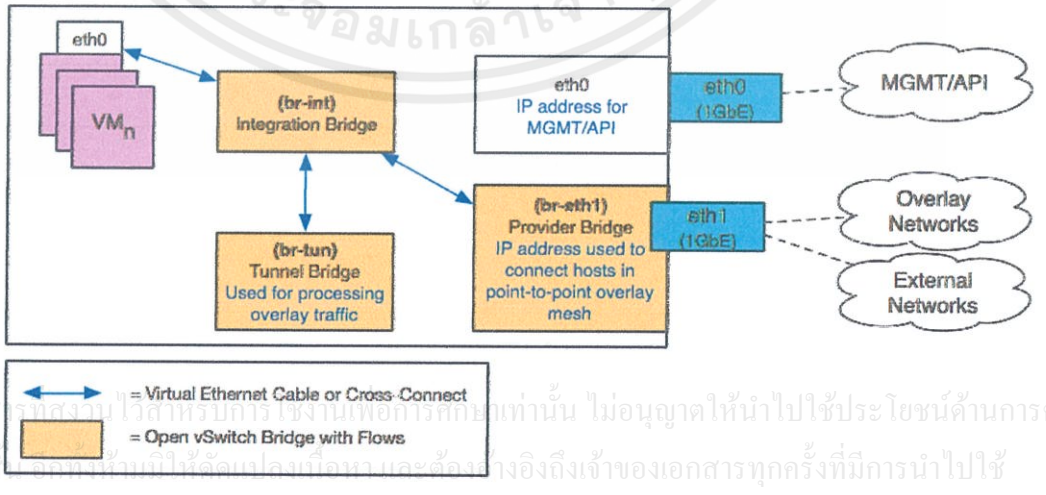
อินเตอร์เฟซเดียว (single interface)



รูปที่ 3.2 ภาพแสดงโครงสร้าง single interface

ในแผนภาพนี้บริการ OpenStack ทั้งหมดและการจราจรการจัดการจะติดต่อทางกายภาพเช่นเดียวกับการจราจรเกสต์

อินเตอร์เฟสแบบหลายอินเตอร์เฟส (Multiple interfaces)



รูปที่ 3.3 ภาพแสดงโครงสร้าง Multiple interfaces

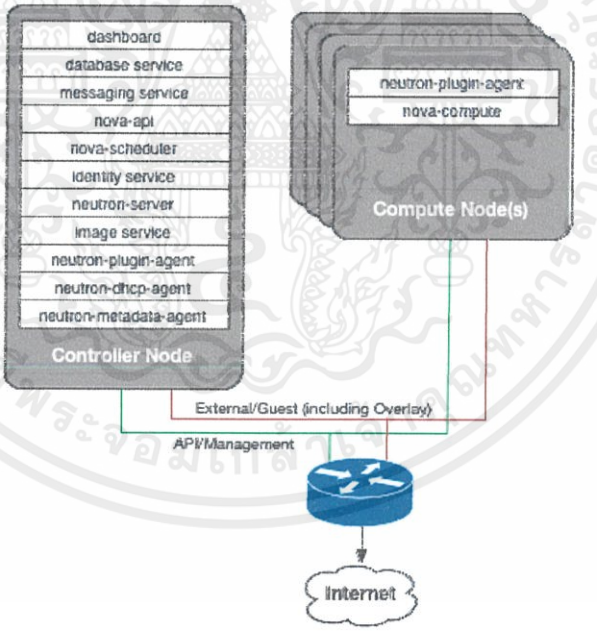
ในแผนรูปที่ 3.3 แสดงการติดต่อทางกายภาพโดยเฉพาะจัดการกับจราจรกำกับการไปมาของอินสแตนซ์หรือบริการอื่น ๆ ของเครือข่าย OpenStack เช่น LBaaS และ FWaaS ขณะที่อินเตอร์เฟสอื่นจัดการกับ OpenStack API และการจัดการจราจร

3.2.2 Node

A single controller with one or more compute nodes

ในสภาพแวดล้อมที่ประกอบด้วยคอนโทรลเลอร์เดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลคอนโทรลเลอร์อาจจะจัดการกับทุกบริการเครือข่ายและการบริการอื่น ๆ ของเครือข่าย OpenStack ในขณะที่โหนดประมวลผลให้ทรัพยากรการประมวลผล

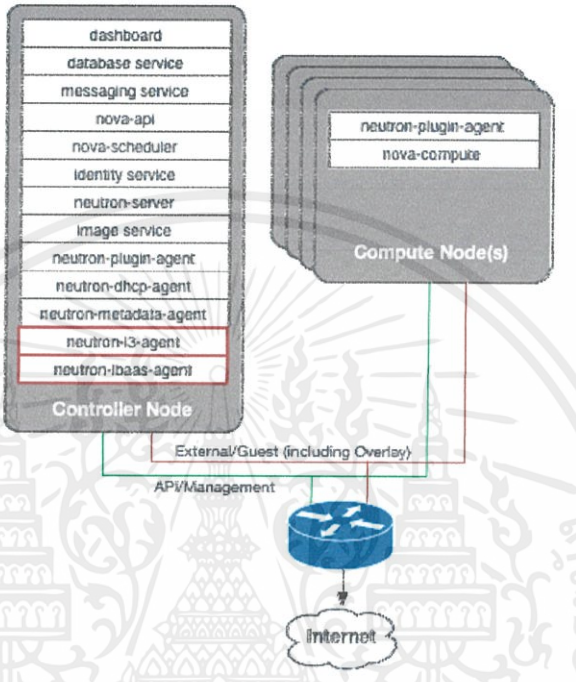
แผนภาพต่อไปนี้โหนดประมวลผลจะจัดการ OpenStack management และ networking services โดยไม่ใช้ layer 3 agent แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes



รูปที่ 3.4 ภาพแสดงโครงสร้างที่มี Controller Node ต่อกับ หลายๆCompute Node

รูปที่ 3.4 สะท้อนให้เห็นถึงการใช้ควบคุมเดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลที่นิวตรอนให้เพียงเลเยอร์ 2 เชื่อมต่อกับอินสแตนซ์ เราเตอร์ภายนอกเป็นสิ่งจำเป็นในการจัดการเส้นทางระหว่างกลุ่มเครือข่าย

รูปที่ 3.5 โหนดประมวลผลจะจัดการ OpenStack management และ networking services โดยเพิ่ม layer 3 agent แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes



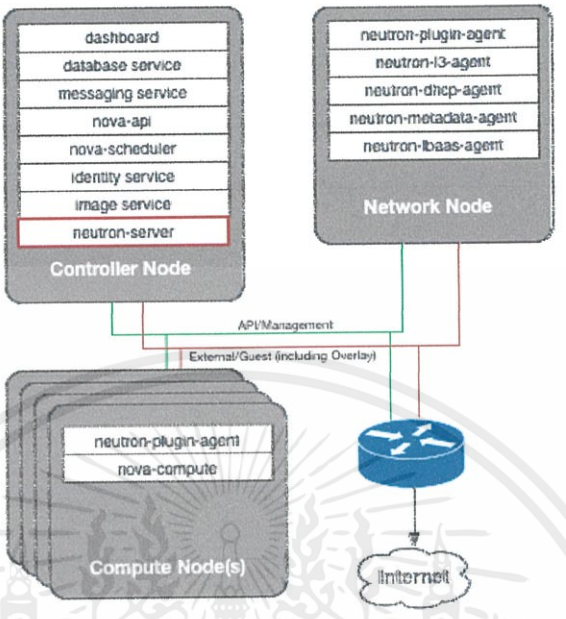
รูปที่ 3.5 ภาพแสดงโครงสร้างที่มี Controller Node ต่อกับ หลายๆCompute Node

รูปที่ 3.5 สะท้อนให้เห็นถึงการใช้คอนโทรลเลอร์โหนดเดียวและหนึ่งหรือมากกว่าหนึ่งโหนดประมวลผลในการกำหนดค่าเครือข่ายที่ใช้ Neutron L3 agent ส่วนเราเตอร์ซอฟแวร์ที่สร้างขึ้นด้วยนิวตรอนอยู่ในโหนดคอนโทรลเลอร์และจัดการเส้นทางระหว่างเครือข่ายที่เชื่อมต่อผู้เช่า

A single controller plus network node with one or more compute nodes

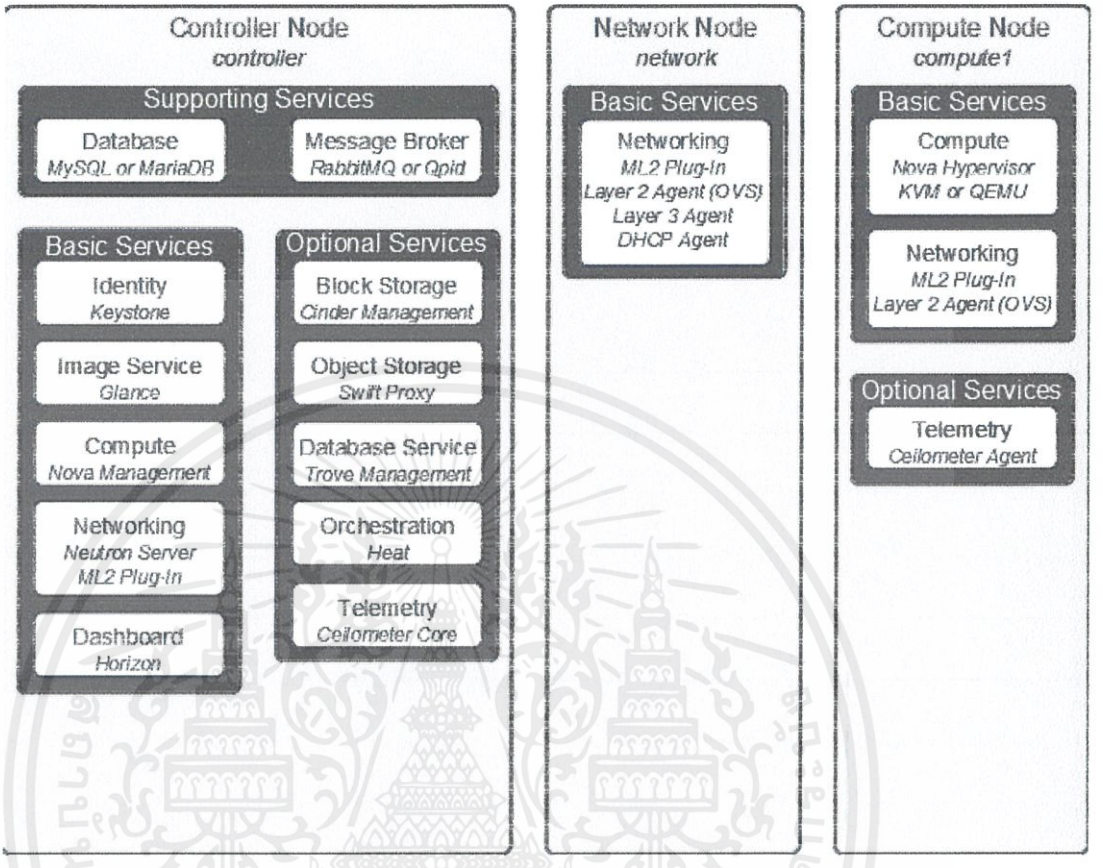
โหนดเครือข่ายเป็นสิ่งหนึ่งที่มีความมุ่งมั่นที่จะจัดการส่วนใหญ่หรือทั้งหมด ของเครือข่าย OpenStack บริการเครือข่ายรวมทั้ง L3 agent, DHCP agent, metadata agent และอื่นๆ การใช้โหนดเครือข่ายเฉพาะให้การรักษาความปลอดภัยและความยืดหยุ่นเพิ่มขึ้น

รูปที่ 3.6 แสดงให้เห็นถึงโหนดเครือข่ายบริการโฮสติ้งเครือข่าย ของเครือข่ายOpenStack ทั้งหมดรวมทั้ง Neutron L3 agent นิวตรอน API มีการติดตั้งบนโหนดคอนโทรลเลอร์ แล้วได้นำสองการเชื่อมต่อทางกายภาพถูกนำมาใช้เพื่อให้แยกระหว่าง control planes และ data planes



รูปที่ 3.6 ภาพแสดงโครงสร้างที่มี Controller Node ที่มี Network Node ต่อกับหลายๆCompute Node หรือตัวเดียว

รูปที่ 3.6 สะท้อนให้เห็นถึงการใช้โหนดเครือข่ายเฉพาะในการกำหนดค่าเครือข่ายที่ใช้ Neutron L3 agent ส่วนเราเตอร์ซอฟแวร์ที่สร้างขึ้นด้วยนิวตรอนอยู่ในโหนดเครือข่าย และการจัดการเส้นทางระหว่างเครือข่ายที่เชื่อมต่อผู้เช่าบริการ API นิวตรอนเซิร์ฟเวอร์ยังคงอยู่ในโหนดคอนโทรลเลอร์



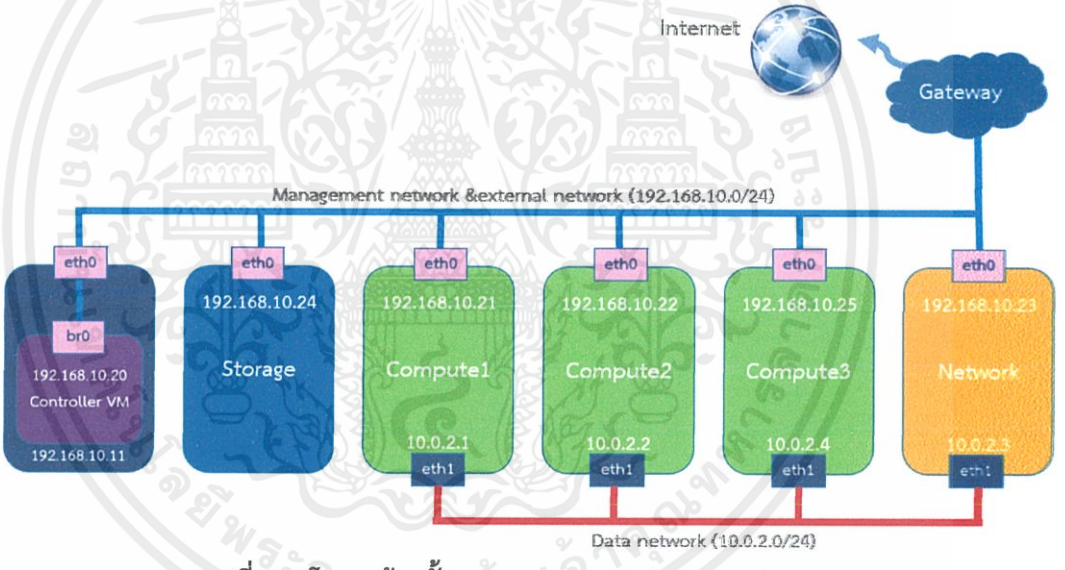
รูปที่ 3.7 ส่วนประกอบของแต่ละ Node

- **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 ที่ทำงานเครือข่ายผู้เช่าและดำเนินการตามกลุ่มรักษาความปลอดภัย สามารถเลือกให้ Compute node ทำงานเป็นเป็น Telemetry agent. ส่วนประกอบเหล่านี้มีคุณสมบัติเพิ่มเติมสำหรับสภาพแวดล้อมของคุณ

3.3 Openstack Cloud Network

3.3.1 ครื่อข่ายทางกายภาพ (Physical network)



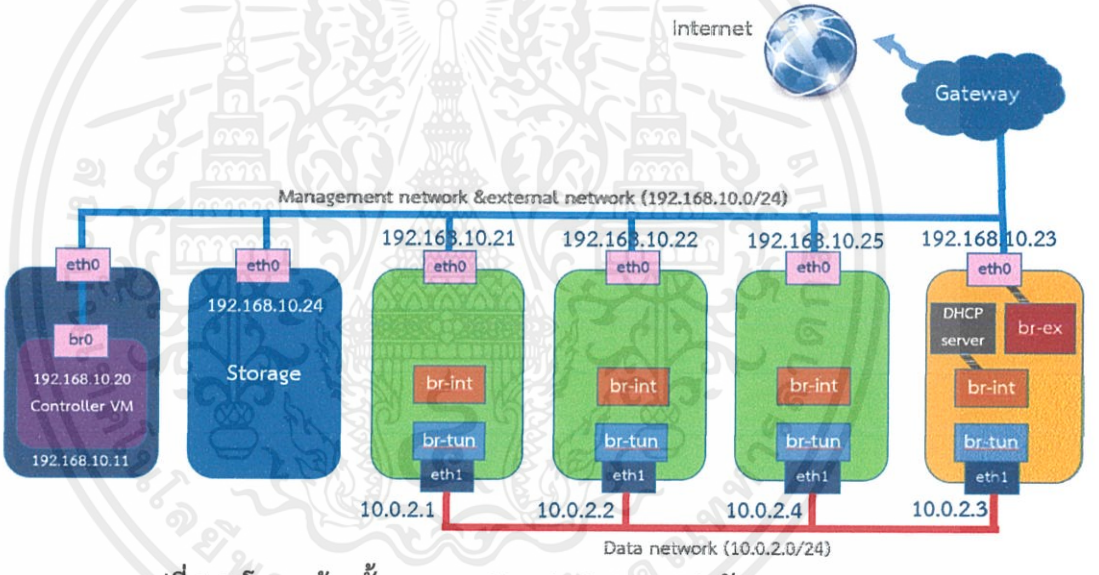
รูปที่ 3.8 โครงสร้างทั้งหมดของPhysical network

จากรูปที่ 3.8 เป็นเครือข่ายทางกายภาพที่ใช้ในการทดลอง

- ประกอบด้วย Controller 1 node, Storage 1 node, Compute 3 nodes และ Network 1 node
- 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 (192.168.10.24) มี 1 interface เชื่อมต่อเข้ากับ Management network

- Compute มี 2 interfaces โดย interface แรก (192.168.10.21 , 192.168.10.22 , 192.168.10.25) เพื่อเชื่อมต่อ Management network ส่วนอีก interface (10.0.2.1 , 10.0.2.2 10.0.2.4) เพื่อเชื่อมต่อกับ Data network
- Network มี 2 interfaces โดย interface แรก (192.168.10.23) เพื่อเชื่อมต่อ Management network และ รับส่งข้อมูลจากอินเตอร์เน็ตส่วนอีก interface (10.0.2.3) เพื่อเชื่อมต่อกับ Data network เพื่อรับส่งข้อมูลจาก Compute

3.3.2) Neutron network with VXLAN

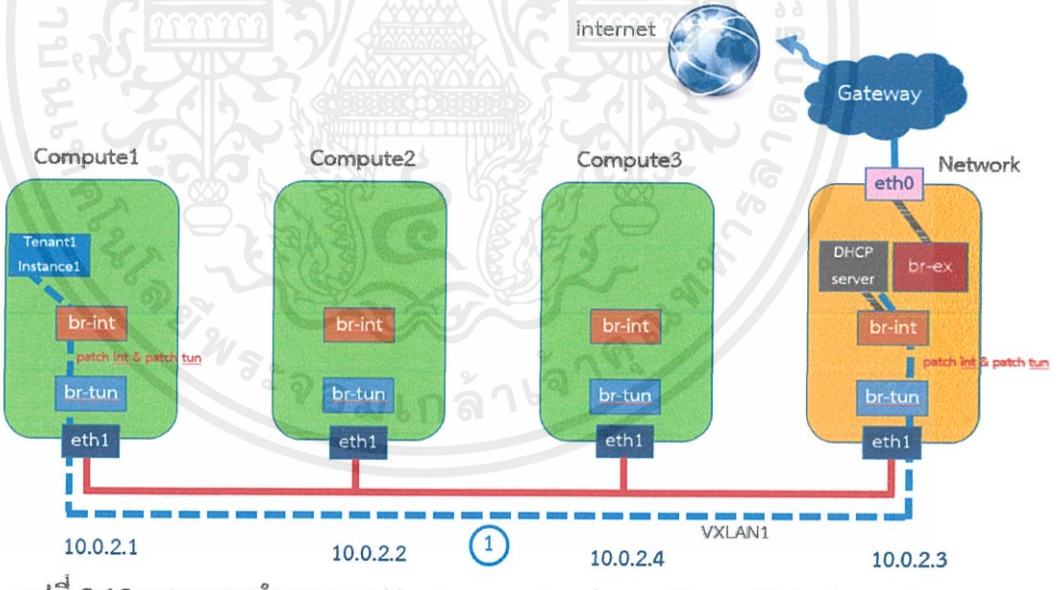


รูปที่ 3.9 โครงสร้างทั้งหมดของPhysical network กับ VXLAN

จากรูปที่ 3.9 เป็น Neutron network โดยมีรายละเอียดดังนี้

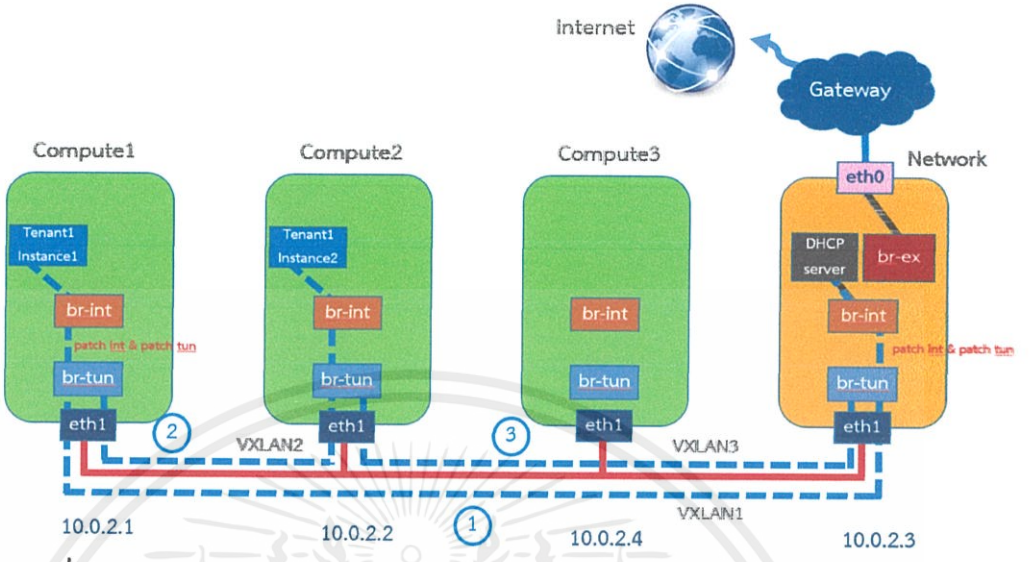
- Compute จะประกอบไปด้วย br-int และ br-tun
- Integration bridge (br-int) จะทำการ tagging และ un-tagging กับ VXLAN สำหรับแพ็คเก็ตที่เข้ามาและออกไป โดยอินเตอร์เฟซที่ชื่อ patch-tun เชื่อมต่อกับ integration bridge ไปยัง tunnel bridge (br-tun)
- Tunnel bridge จะแปลง VXLAN-tagged ที่มาจาก integration bridge ไปยัง VXLAN tunnels การแปลงระหว่าง VXLAN IDs และ tunnel IDs จะดำเนินตาม OpenFlow rules ที่ติดตั้งบน br-tun
- Network จะประกอบไปด้วย br-int , br-tun , DHCP server และ br-ex

- Integration bridge บน network controller ช่วยในการเชื่อมต่อ instances ไปยัง network services เช่น routers and DHCP servers มันจะเชื่อมต่อไปยัง tunnel bridge (br-tun) ผ่าน patch interface (patch-tun)
- แต่ละ network ที่มี DHCP เปิดใช้งานจะมี DHCP server ทำงานอยู่บน network controller โดย DHCP server เป็น instance ของ dnsmasq ที่ทำงานใน network namespace โดยที่ network namespace คือสิ่งอำนวยความสะดวกที่ช่วยจัดกลุ่มของ processes ที่มี network stack (interfaces, routing tables, iptables rules) ที่แตกต่างจาก host.
- การจะออกไปข้างนอกจะมีแพ็คเก็ตไฟล์ผ่าน br-ex ที่ต้องผ่าน interface ที่อยู่ใน router name space ที่เชื่อมต่อมาที่ br-ex
- การทำงานของ Neutron network
- เมื่อ Tenant1 เริ่มรัน Instance1 จะมีการสร้างเส้นทางการเชื่อมต่อหมายเลข 1 ที่ VXLAN1 เพื่อใช้งาน DHCP server ตามรูปที่ 3.10



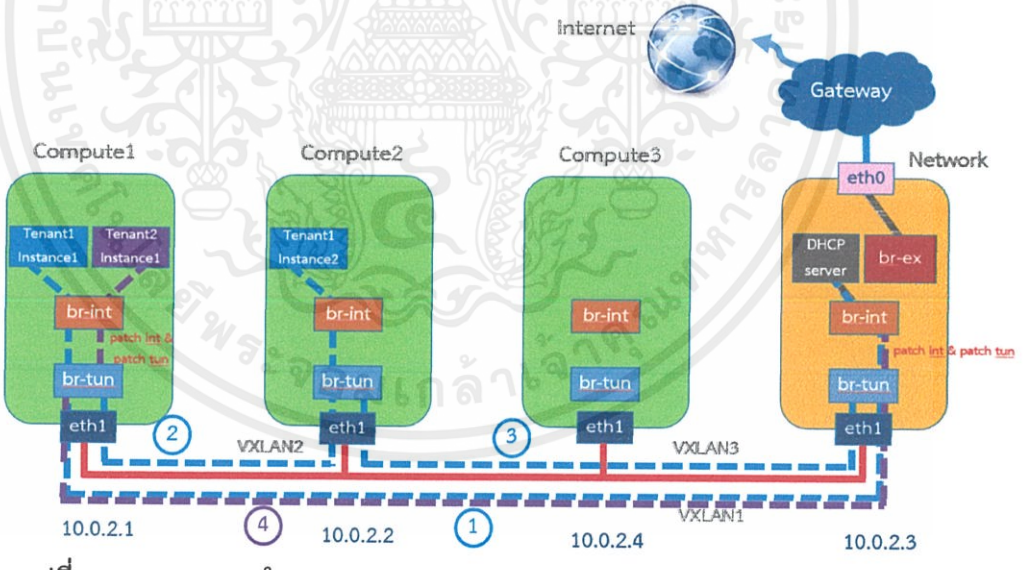
รูปที่ 3.10 แสดงการทำงานของ Neutron network ของ Tenant1 Instance1

- เมื่อ Tenant1 เริ่มรัน Instance2 จะมีการสร้างเส้นทางการเชื่อมต่อหมายเลข 2 ที่ VXLAN2 เพื่อเชื่อมต่อเครือข่ายของ Tenant1 และสร้างเส้นทางการเชื่อมต่อหมายเลข 3 ที่ VXLAN3 เพื่อใช้งาน DHCP server ตามรูปที่ 3.11



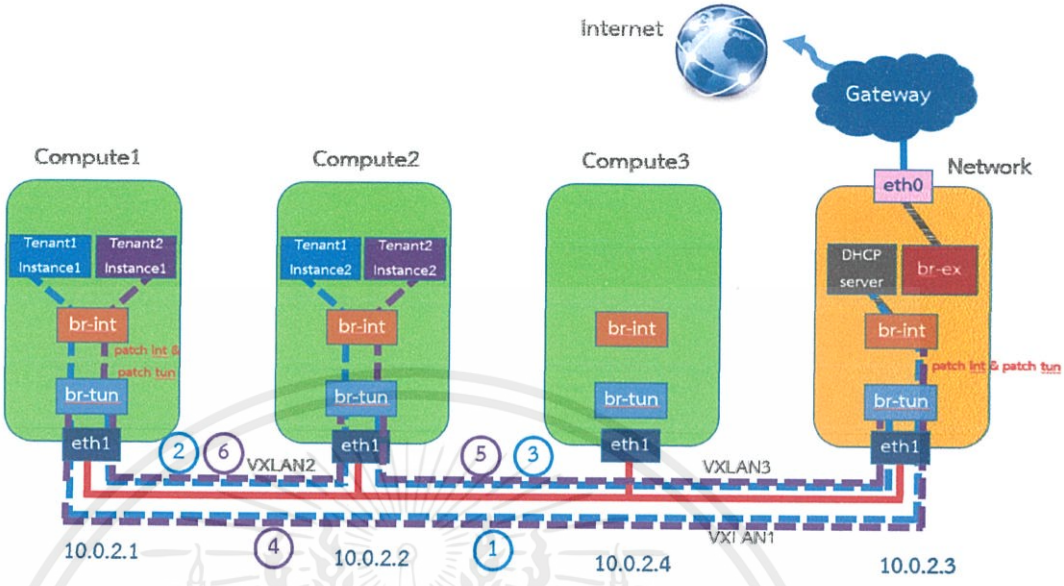
รูปที่ 3.11 แสดงการทำงานของ Neutron network ของ Tenant1 Instance2

- เมื่อ Tenant2 เริ่มรัน Instance1 จะมีการสร้างเส้นทางการเชื่อมต่อหมายเลข 4 ที่ VXLAN1 เพื่อใช้งาน DHCP server ตามรูปที่ 3.12



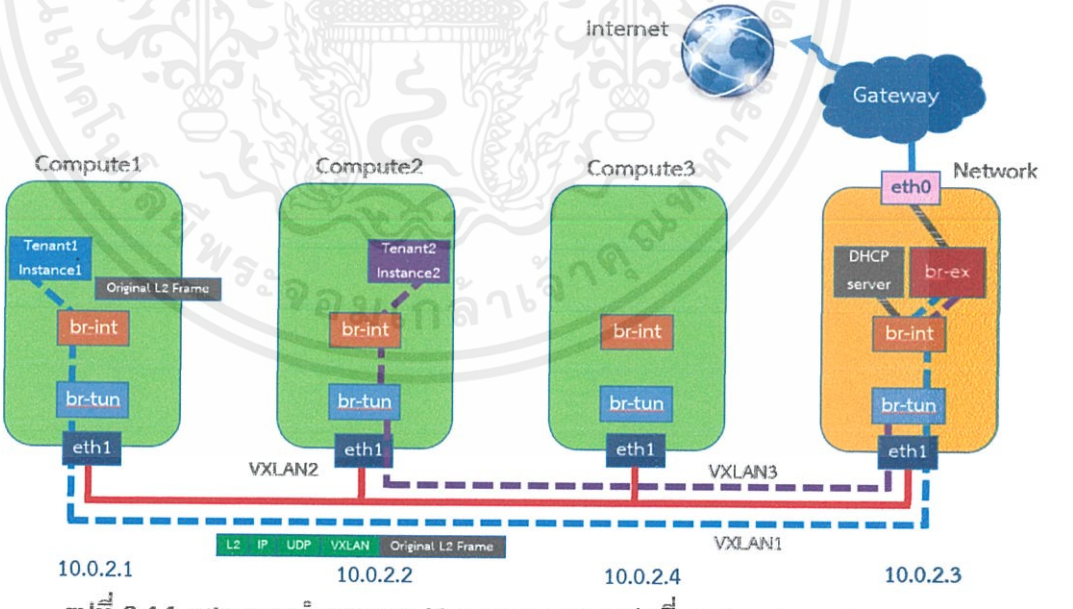
รูปที่ 3.12 แสดงการทำงานของ Neutron network ของ Tenant2 Instance1

- เมื่อ Tenant2 เริ่มรัน Instance2 จะมีการสร้างเส้นทางการเชื่อมต่อหมายเลข 6 ที่ VXLAN2 เพื่อเชื่อมต่อเครือข่ายของ Tenant2 และสร้างเส้นทางการเชื่อมต่อหมายเลข 5 ที่ VXLAN3 เพื่อใช้งาน DHCP server ตามรูปที่ 3.13



รูปที่ 3.13 แสดงการทำงานของ Neutron network ของ Tenant2 Instance2

- การเชื่อมต่อกับภายนอกของ instance เมื่อ instance ต้องการติดต่อกับภายนอก จะสร้าง Original L2 Frame ขึ้นมาแล้วส่งออกไป ขณะที่ผ่านออกจาก Compute จะมีการใส่ header ของ VXLAN, UDP, IP และ L2 ออกไปที่เครือข่ายทางกายภาพ ตามรูปที่ 3.14



รูปที่ 3.14 แสดงการทำงานของ Neutron network ที่ network node

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองสร้างpartutionบนfile system

4.1.1 สร้างpartitionsที่เป็นExt4

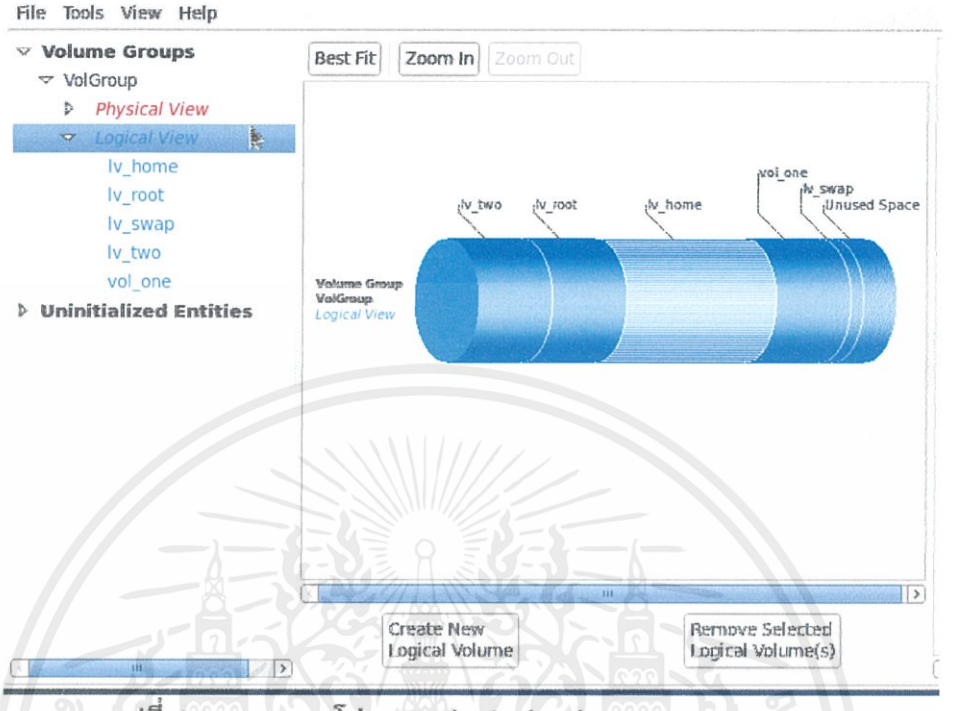
- -yum install system-config-lvm

คือคำสั่งใช้ลงโปรแกรม logical volume management เพื่อจัดการ logical volume

```
[root@storage1 ~]# yum install system-config-lvm
Loaded plugins: fastestmirror, refresh-packagekit
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: mirrors.thzhost.com
 * extras: mirrors.thzhost.com
 * updates: mirrors.thzhost.com
base                                                    | 3.7 kB     00:00
extras                                                  | 3.4 kB     00:00
glusterfs-epel                                          | 2.9 kB     00:00
glusterfs-noarch-epel                                   | 2.9 kB     00:00
updates                                                 | 3.4 kB     00:00
Resolving Dependencies
--> Running transaction check
---> Package system-config-lvm.noarch 0:1.1.12-17.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

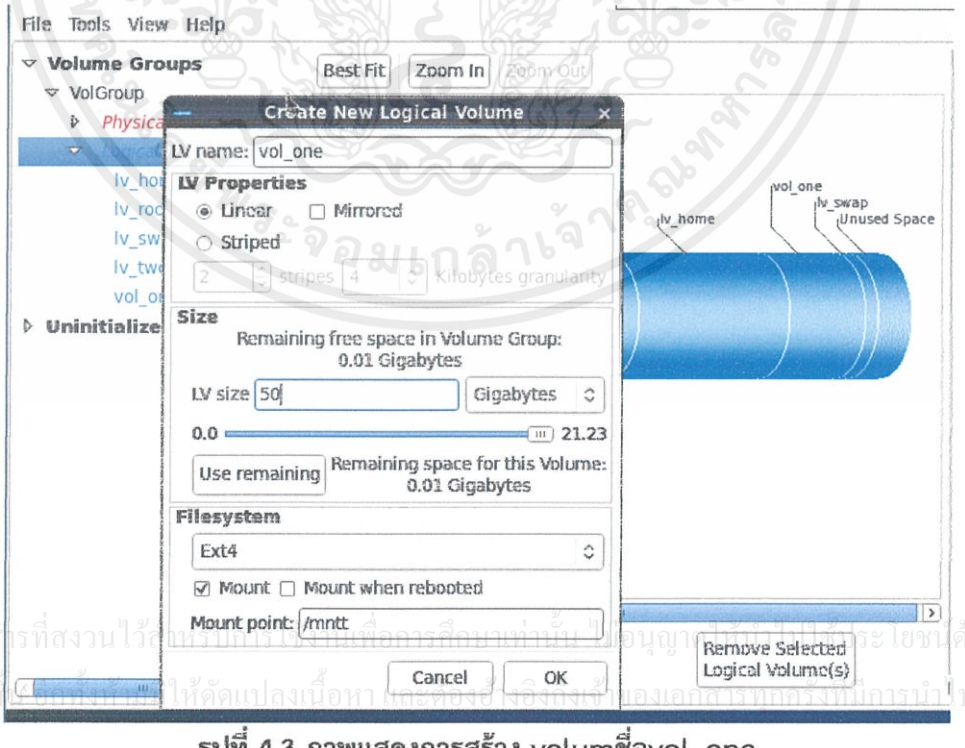
รูปที่ 4.1 ภาพแสดงคำสั่งที่ใช้ลงโปรแกรม logical volume management

จากรูปที่ 4.2 เป็นตัวโปรแกรม logical volume management ถ้าจะสร้าง volume ให้กด Create new logical volume



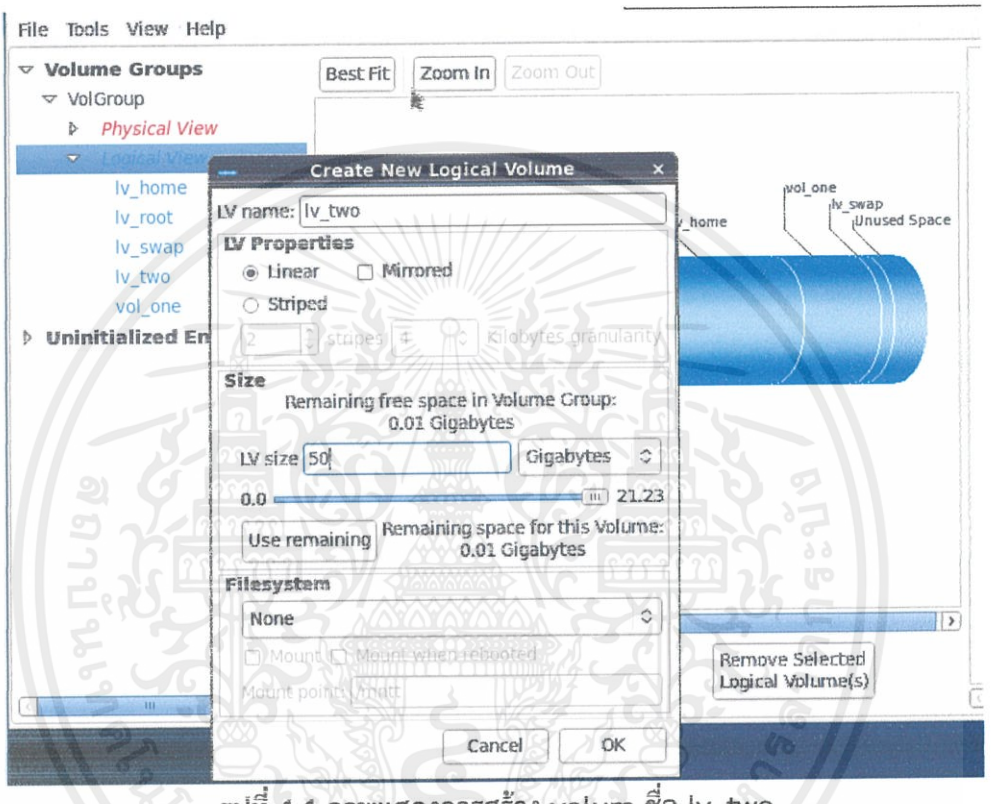
รูปที่ 4.2 ภาพแสดงโปรแกรม logical volume management

จากรูปที่ 4.3 เป็นการสร้าง volumชื่อvol_one เลือกประเภท filesystem คือ Ext4 mount point คือ /mntt



รูปที่ 4.3 ภาพแสดงการสร้าง volumชื่อvol_one

จากรูปที่ 4.4 เป็นการสร้าง volum ชื่อ lv_two เลือกประเภท filesystem คือ none เพราะไม่มีแบบ XFS ให้เลือก



รูปที่ 4.4 ภาพแสดงการสร้าง volum ชื่อ lv_two

รูปที่ 4.5 แสดงถึง volume ทั้งสองได้ถูกสร้างแล้ว

```
Disk /dev/mapper/VolGroup-vol_one: 53.7 GB, 53687091200 bytes
255 heads, 63 sectors/track, 6527 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000


Disk /dev/mapper/VolGroup-lv_two: 53.7 GB, 53687091200 bytes
255 heads, 63 sectors/track, 6527 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```


รูปที่ 4.5 ภาพแสดง volume ทั้งสองได้ถูกสร้าง

4.1.2 สร้างpartitionsที่เป็นXFS

- yum install xfsprogs yum-kmod kmod-xfs-xen

คือการโหลดและลงตัวที่จะใช้สร้าง filesystem XFS

```
[root@storage1 ~]# yum install xfsprogs yum-kmod kmod-xfs-xen
Loaded plugins: fastestmirror, refresh-packagekit
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: mirrors.thzhost.com
 * extras: mirrors.thzhost.com
 * updates: mirrors.thzhost.com
No package yum-kmod available.
No package kmod-xfs-xen available.
Resolving Dependencies
--> Running transaction check
---> Package xfsprogs.x86_64 0:3.1.1-16.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

รูปที่ 4.6 ภาพแสดงการโหลดและลงตัวที่จะใช้สร้าง filesystem XFS

- mkfs.xfs /dev/mapper/VolGroup-lv_two

คือการสร้าง file sytem XFS ไปที่ volume ของ /dev/mapper/VolGroup-lv_two

```
[root@storage1 ~]# mkfs.xfs /dev/mapper/VolGroup-lv_two
meta-data=/dev/mapper/VolGroup-lv_two isize=256    agcount=4, agsize=3276800 b
s
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=13107200, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=6400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

รูปที่ 4.7 ภาพแสดงการสร้าง file sytem XFS ไปที่ volume

- mkdir /xfsmnt

คือการสร้างไดเร็กทอรี่ไว้ใช้เป็น mount point

- mount /dev/mapper/VolGroup-lv_two /xfsmnt

คือการ mount volume ให้เชื่อมกับ mount point

```
[root@storage1 ~]# mkdir /xfsmnt
[root@storage1 ~]# mount /dev/mapper/VolGroup-lv_two /xfsmnt
[root@storage1 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root
                       50G   12G   36G  25% /
tmpfs                 7.7G  360K  7.7G   1% /dev/shm
/dev/sda1             485M   81M  379M  18% /boot
/dev/mapper/VolGroup-lv_home
                       99G  189M   94G   1% /home
/dev/mapper/VolGroup-vol_one
                       50G  180M   47G   1% /mntt
/dev/mapper/VolGroup-lv_two
                       50G   33M   50G   1% /xfsmnt
```

รูปที่ 4.8 ภาพแสดงการสร้างไดเร็กทอรี่ไว้ใช้เป็น mount point

- vi /etc/fstab

คือคำสั่งนี้เข้าไปเพิ่ม volume และชนิด file system ไปใน fstab เพื่อให้ตอนเปิดเครื่องใหม่ให้ mount ค้างไว้

```
# /etc/fstab
# Created by anaconda on Wed Sep 17 05:26:47 2014
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/VolGroup-lv_root /                       ext4    defaults        1 1
UUID=a8b7ed33-64e0-43cf-a45f-71dcbaadb891 /boot                   ext4    defaul
ts        1 2
/dev/VolGroup/lv_home           /home                   ext4    defaults        1 2
/dev/mapper/VolGroup-lv_swap swap                    swap    defaults        0 0
tmpfs                   /dev/shm                tmpfs   defaults        0 0
devpts                  /dev/pts                devpts  gid=5,mode=620  0 0
sysfs                   /sys                    sysfs   defaults        0 0
proc                    /proc                   proc    defaults        0 0
/dev/mapper/VolGroup-lv_two    /xfsmnt          xfs     defaults        0 0
/dev/mapper/VolGroup-vol_one   /mntt            ext4    defaults        0 0
```

รูปที่ 4.9 ภาพแสดงคำสั่งที่เข้าไปเพิ่ม volume และชนิด file system

4.2การทดลองGlusterFS

4.2.1การสร้าง Striped Volume

1) Setup

- เข้าไปแก้ไขไฟล์ hosts เพื่อให้แต่ละเครื่องรู้จักกัน (ทำแบบนี้กับทุกเครื่อง)

```
[root@storage ~]# vi /etc/hosts
# add GlusterFS servers
192.168.10.141   storage.example.com gls1
192.168.10.142   storage1.example.com gls2
```

```
127.0.0.1 localhost
127.0.0.1 storage
192.168.10.141 storage.example.com gls1
192.168.10.142 storage1.example.com gls2
```

รูปที่ 4.10 ภาพแสดงแก้ไขไฟล์ hosts

2) Installation

- เพิ่ม GlusterFS เข้าไปในเครื่อง (ทำแบบนี้กับทุกเครื่อง)

```
[root@storage ~]# wget -P /etc/yum.repos.d
http://download.gluster.org/pub/gluster/glusterfs/LATEST/CentOS/glusterfs-epel.repo
```

2.1) การติดตั้งบนเครื่อง server (192.168.10.141 , 192.168.10.142)

- ติดตั้ง glusterfs server ลงบนเครื่อง Host (ทำแบบนี้กับทุกที่เป็น server)

```
[root@storage ~]# yum -y install glusterfs glusterfs-fuse
glusterfs-server
```

- Start glusterfs services บนทุก servers

```
[root@storage ~]# /etc/init.d/glusterd start
```

- ตรวจสอบว่า glusterfs ได้ start แล้ว

```
[root@storage ~]# chkconfig glusterfsd on
```


2.2) การติดตั้งบนเครื่อง client (192.168.10.135)

```
[root@centosmini ~]# yum -y install glusterfs glusterfs-fuse
```

3) Creating Trusted Storage Pool

[root@storage ~]# gluster peer probe gls2

- ตรวจสอบ peer status

[root@storage ~]# gluster peer status

Number of Peers: 1

Hostname: gls2

Uuid: 585e58b6-3fd0-4988-836a-31372c686598

State: Peer in Cluster (Connected)

4) Stripped

- สร้าง volume ที่เป็นแบบ stripe

[root@storage ~]# gluster volume create stripe-volume stripe 2 transport tcp gls1:/stripe-vol gls2:/stripe-vol force

- Start volume ที่สร้างไว้

[root@storage ~]# gluster volume start stripe-volume

- ตรวจสอบสถานะของ volume

[root@storage ~]# gluster volume info stripe-volume

```
[root@storage ~]# gluster volume create stripe-volume stripe 2 transport tcp gls
1:/stripe-vol gls2:/stripe-vol force
volume create: stripe-volume: success: please start the volume to access data
[root@storage ~]# gluster volume start stripe-volume
volume start: stripe-volume: success
[root@storage ~]# gluster volume info stripe-volume

Volume Name: stripe-volume
Type: Stripe
Volume ID: 49684b19-3741-4b7d-8719-46bb260e69ff
Status: Started
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: gls1:/stripe-vol
Brick2: gls2:/stripe-vol
[root@storage ~]#
```

รูปที่ 4.11 ภาพแสดงการสร้าง Volume ที่เป็นแบบ stripped

4.2.2 การเพิ่ม Brick

1) ติดตั้ง GlusterFS บน server ใหม่ให้เรียบร้อย (192.168.10.144)

2) แก้ไขไฟล์ /etc/hosts โดยเพิ่ม (server ทุกตัว)

```
# add GlusterFS servers
192.168.10.141   storage.example.com gls1
192.168.10.142   storage1.example.com gls2
192.168.10.144 storage2.example.com gls3
```

3) probe server ใหม่ โดยทำบน gls1 (192.168.10.141)

```
[root@storage ~]# gluster peer probe gls3
```

4) stop volume ที่ต้องการเพิ่ม (192.168.10.141)

```
[root@storage ~]# gluster volume stop stripe-volume
```

5) เพิ่ม volume (192.168.10.141)

```
[root@storage ~]# gluster volume add-brick stripe-volume stripe 3
gls3:/stripe-vol force
```

6) ตรวจสอบสถานะของ volume (192.168.10.141)

```
root@storage ~]# gluster volume info stripe-volume
```

7) Start volume ที่ทำไว้ (192.168.10.141)

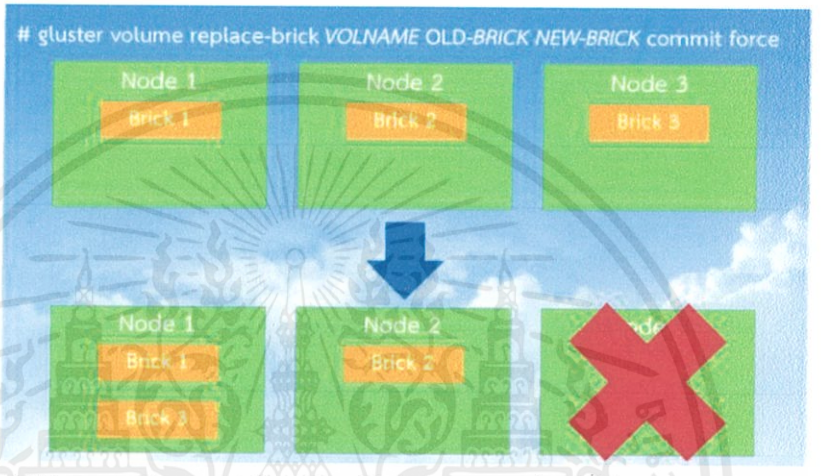
```
[root@storage ~]# gluster volume start stripe-volume
```

```
[root@storage ~]# gluster volume stop stripe-volume
Stopping volume will make its data inaccessible. Do you want to continue? (y/n)
y
volume stop: stripe-volume: success
[root@storage ~]# gluster volume add-brick stripe-volume stripe 3 gls3:/stripe-v
ol force
Changing the 'stripe count' of the volume is not a supported feature. In some ca
ses it may result in data loss on the volume. Also there may be issues with regu
lar filesystem operations on the volume after the change. Do you really want to
continue with 'stripe' count option ?  (y/n) y
volume add-brick: success
[root@storage ~]# gluster volume info stripe-volume

Volume Name: stripe-volume
Type: Stripe
Volume ID: 49684b19-3741-4b7d-8719-46bb260e69ff
Status: Stopped
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: gls1:/stripe-vol
Brick2: gls2:/stripe-vol
Brick3: gls3:/stripe-vol
[root@storage ~]#
```

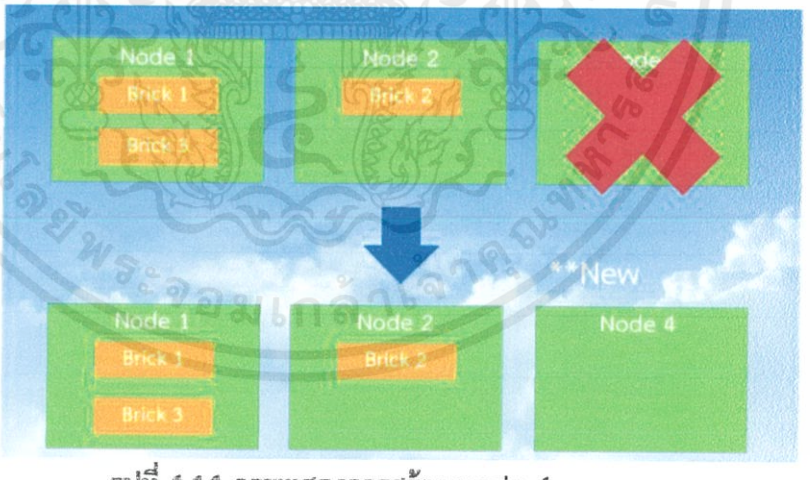
รูปที่ 4.12 ภาพแสดงการเพิ่ม Brick

4.2.3 Migrating Volumes

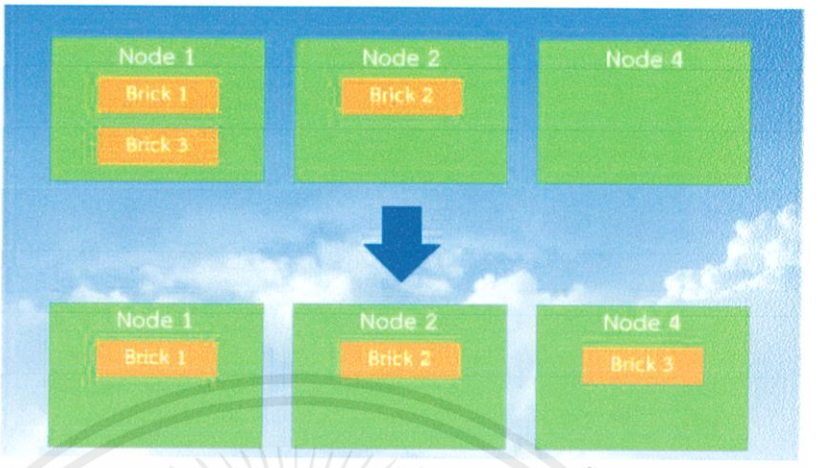
- เป็นการจัดการกับ Brick แบบออนไลน์ สามารถนำไปใช้ประโยชน์ได้หลายอย่าง เช่นการเพิ่มหรือลด Node
- ใช้คำสั่ง # gluster volume replace-brick *VOLNAME* OLD-*BRICK NEW-BRICK* commit force



รูปที่ 4.13 ภาพแสดงการ Migrating Volumes node 3 ไป node 1



รูปที่ 4.14 ภาพแสดงการสร้าง node 4



รูปที่ 4.15 ภาพแสดงการ Migrating Volumes node 1 ไป node 4

4.2.4การทดสอบเก็บไฟล์

1) ทำการ mount directory จาก client (192.168.10.135) ไปที่ gls1 (192.168.10.141)

```
[root@centosmini ~]# mount -t glusterfs 192.168.10.141:/stripe-volume
/mnt
```

2) ตรวจสอบการ mount

```
[root@centosmini ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vg_centosmini-lv_root
                      18102140    923116  16259472   6% /
tmpfs                   510188         0    510188   0% /dev/shm
/dev/vda1               495844     54897    415347  12% /boot
192.168.10.141:/stripe-volume
                     154556160  46802304  99889536  32% /mnt
```

3) สร้างไฟล์ทดสอบ

```
[root@centosmini ~]# echo 'This is a test' > test.txt
```

4) คัดลอกไฟล์ทดสอบไปยังจุดที่ mount

```
[root@centosmini ~]# cp test.txt /mnt
```

5) เข้าไปดูที่ client ว่ามีไฟล์อยู่หรือไม่ (Client : 192.168.10.135)

```
[root@centosmini ~]# cd /mnt
[root@centosmini mnt]# ls -l
```

total 1

-rw-r--r--. 1 root root 15 Nov 17 2014 test.txt

```
[root@centosmini ~]# mount -t glusterfs 192.168.10.141:/stripe-volume /mnt
[root@centosmini ~]# df
Filesystem           1K-blocks     Used Available Use% Mounted on
/dev/mapper/vg_centosmini-lv_root
                      18102140   923116  16259472   6% /
tmpfs                   510188        0    510188   0% /dev/shm
/dev/vda1               495844    54897    415347  12% /boot
192.168.10.141:/stripe-volume
                      154556160 46802304  99889536  32% /mnt
[root@centosmini ~]# echo 'This is a test' > test.txt
[root@centosmini ~]# cp test.txt /mnt
[root@centosmini ~]# cd /mnt
[root@centosmini mnt]# ls -l
total 1
-rw-r--r--. 1 root root 15 Nov 17  2014 test.txt
[root@centosmini mnt]#
```

รูปที่ 4.16 ภาพแสดงการส่งไฟล์

6) เข้าไปดูที่ server ทั้ง 3 ตัวว่ามีไฟล์อยู่หรือไม่

Storage: 192.168.10.141

```
[root@storage ~]# cd /stripe-vol
[root@storage stripe-vol]# ls -l
total 8
-rw-r--r--. 2 root root 15 Nov 17 07:08 test.txt
[root@storage stripe-vol]#
```

รูปที่ 4.17 ภาพแสดงไฟล์ที่อยู่ใน Storage

Storage1: 192.168.10.142

```
[root@storage1 ~]# cd /stripe-vol
[root@storage1 stripe-vol]# ls -l
total 4
-rw-r--r--. 2 root root 0 Nov 16 12:06 test.txt
[root@storage1 stripe-vol]#
```

รูปที่ 4.18 ภาพแสดงไฟล์ที่อยู่ใน Storage1

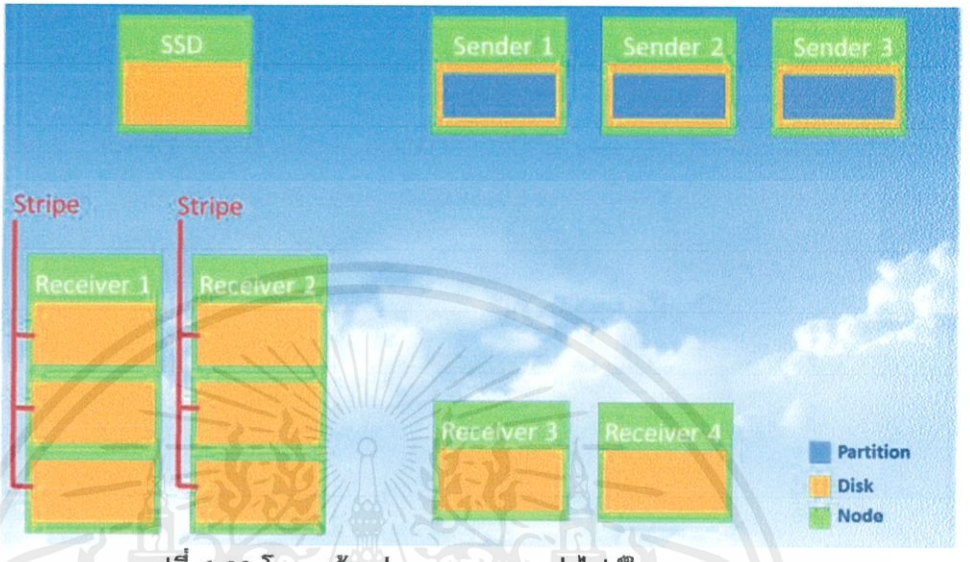
Storage2: 192.168.10.144

```
[root@storage2 ~]# cd /stripe-vol
[root@storage2 stripe-vol]# ls -l
total 4
-rw-r--r-- 2 root root 0 Nov 17 00:06 test.txt
[root@storage2 stripe-vol]#
```

รูปที่ 4.19 ภาพแสดงไฟล์ที่อยู่ใน Storage2

4.2.5การทดสอบ Performance ของการทำ Striped

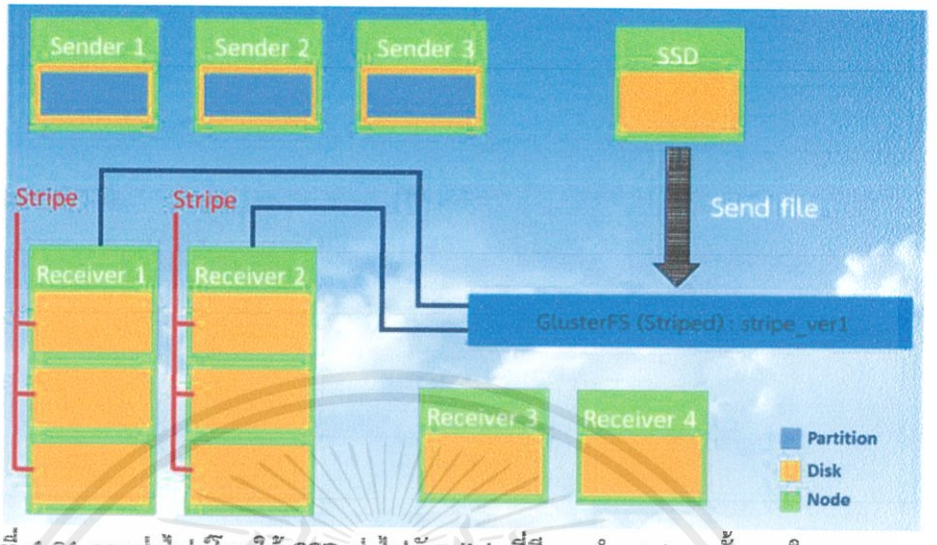
1) เตรียมเครื่องคอมพิวเตอร์สำหรับทดลองดังนี้



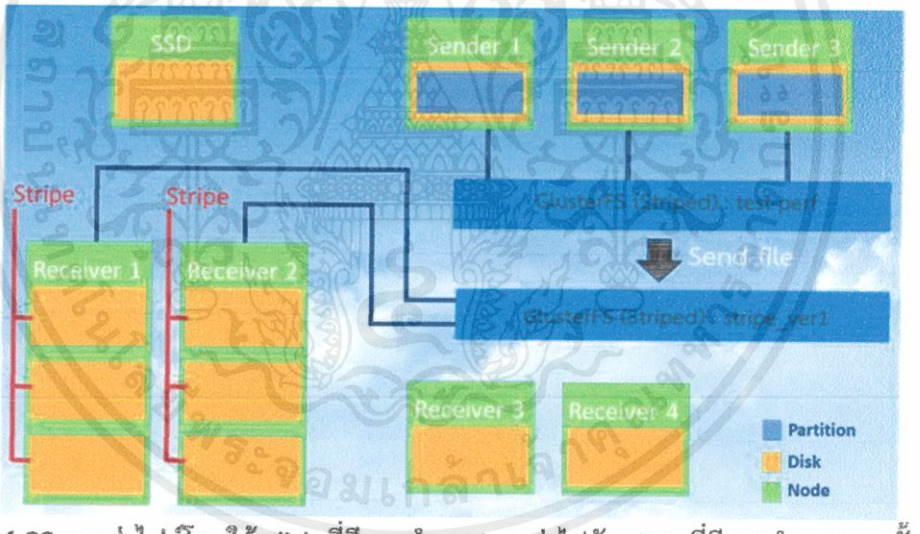
รูปที่ 4.20 โครงสร้างก่อนการทดลองส่งไฟล์ใน

- เครื่องคอมพิวเตอร์ที่มี disk เป็น SSD จำนวน 1 เครื่อง
- เครื่องคอมพิวเตอร์ที่มี disk ธรรมดา 1 disk จำนวน 5 เครื่อง
- เครื่องคอมพิวเตอร์ที่มี disk ธรรมดา 3 disk จำนวน 2 เครื่อง

2) นำเครื่องคอมพิวเตอร์ที่มี disk ธรรมดา 3 disk มาทำ striped กันภายในเครื่อง โดยไม่ใช้ GlusterFS ทำทั้ง 2 เครื่อง
3) ใช้ GlusterFS สร้าง Volume แบบ striped 3 brick ที่ sender1, sender2 และ sender3
4) ใช้ GlusterFS สร้าง Volume แบบ striped 2 brick ที่ receiver1 และ receiver2
5) ทดสอบส่งไฟล์ขนาด 1GB 10GB และ 20GB จาก SSD และ volume แบบ striprd 3 brick ที่เตรียมไว้ดังภาพและบันทึกผล



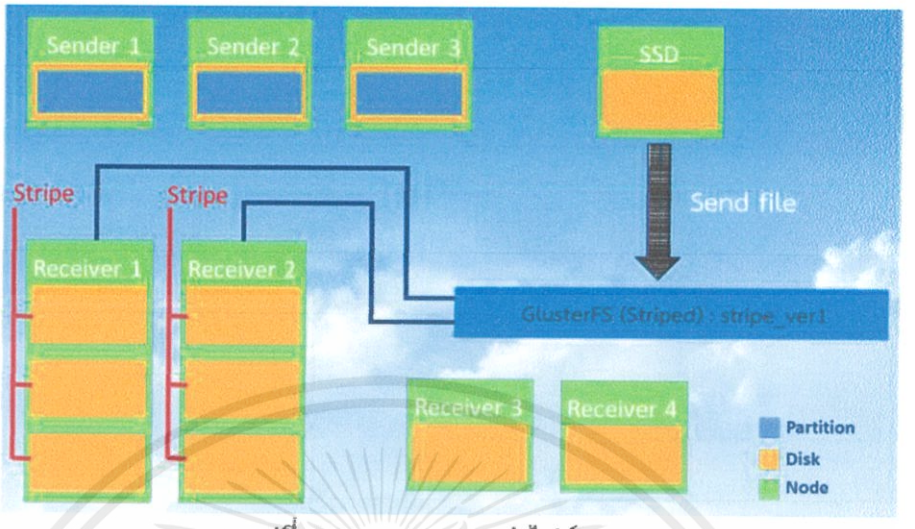
รูปที่ 4.21 การส่งไฟล์โดยใช้ SSD ส่งไปยัง disk ที่มีการทำ stripe ทั้งภายในและภายนอก



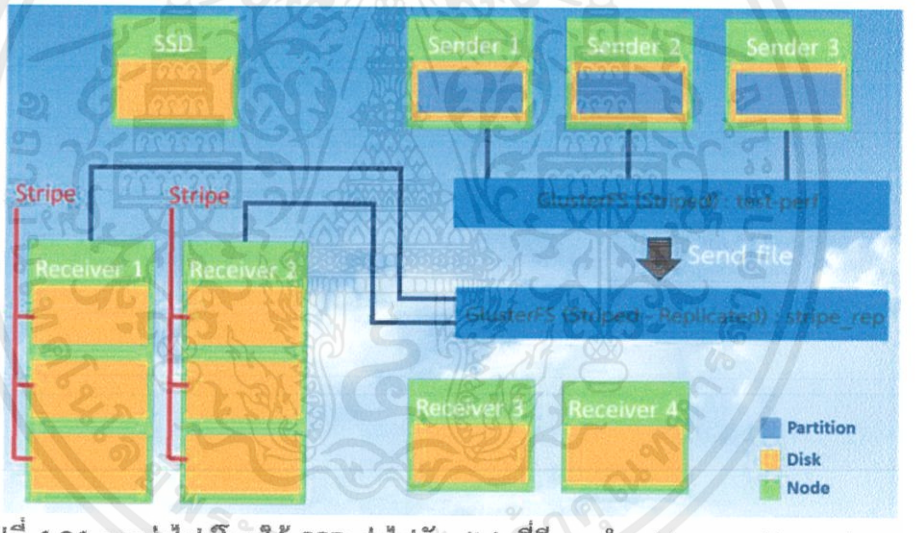
รูปที่ 4.22 การส่งไฟล์โดยใช้ disk ที่มีการทำ stripe ส่งไปยัง disk ที่มีการทำ stripe ทั้งภายในและภายนอก

6) ใช้ GlusterFS สร้าง Volume แบบ Striped - Replicated 4 brick ที่ receiver1 และ receiver2

7) ทดสอบส่งไฟล์ขนาด 1GB 10GB และ 20GB จาก SSD และ volume แบบ striprd 3 brick ที่เตรียมไว้ดังภาพและบันทึกผล



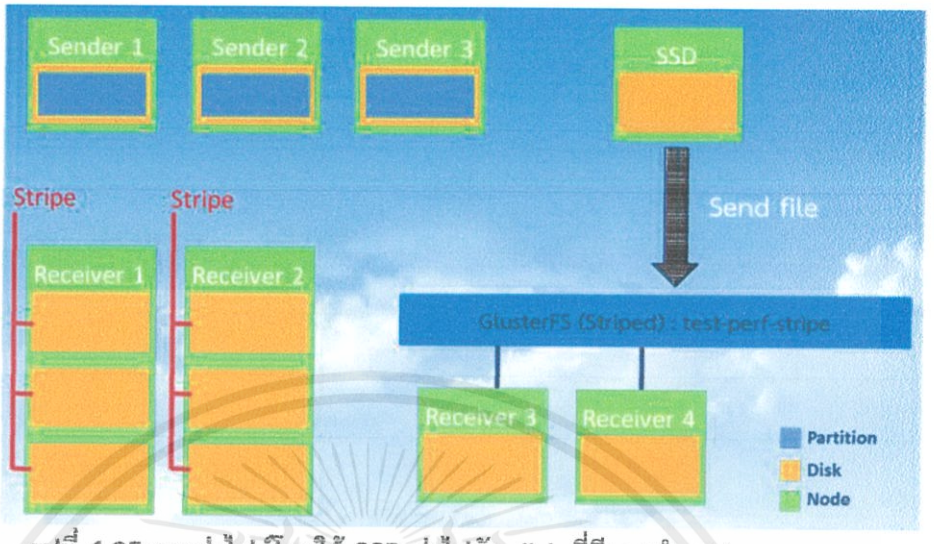
รูปที่ 4.23 การทดสอบส่งไฟล์



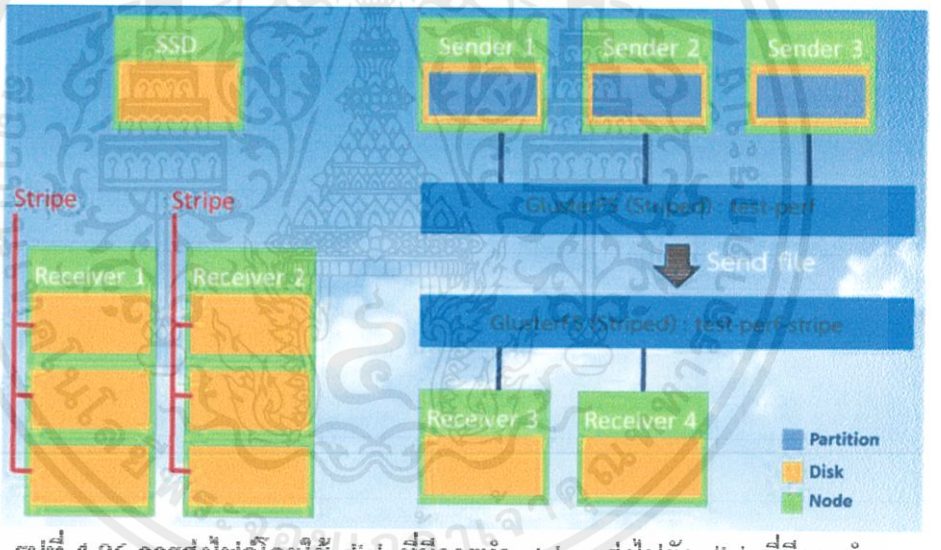
รูปที่ 4.24 การส่งไฟล์โดยใช้ SSD ส่งไปยัง disk ที่มีการทำ stripe-replicated ภายนอก

8) ใช้ GlusterFS สร้าง Volume แบบ striped 2 brick ที่ receiver3 และ receiver4

9) ทดสอบส่งไฟล์ขนาด 1GB 10GB และ 20GB จาก SSD และ volume แบบ striprd 3 brick ที่เตรียมไว้ดังภาพและบันทึกผล



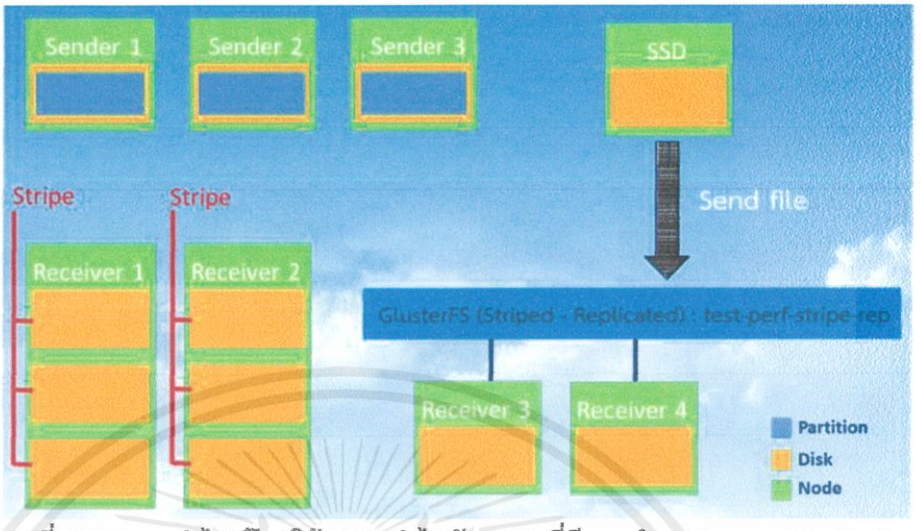
รูปที่ 4.25 การส่งไฟล์โดยใช้ SSD ส่งไปยัง disk ที่มีการทำ stripe



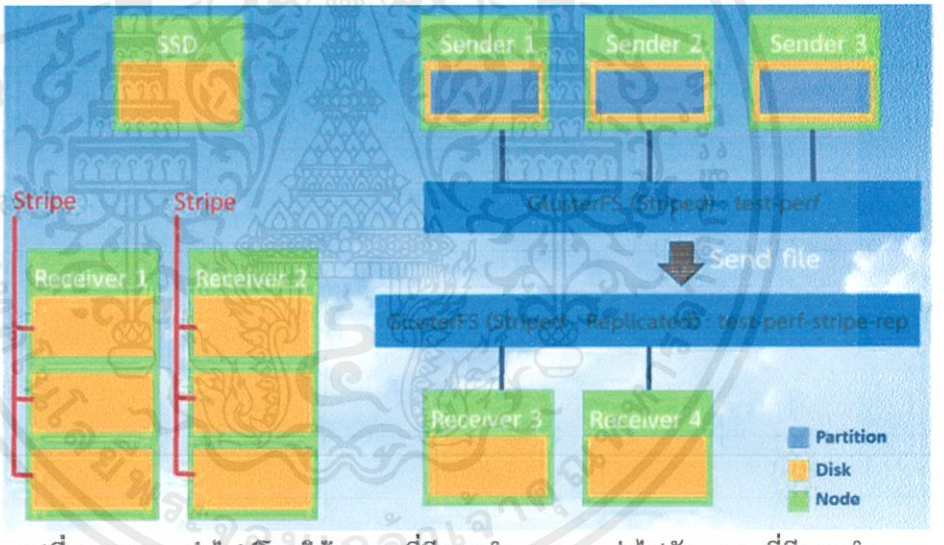
รูปที่ 4.26 การส่งไฟล์โดยใช้ disk ที่มีการทำ stripe ส่งไปยัง disk ที่มีการทำ stripe

10) ใช้ GlusterFS สร้าง Volume แบบ Striped - Replicated 4 brick ที่ receiver3 และ receiver4

11) ทดสอบส่งไฟล์ขนาด 1GB 10GB และ 20GB จาก SSD และ volume แบบ striprd 3 brick ที่เตรียมไว้ดังภาพและบันทึกผล



รูปที่ 4.27 การส่งไฟล์โดยใช้ SSD ส่งไปยัง disk ที่มีการทำ stripe-replicated



รูปที่ 4.28 การส่งไฟล์โดยใช้ disk ที่มีการทำ stripe ส่งไปยัง disk ที่มีการทำ stripe-replicated

Stripe 3 node	create file	Stripe (stripe ภายใน)			Stripe-Replicate (stripe ภายใน)			Stripe (ไม่ stripe ภายใน)			Stripe-Replicate (ไม่ stripe ภายใน)		
		ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)
1 GB	59.75	46.15	45.96	46.05	94.36	93.37	93.87	9.48	9.28	9.38	19.63	18.47	19.05
10 GB	548.68	458.41	458.50	458.45	995.72	928.19	961.96	165.08	167.93	166.51	231.45	226.96	229.21
20 GB	1146.07	915.78	915.86	915.82	1854.65	1855.26	1854.95	351.74	357.46	354.60	467.31	494.18	480.74

	rate (kBs)	rate (kBs)	rate (kBs)	rate (kBs)	rate (kBs)
1 GB	22768.40	22871.98	11170.87	111788.49	55036.14
10 GB	22871.98	22880.21	11090.43	62975.27	45748.09
20 GB	22899.21	22924.91	11305.70	59141.59	43623.10

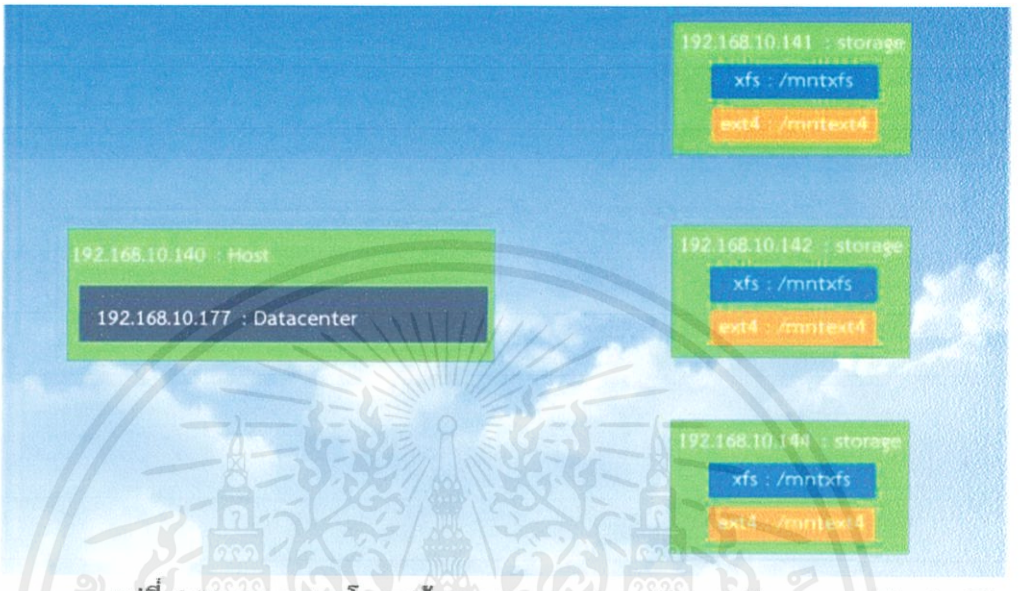
SSD	create file	Stripe (stripe ภายใน)			Stripe-Replicate (stripe ภายใน)			Stripe (ไม่ stripe ภายใน)			Stripe-Replicate (ไม่ stripe ภายใน)		
		ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	เฉลี่ย(s)
1 GB	3.324	46.10	52.92	49.51	93.84	94.84	94.34	9.45	10.81	10.13	19.29	20.96	20.12
10 GB	41.044	457.58	459.00	458.29	921.77	978.17	949.97	92.40	94.90	93.65	193.55	193.62	193.59
20 GB	83.353	913.87	915.72	914.79	1846.25	1856.69	1851.47	184.51	186.19	185.35	388.13	387.64	387.88

	rate (kBs)	rate (kBs)	rate (kBs)	rate (kBs)	rate (kBs)
1 GB	21179.29	11111.57	103542.61	52108.33	
10 GB	22880.21	11037.98	111966.34	54165.76	
20 GB	22924.91	11326.94	113145.20	54066.47	

รูปที่ 4.29 ผลการทดสอบส่งไฟล์ทั้งหมด

4.3 การทดสอบ Object Storage – GlusterFS

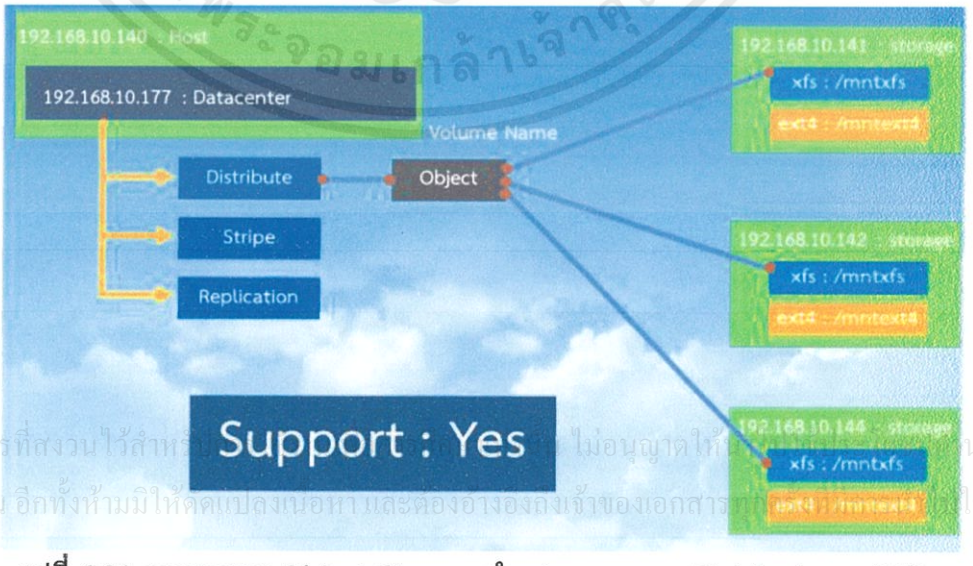
1) เตรียมเครื่องคอมพิวเตอร์สำหรับทดลองดังนี้



รูปที่ 4.30 ภาพแสดงโครงสร้างของการทดสอบ Object Storage – GlusterFS

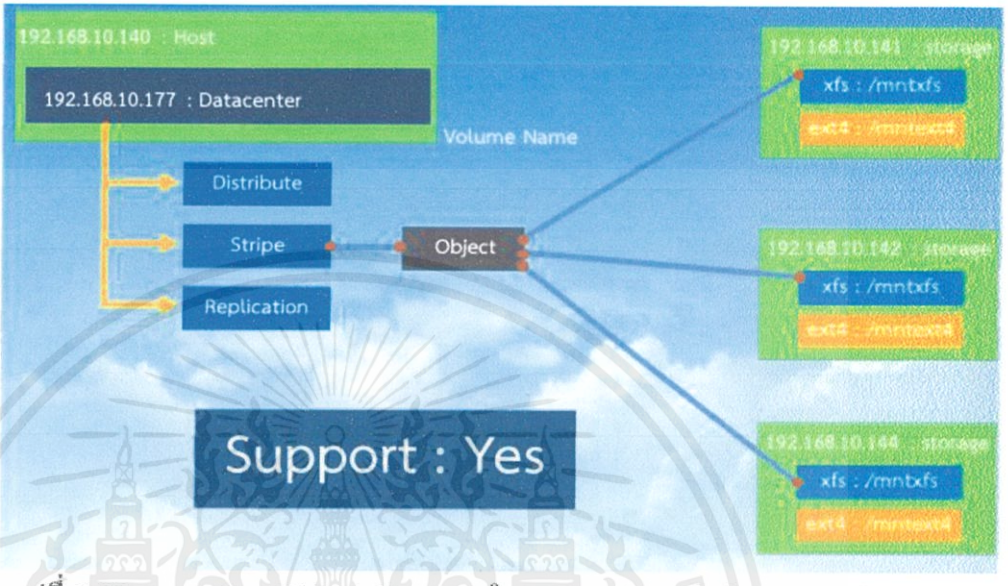
- สร้างเครื่อง Host แล้วรันเครื่องข่าลองที่เป็น VM บนนั้น
- เตรียมเครื่องสำหรับทำเป็น Storage Node 3 เครื่อง ใช้ LVM แบ่งออกมา 2 Partition แล้วลง file system เป็น XFS และ EXT4 ทุกเครื่อง

2) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



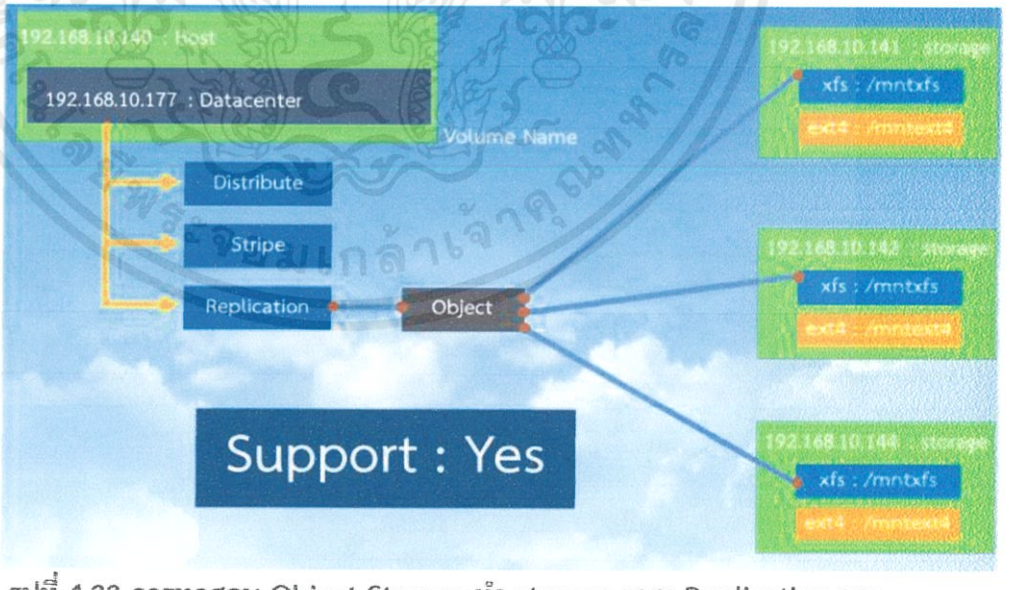
รูปที่ 4.31 การทดสอบ Object Storage ทำ storage แบบ Distribute บน XFS

3) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



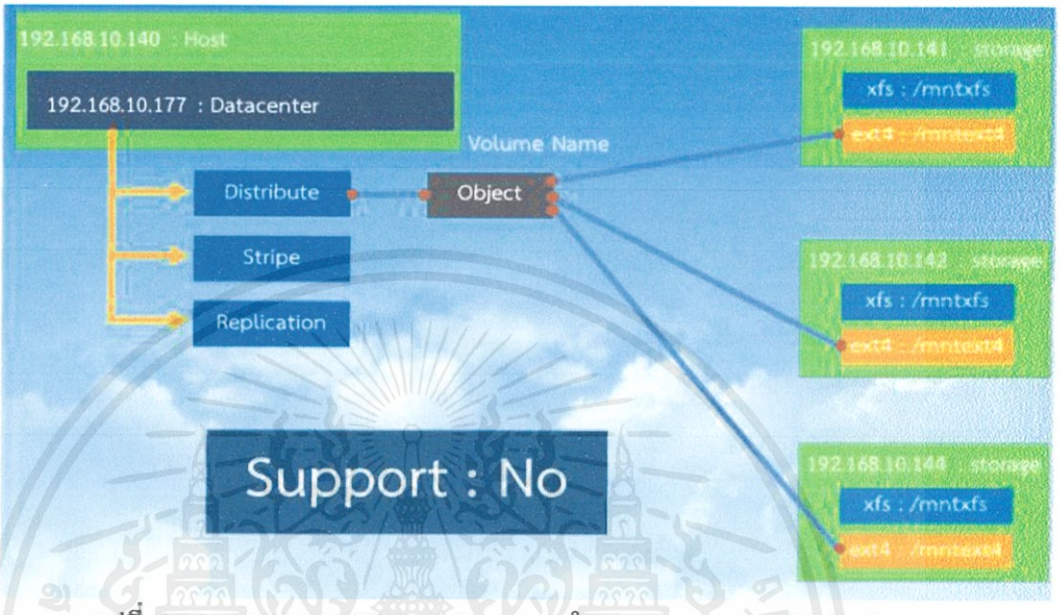
รูปที่ 4.32 การทดสอบ Object Storage ทำ storage แบบ Stripe บน XFS

4) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



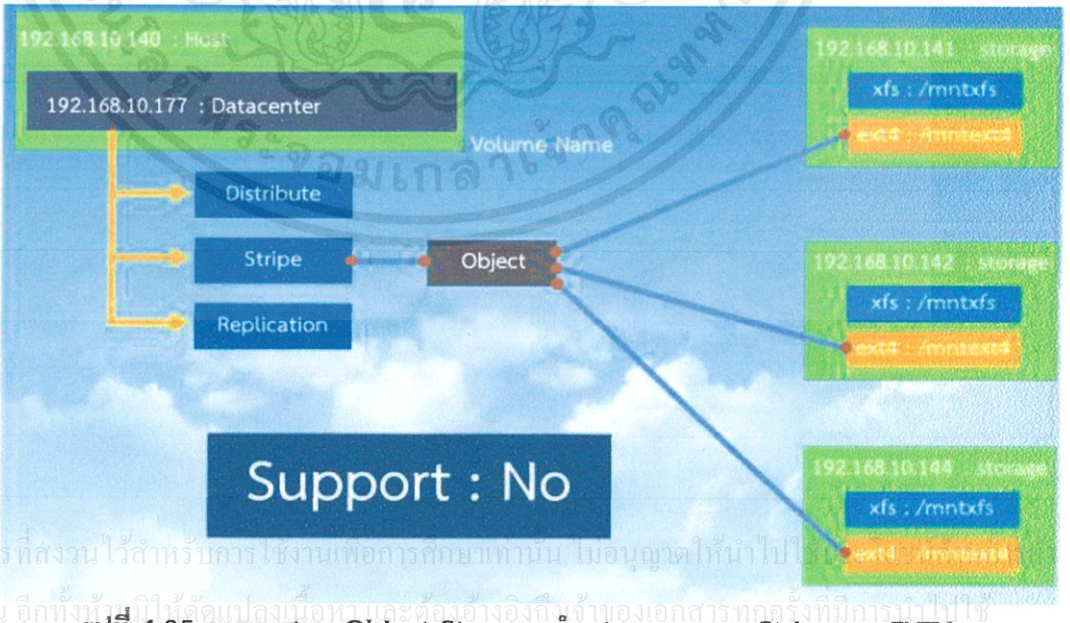
รูปที่ 4.33 การทดสอบ Object Storage ทำ storage แบบ Replication บน XFS

5) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



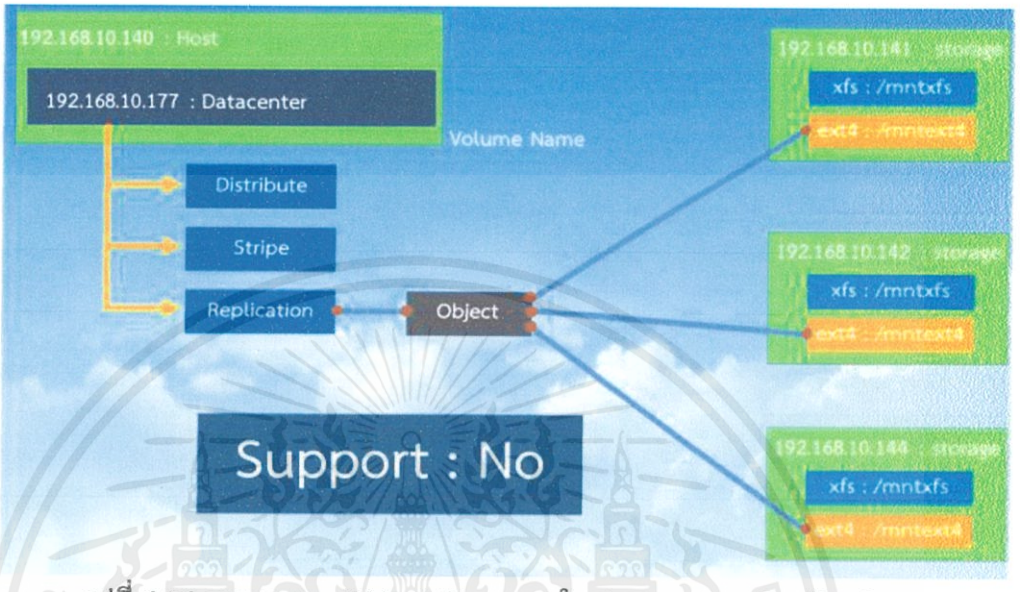
รูปที่ 4.34 การทดสอบ Object Storage ทำ storage แบบ Distribute บน EXT4

6) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



รูปที่ 4.35 การทดสอบ Object Storage ทำ storage แบบ Stripe บน EXT4

7) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Object แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



รูปที่ 4.36 การทดสอบ Object Storage ทำ storage แบบ Replication บน EXT4

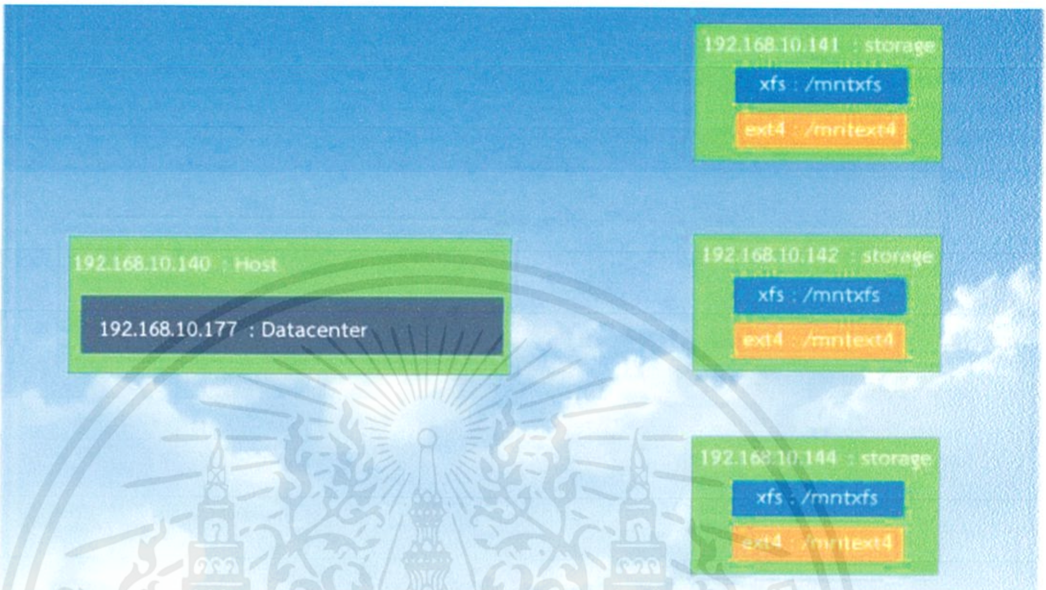
4.3.1 สรุปผลการทดลอง

Volume Types	XFS	EXT4
Distribute	Yes	No
Stripe	Yes	No
Replication	Yes	No

ตารางที่ 4.1 ผลการทดสอบ Object Storage – GlusterFS

4.4การทดสอบ Block Storage – GlusterFS

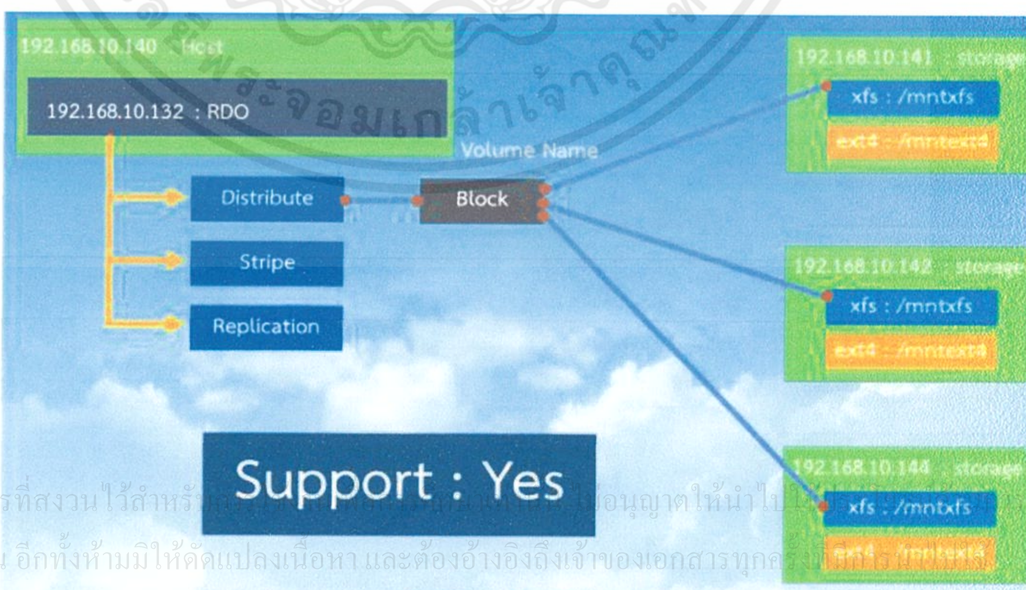
1) เตรียมเครื่องคอมพิวเตอร์สำหรับทดลองดังนี้



รูปที่ 4.37 ภาพแสดงโครงสร้างของการทดสอบ Block Storage

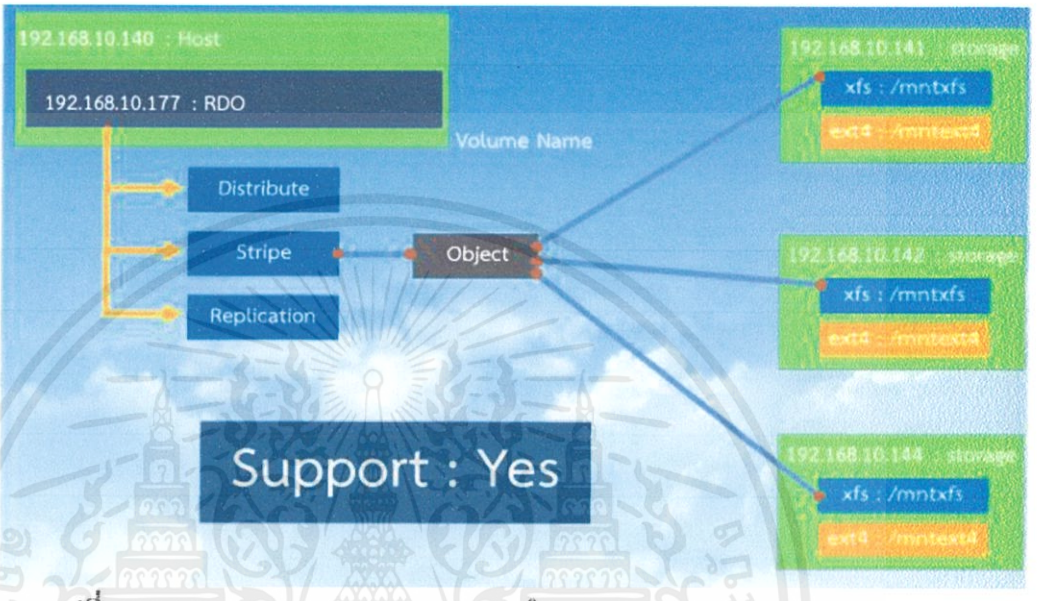
- สร้างเครื่อง Host แล้วรันเครื่องจำลองที่เป็น VM บนนั้น
- เตรียมเครื่องสำหรับทำเป็น Storage Node 3 เครื่อง ใช้ LVM แบ่งออกมา 2 Partition แล้วลง file system เป็น XFS และ EXT4 ทุกเครื่อง

2) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



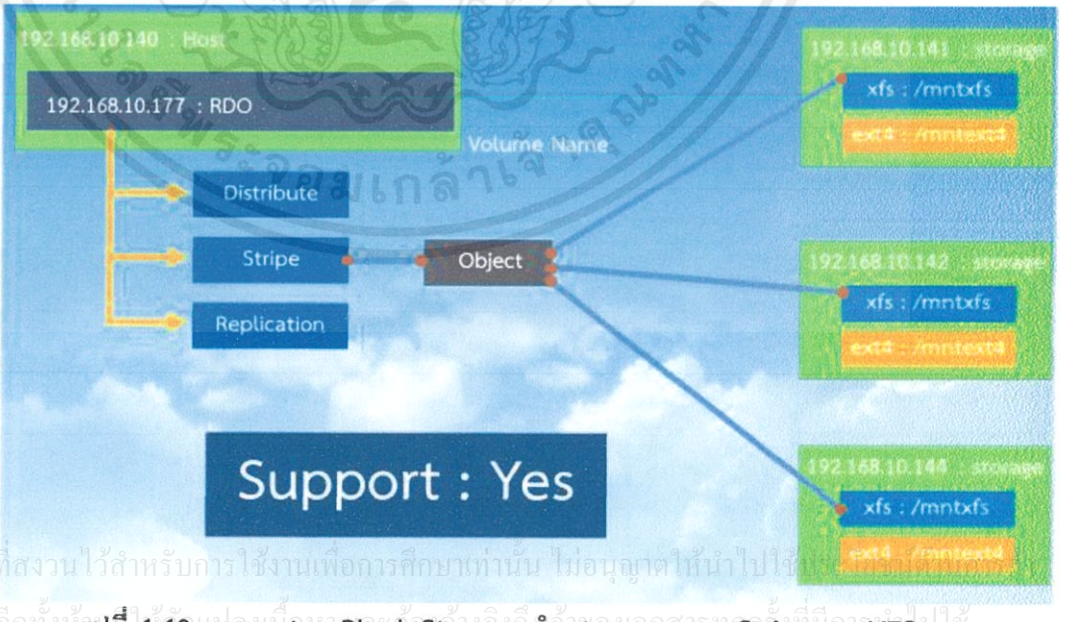
รูปที่ 4.38 การทดสอบ Object Storage ทำ storage แบบ Stripe บน XFS

3) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



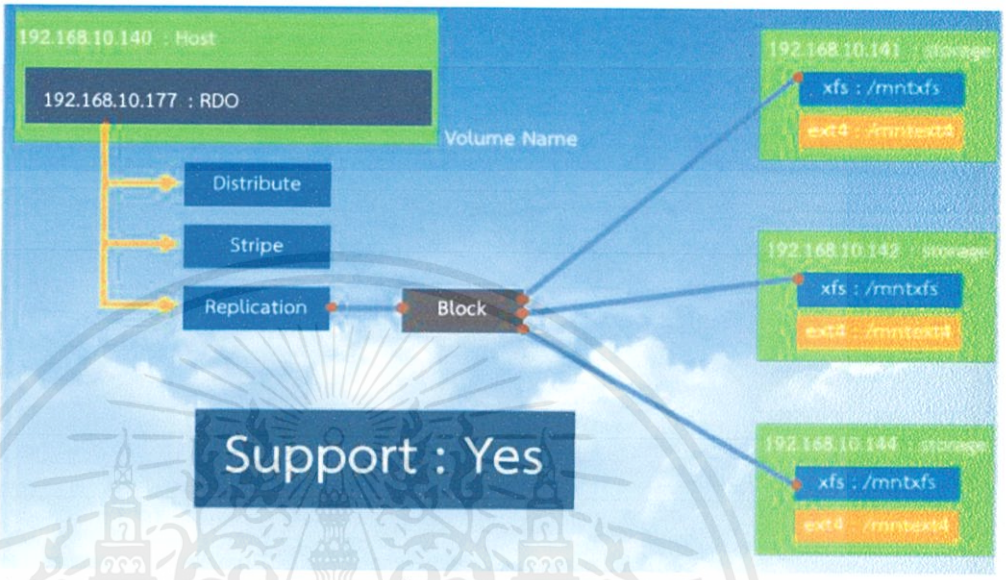
รูปที่ 4.39 การทดสอบ Object Storage ทำ storage แบบ Stripe บน XFS

4) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน XFS จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



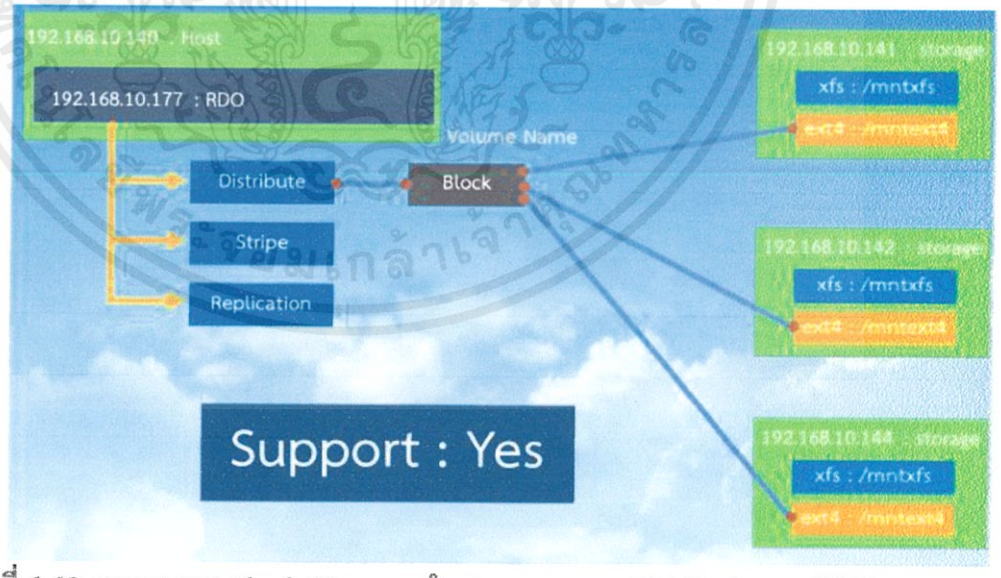
รูปที่ 4.40 การทดสอบ Block Storage ทำ storage แบบ Stripe บน XFS

5) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



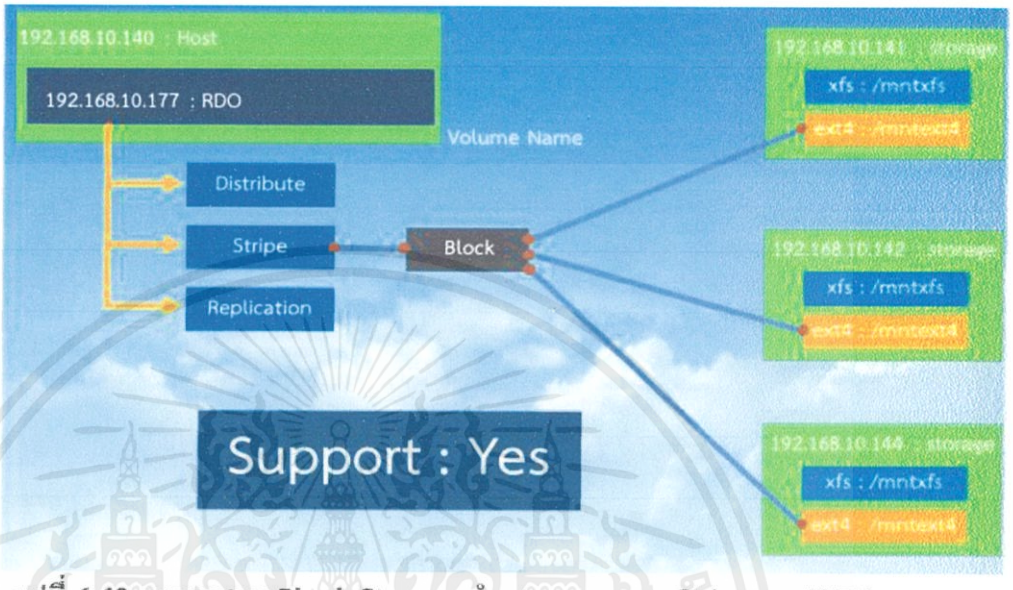
รูปที่ 4.41 การทดสอบ Block Storage ทำ storage แบบ Replication บน XFS

6) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



รูปที่ 4.42 การทดสอบ Block Storage ทำ storage แบบ Distribute บน EXT4

7) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน EXT4 จากนั้นสร้างไฟล์แบบ Block โดยใช้ ISCSI แล้วนำไปเก็บบน volume นั้น บันทึกผลการทดลอง



รูปที่ 4.43 การทดสอบ Block Storage ทำ storage แบบ Stripe บน EXT4

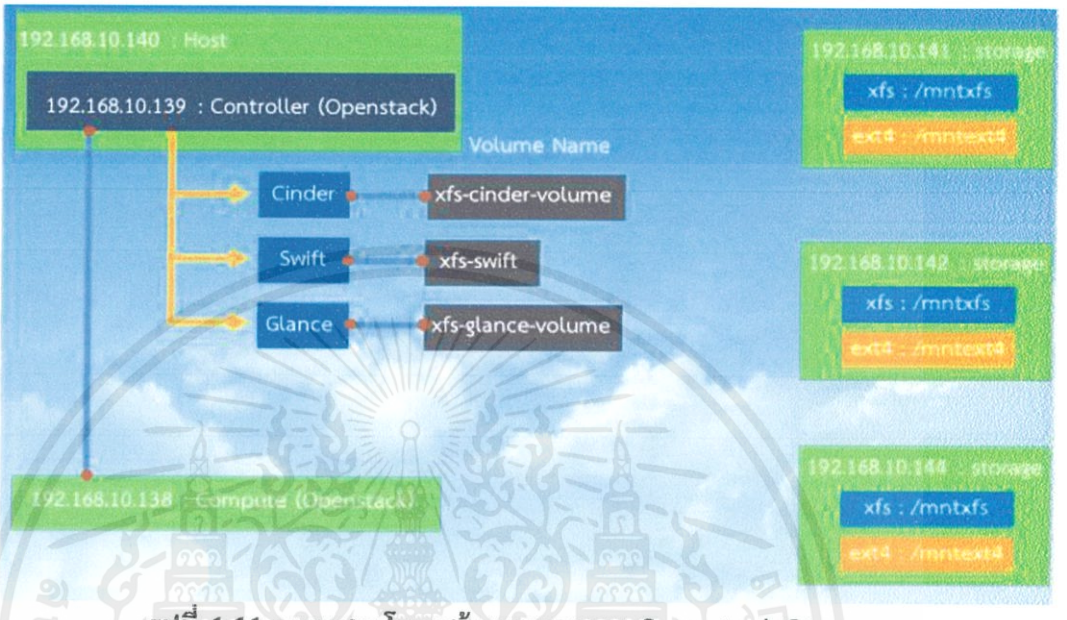
4.4.1 สรุปผลการทดลอง

Volume Types	XFS	EXT4
Distribute	Yes	Yes
Stripe	Yes	Yes
Replication	Yes	Yes

ตารางที่ 4.2 ผลการทดสอบ Block Storage – GlusterFS

4.5การทดสอบ Openstack Storage

1) เตรียมเครื่องคอมพิวเตอร์สำหรับทดลองดังนี้



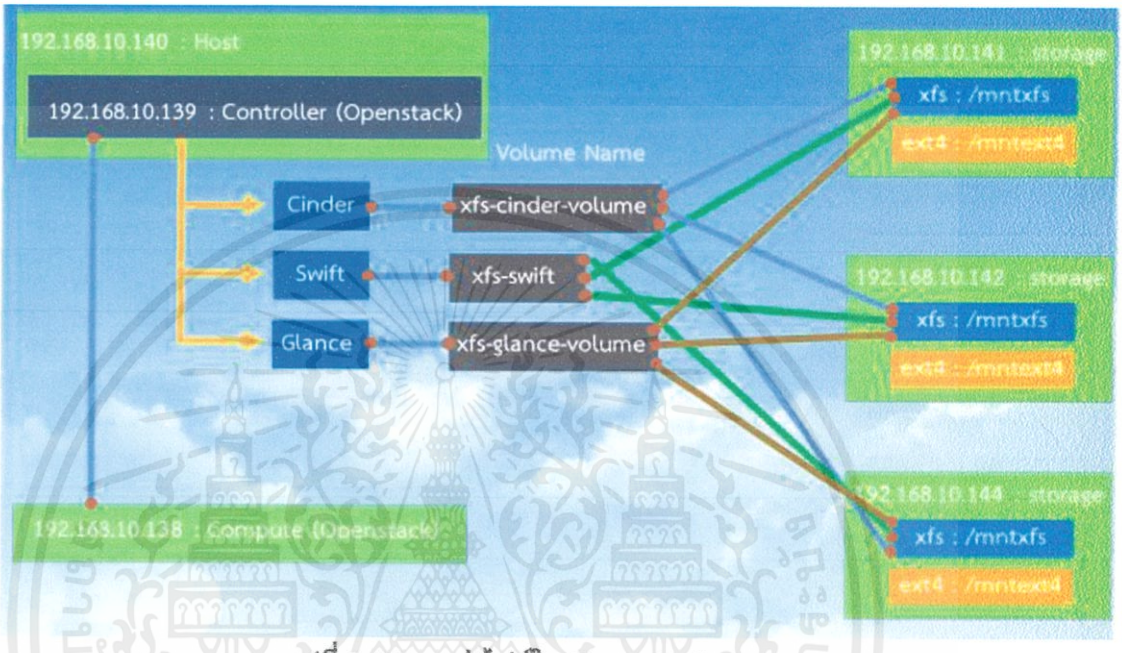
รูปที่ 4.44 ภาพแสดงโครงสร้างการทดสอบ Openstack Storage

- สร้างเครื่อง Host แล้วรันเครื่อง controller ของ openstack บน VM
- เตรียมเครื่องสำหรับทำเป็น Storage Node 3 เครื่อง ใช้ LVM แบ่งออกมา 2 Partition แล้วลง file system เป็น XFS และ EXT4 ทุกเครื่อง
- สร้างเครื่องสำหรับ compute ของ openstack แล้วเชื่อมเข้ากับเครื่อง controller

2) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน XFS สำหรับ Cinder
3) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน XFS สำหรับ Cinder
4) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน XFS สำหรับ Cinder
5) เชื่อมต่อ Cinder กับ volume แบบต่างๆ และทดลองใช้งาน แล้วบันทึกผลการทดลอง
6) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน XFS สำหรับ Swift
7) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน XFS สำหรับ Swift
8) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน XFS สำหรับ Swift
9) เชื่อมต่อ Swift กับ volume แบบต่างๆ และทดลองใช้งาน แล้วบันทึกผลการทดลอง
10) ใช้ GlusterFS สร้าง volume แบบ Distribute 3 brick บน XFS สำหรับ Glance
11) ใช้ GlusterFS สร้าง volume แบบ Stripe 3 brick บน XFS สำหรับ Glance

12) ใช้ GlusterFS สร้าง volume แบบ Replication 3 brick บน XFS สำหรับ Glance

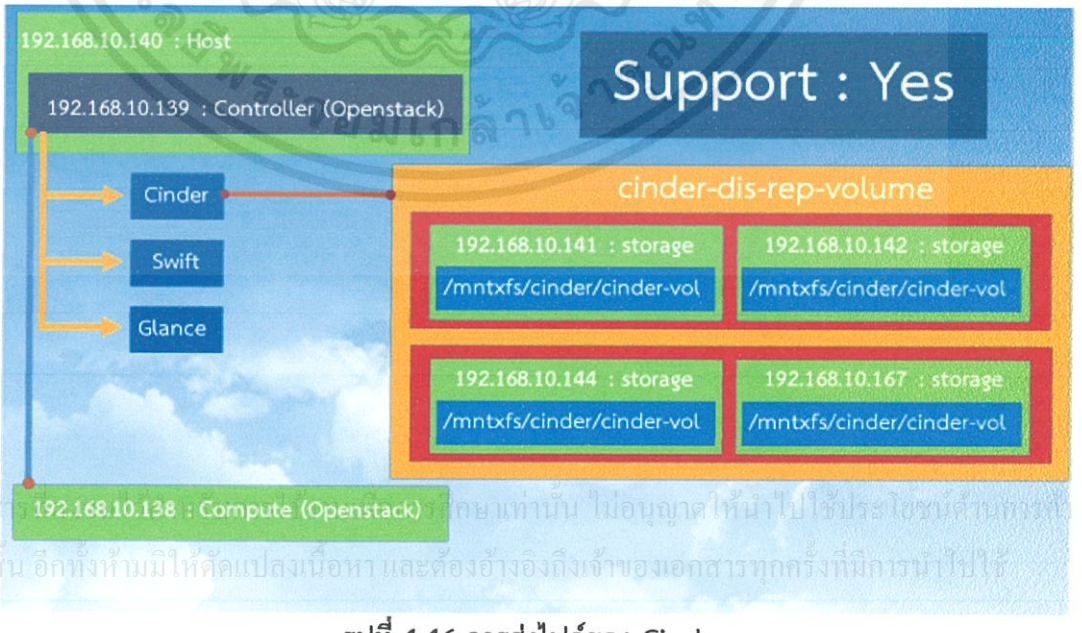
13) เชื่อมต่อ Glance กับ volume แบบต่างๆ และทดลองใช้งาน แล้วบันทึกผลการทดลอง



รูปที่ 4.45 การส่งไฟล์ใน Openstack storage

14) ใช้ GlusterFS สร้าง volume แบบ Distribute - Replication 4 brick บน XFS สำหรับ Cinder

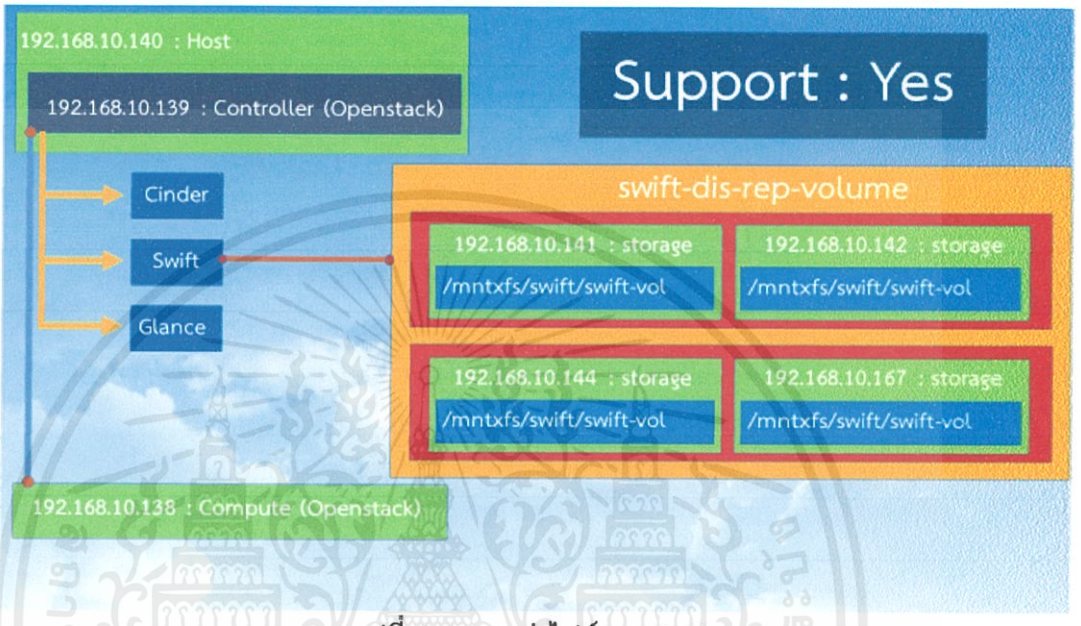
15) เชื่อมต่อ cinder กับ volume ที่สร้างและทดลองใช้งาน แล้วบันทึกผลการทดลอง



รูปที่ 4.46 การส่งไฟล์ของ Cinder

16) ใช้ GlusterFS สร้าง volume แบบ Distribute - Replication 4 brick บน XFS สำหรับ Swift

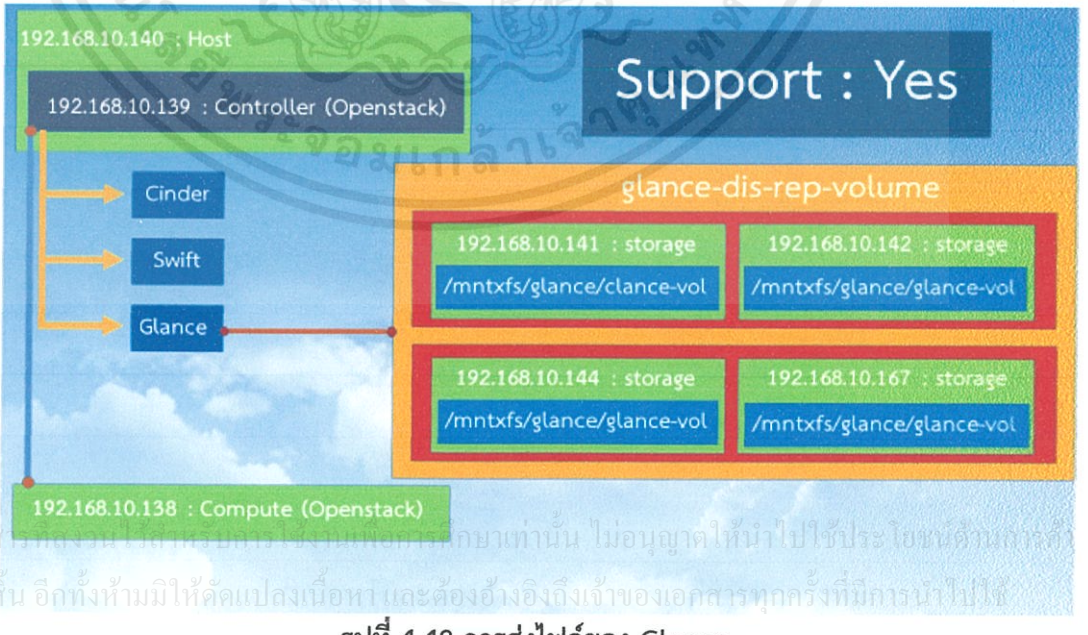
17) เชื่อมต่อ Swift กับ volume ที่สร้างและทดลองใช้งาน แล้วบันทึกผลการทดลอง



รูปที่ 4.47 การส่งไฟล์ของ Swift

18) ใช้ GlusterFS สร้าง volume แบบ Distribute - Replication 4 brick บน XFS สำหรับ Glance

19) เชื่อมต่อ Glance กับ volume ที่สร้างและทดลองใช้งาน แล้วบันทึกผลการทดลอง



รูปที่ 4.48 การส่งไฟล์ของ Glance

4.5.1 ผลการทดลอง

Cinder

Volume Types	Support
Distribute	Yes
Stripe	Yes
Replication	Yes
Distribute- Replication	Yes

ตารางที่ 4.3 ผลการทดสอบของ Cinder

Swift

Volume Types	Support
Distribute	Yes
Stripe	No
Replication	Yes
Distribute- Replication	Yes

ตารางที่ 4.4 ผลการทดสอบของ Swift

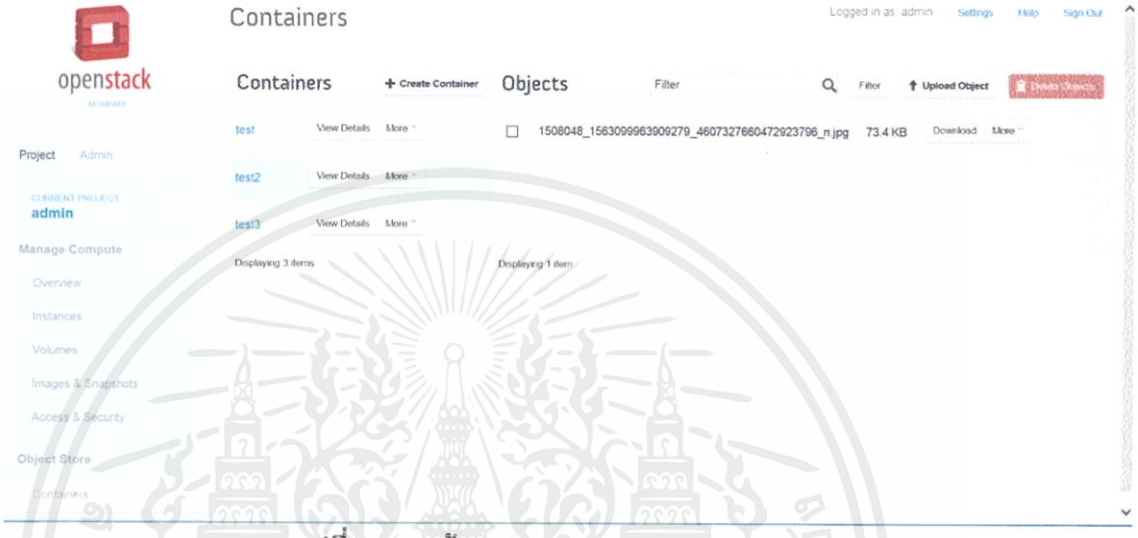
Glance

Volume Types	Support
Distribute	Yes
Stripe	Yes
Replication	Yes
Distribute- Replication	Yes

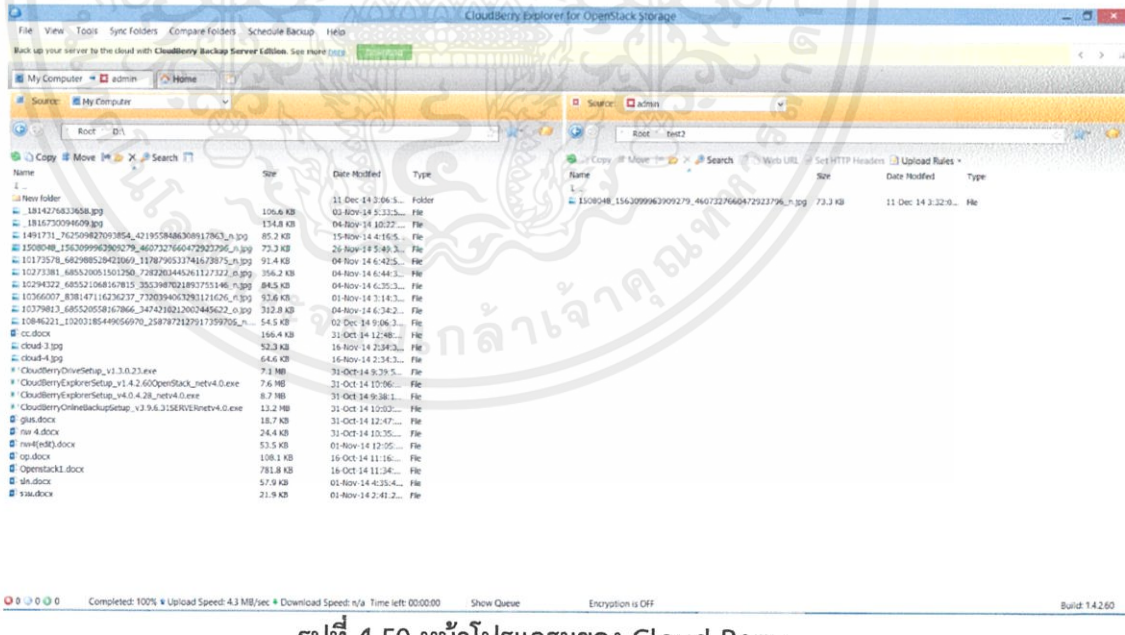
ตารางที่ 4.5 ผลการทดสอบของ Glance

4.6การใช้ Object Store บน Dashboard และ Cloud Berry

1) Overview

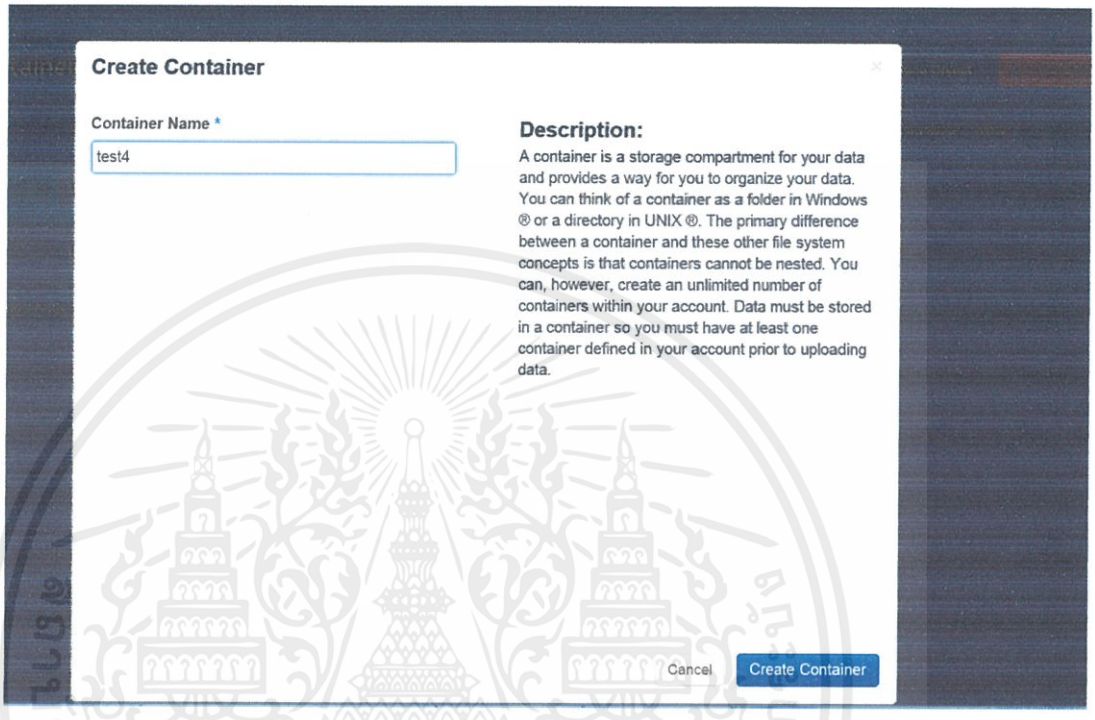


รูปที่ 4.49 หน้า Dashboard ของ Openstack

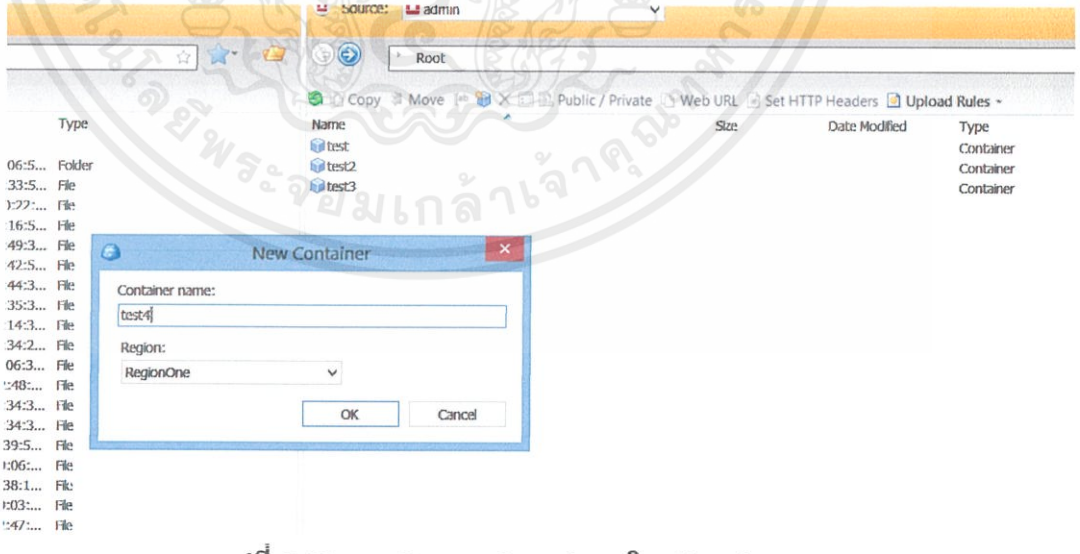


รูปที่ 4.50 หน้าโปรแกรมของ Cloud Berry

2) New Container

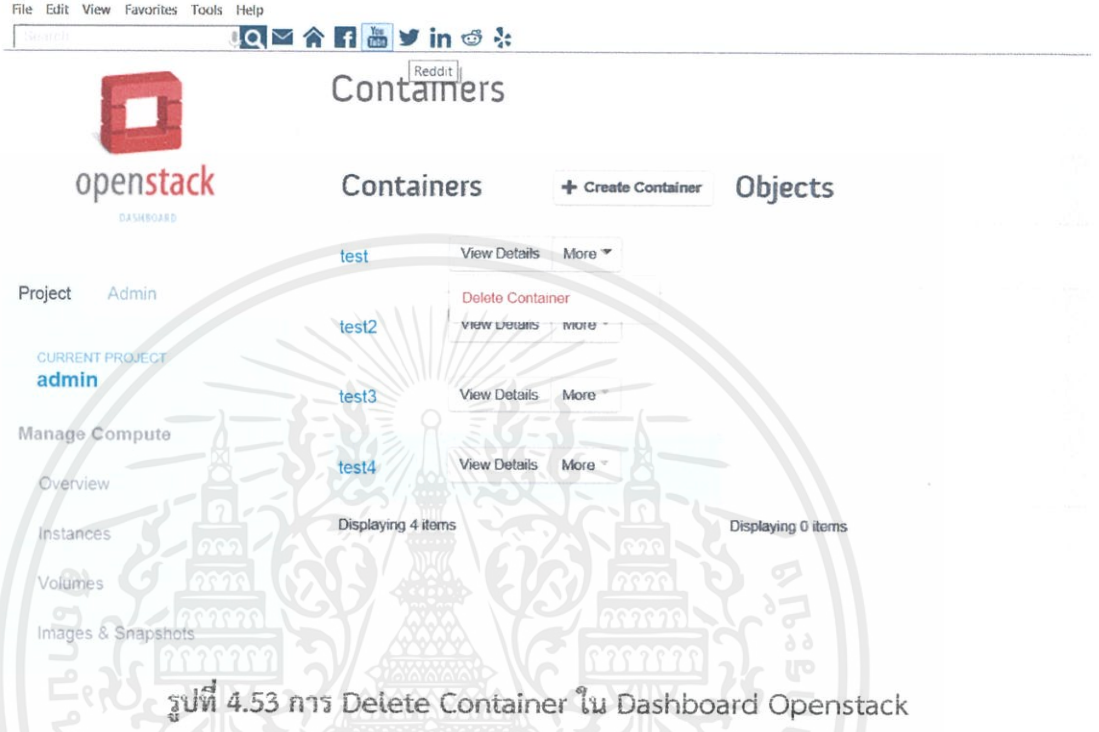


รูปที่ 4.51 การ Create Container ใน Dashboard Openstack

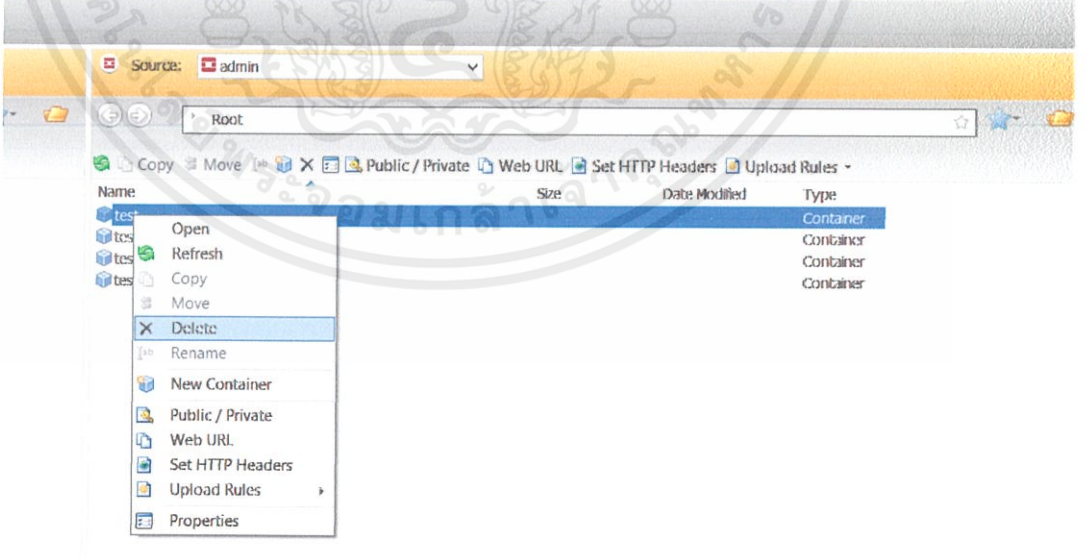


รูปที่ 4.52 การ Create Container ใน Cloud Berry

3) Delete Container

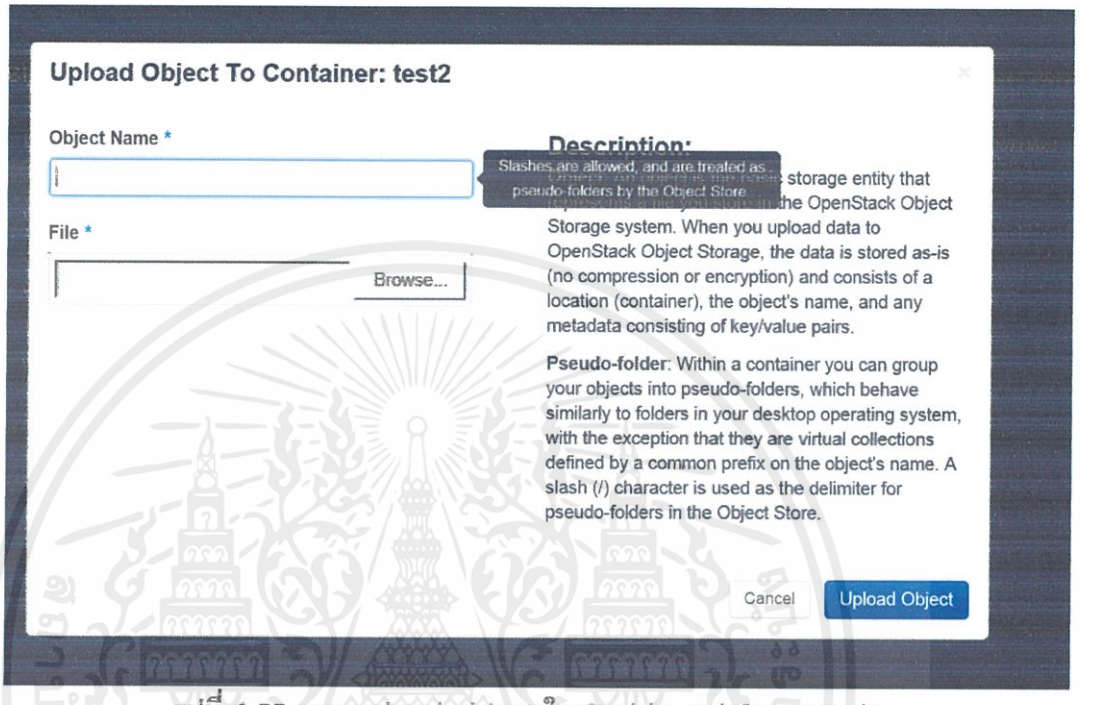


รูปที่ 4.53 การ Delete Container ใน Dashboard Openstack

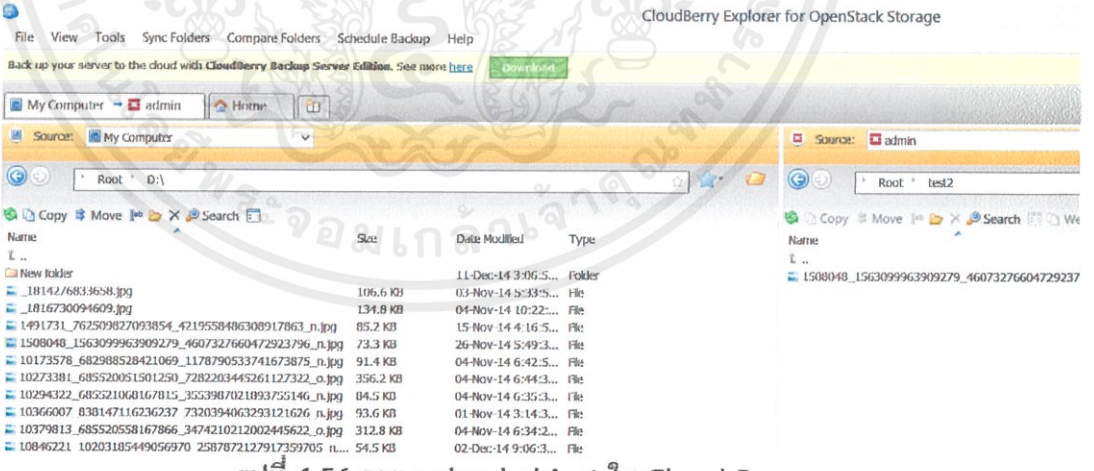


รูปที่ 4.54 การ Delete Container ใน Cloud Berry

4) Upload Object



รูปที่ 4.55 การ upload object ใน Dashboard Openstack



รูปที่ 4.56 การ upload object ใน Cloud Berry

4.7 Quick start Deployment using Packstack with Neutron and Ceilometer

4.7.1 การเตรียมซอฟแวร์

- อัพเดทแพคเกจปัจจุบันของเครื่องด้วยคำสั่ง yum update –y ตามรูปที่ 4.57

```
[root@controller ~]# yum update -y
```

รูปที่ 4.57 การอัพเดทแพคเกจให้เป็นปัจจุบัน

- ติดตั้ง RDO repositories ด้วยคำสั่ง yum install -y

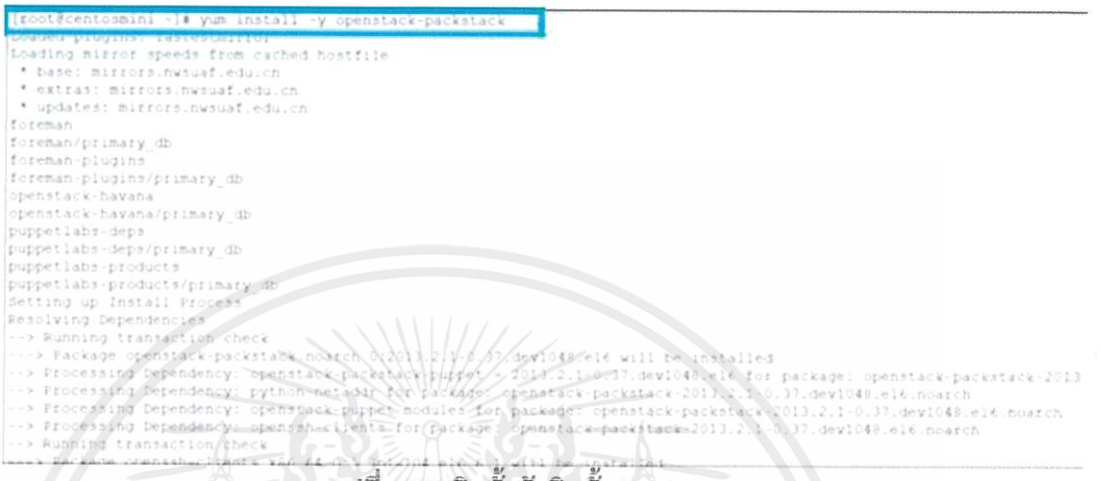
https://rdoproject.org/repos/rdo-release.rpm จะได้ผลลัพท์ตามรูปที่ 4.58

```
Loaded plugins: fastestmirror
Determining fastest mirrors
 * base: mirrors.nwsuaf.edu.cn
 * extras: mirrors.nwsuaf.edu.cn
 * updates: mirrors.nwsuaf.edu.cn
base                                                     | 3.7 kB     00:00
base/primary_db                                          | 4.6 MB     00:16
extras                                                   | 3.4 kB     00:00
extras/primary_db                                        |  30 kB     00:00
updates                                                  | 2.4 kB     00:00
updates/primary_db                                       | 1.5 MB     00:03
Setting up Install Process
rdo-release-havana-9.noarch.rpm                          |  13 kB     00:00
Examining /var/tmp/yum-root-aRDsx4/rdo-release-havana-9.noarch.rpm: rdo-release-
havana-9.noarch
Marking /var/tmp/yum-root-aRDsx4/rdo-release-havana-9.noarch.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package rdo-release.noarch 0:havana-9 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch        Version          Repository                    Size
================================================================================
Installing:
 rdo-release     noarch      havana-9         /rdo-release-havana-9.noarch  10 k

Transaction Summary
================================================================================
Install       1 Package(s)

Total size: 10 k
Installed size: 10 k
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : rdo-release-havana-9.noarch                                  1/1
  Verifying  : rdo-release-havana-9.noarch                                  1/1

Installed:
  rdo-release.noarch 0:havana-9

Complete!
```

รูปที่ 4.58 การติดตั้ง RDO repositories

4.7.2 ติดตั้งตัวติดตั้ง Packstack

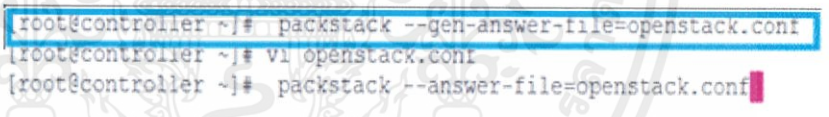
- ใช้คำสั่ง yum install -y openstack-packstack จะได้ผลลัพท์ดังรูปที่ 4.59



รูปที่ 4.59 ติดตั้งตัวติดตั้ง Packstack

4.7.3 สร้างไฟล์ PackStack

- ใช้คำสั่ง packstack --gen-answer-file=FILE (ชื่อไฟล์ที่ต้องการสร้าง) ตามตัวอย่างในรูปที่ 4.60



รูปที่ 4.60 สร้างไฟล์ PackStack

4.7.4 แก้ไขไฟล์ PackStack

- ใช้คำสั่ง vi FILE (ชื่อไฟล์ที่สร้างไว้) เพื่อเข้าไปแก้ไข ตามตัวอย่างในรูปที่ 4.61

รูปที่ 4.61 แก้ไขไฟล์ PackStack

- เพื่อติดตั้ง service ของ OpenStack ที่ต้องการใช้งาน จะต้องเปลี่ยนจาก n เป็น y ดังนี้ ตามรูปที่ 4.62

1) ติดตั้ง MariaDB
2) ติดตั้ง Glance

3) ติดตั้ง Cinder

4) ติดตั้ง Nova

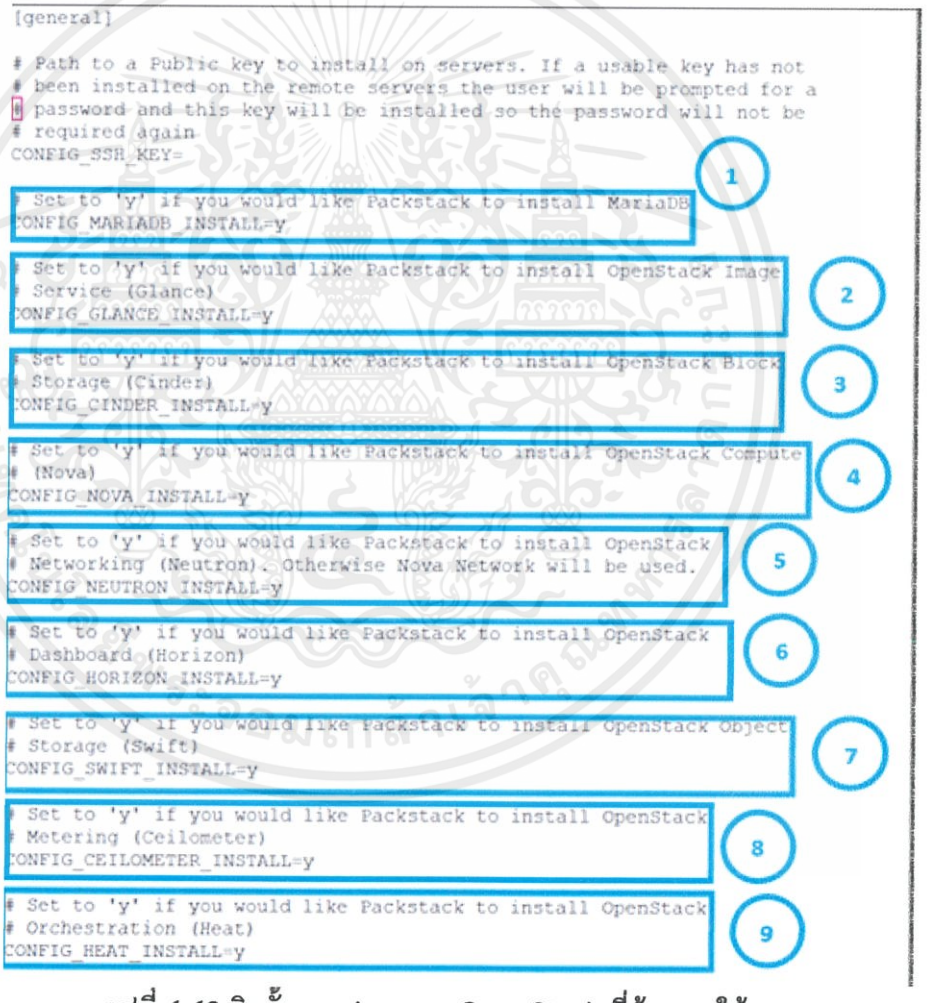
5) ติดตั้ง Neutron

6) ติดตั้ง Horizon

7) ติดตั้ง Swift

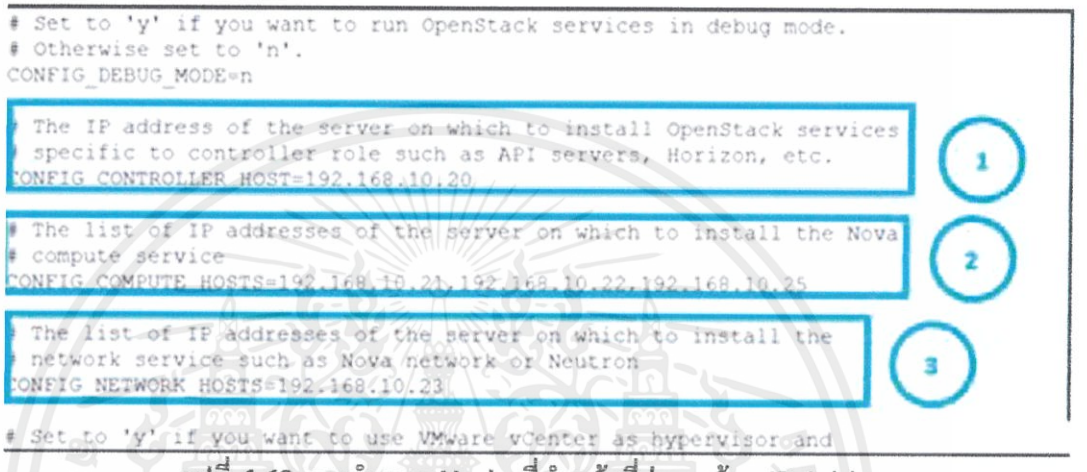
8) ติดตั้ง Ceilometer

9) ติดตั้ง Heat



รูปที่ 4.62 ติดตั้ง service ของ OpenStack ที่ต้องการใช้งาน

- กำหนด Node ที่ทำหน้าที่ต่างๆ ด้วย IP Address ตามรูปที่ 4.63
    1) Controller
    2) Compute
    3) Network



รูปที่ 4.63 การกำหนด Node ที่ทำหน้าที่ต่างๆ ด้วย IP Address

- การตั้งค่า Neutron แก้ไขดังนี้ ตามรูปที่ 4.64
    1) กำหนดชื่อ bridge สำหรับต่อกับเครือข่ายภายนอก โดยแก้ไขที่

       CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex
    2) กำหนด L2 plugin ที่ใช้กับ Neutron โดยแก้ไขที่

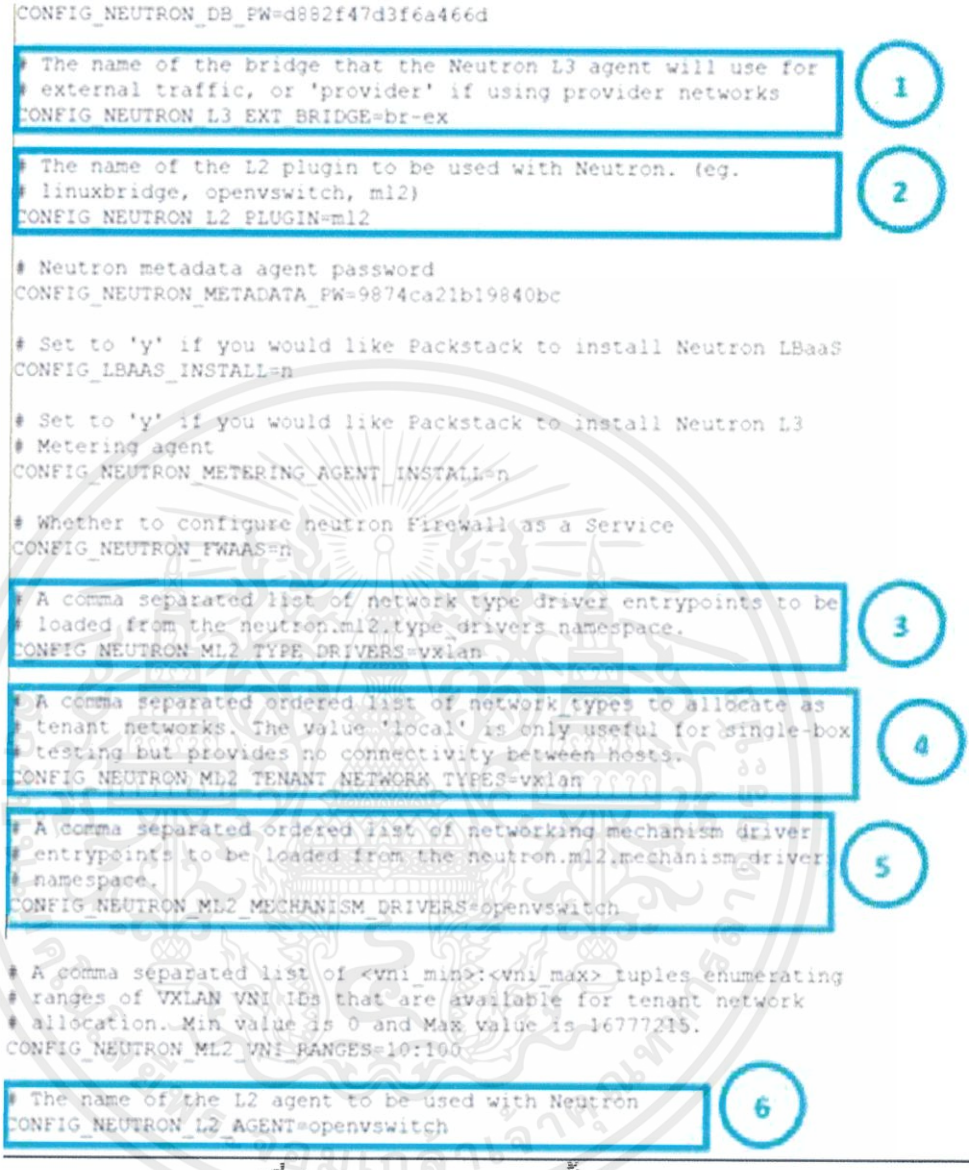
       CONFIG_NEUTRON_L2_PLUGIN=ml2
    3) เมื่อเลือก L2 plugin เป็น ml2 จะต้องมากำหนดประเภทของเครือข่ายของโปรแกรมควบคุม โดยแก้ไขที่

       CONFIG_NEUTRON_ML2_TYPE_DRIVERS=vxlan
    4) กำหนดประเภทเครือข่ายของผู้เช่า โดยแก้ไขที่

       CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vxlan
    5) กำหนด mechanism driver ที่จะนำมาใช้งาน โดยแก้ไขที่

       CONFIG_NEUTRON_ML2_MECHANISM _DRIVERS=openvswitch
    6) กำหนด L2 agent ที่จะนำมาใช้งาน โดยแก้ไขที่

       CONFIG_NEUTRON_L2_AGENT=openvswitch



รูปที่ 4.64 ภาพแสดงการการตั้งค่า Neutron

- บันทึกไฟล์ที่แก้ไขเรียบร้อยแล้ว

4.7.5 ดำเนินการ PackStack กับไฟล์ที่แก้ไขเรียบร้อยแล้ว

- ใช้คำสั่ง packstack --answer-file=FILE (ชื่อไฟล์ที่สร้างไว้) ตามรูปที่ 4.65

```
[root@controller ~]#  packstack --gen-answer-file=openstack.conf
[root@controller ~]# vi openstack.conf
[root@controller ~]#  packstack --answer-file=openstack.conf
```

รูปที่ 4.65 ภาพแสดงการใช้คำสั่ง packstack

- เมื่อ PackStack เรียบร้อยจะได้ดังรูปที่ 4.66

```
Applying 192.168.10.139_heat.pp
192.168.10.139_swift.pp:                                [ DONE ]
192.168.10.139_provision_demo.pp:                       [ DONE ]
192.168.10.139_provision_glance.pp:                     [ DONE ]
192.168.10.139_heat.pp:                                 [ DONE ]
Applying 192.168.10.139_mongodb.pp
192.168.10.139_mongodb.pp:                              [ DONE ]
Applying 192.168.10.139_ceilometer.pp
Applying 192.168.10.139_nagios.pp
Applying 192.168.10.137_nagios_nrpe.pp
Applying 192.168.10.138_nagios_nrpe.pp
Applying 192.168.10.139_nagios_nrpe.pp
192.168.10.138_nagios_nrpe.pp:                          [ DONE ]
192.168.10.137_nagios_nrpe.pp:                          [ DONE ]
192.168.10.139_ceilometer.pp:                           [ DONE ]
192.168.10.139_nagios.pp:                               [ DONE ]
192.168.10.139_nagios_nrpe.pp:                          [ DONE ]
Applying 192.168.10.137_postscript.pp
Applying 192.168.10.138_postscript.pp
Applying 192.168.10.139_postscript.pp
192.168.10.138_postscript.pp:                           [ DONE ]
192.168.10.137_postscript.pp:                           [ DONE ]
192.168.10.139_postscript.pp:                           [ DONE ]
Applying Puppet manifests                               [ DONE ]
Finalizing                                              [ DONE ]

 **** Installation completed successfully ******
```

รูปที่ 4.66 ภาพแสดงเมื่อ PackStack เรียบร้อย

4.7.6 ทำการ Bridge ระหว่าง br-ex กับ eth0 (ทำที่ Network Node)

- แก้ไขไฟล์ eth0 ดังนี้ ตามรูปที่ 4.67

```
DEVICE=eth0
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

รูปที่ 4.67 ภาพแสดงการแก้ไขไฟล์ eth0

- สร้างไฟล์ br-ex ดังนี้ ตามรูปที่ 4.68

```
DEVICE=br-ex
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=192.168.10.23 # Old eth0 IP since we want the network restart to not kill the connec
tion, otherwise pick something outside your dhcp range
NETMASK=255.255.255.0  # your netmask
GATEWAY=192.168.10.10  # your gateway
DNS1=192.168.10.10     # your nameserver
ONBOOT=yes
```

รูปที่ 4.68 สร้างไฟล์ br-ex

- ทำการ bridge โดยใช้คำสั่งตอไปนี้
  1) ovs-vsctl show (เพื่อตรวจดูการเชื่อมต่อของ Open Vswitch)
  2) ovs-vsctl add-port br-ex eth0 (เพื่อเชื่อมต่อ br-ex กับ eth0)
  3) service network restart (เพื่อทำการ restart เครือข่ายใหม่)

4.8 Openstack integration with Open Daylight controller

4.8.1 ติดตั้ง OpenDaylight

ที่โหนด OpenDaylight

- ดาวน์โหลด OpenDaylight
- ยกเลิกการบีบอัดไฟล์ด้วย root ตามรูปที่ 4.69

```
[root@daylight ~]# tar xvfz distribution-karaf-0.2.3-Helium-SR3.tar.gz
```

รูปที่ 4.69 ภาพแสดงการบีบอัดไฟล์

- เริ่มต้น OpenDaylight ด้วยคำสั่งตามรูปที่ 4.70

```
distribution-karaf-0.2.3-Helium-SR3/bin/shell.bat
distribution-karaf-0.2.3-Helium-SR3/bin/start.bat
distribution-karaf-0.2.3-Helium-SR3/bin/status.bat
distribution-karaf-0.2.3-Helium-SR3/bin/stop.bat
[root@daylight ~]# cd distribution-karaf-0.2.3-Helium-SR3
[root@daylight distribution-karaf-0.2.3-Helium-SR3]# ./bin/start # Start OpenDaylight as a
server process
[root@daylight distribution-karaf-0.2.3-Helium-SR3]#
```

รูปที่ 4.70 ภาพแสดงการเริ่มต้น OpenDaylight

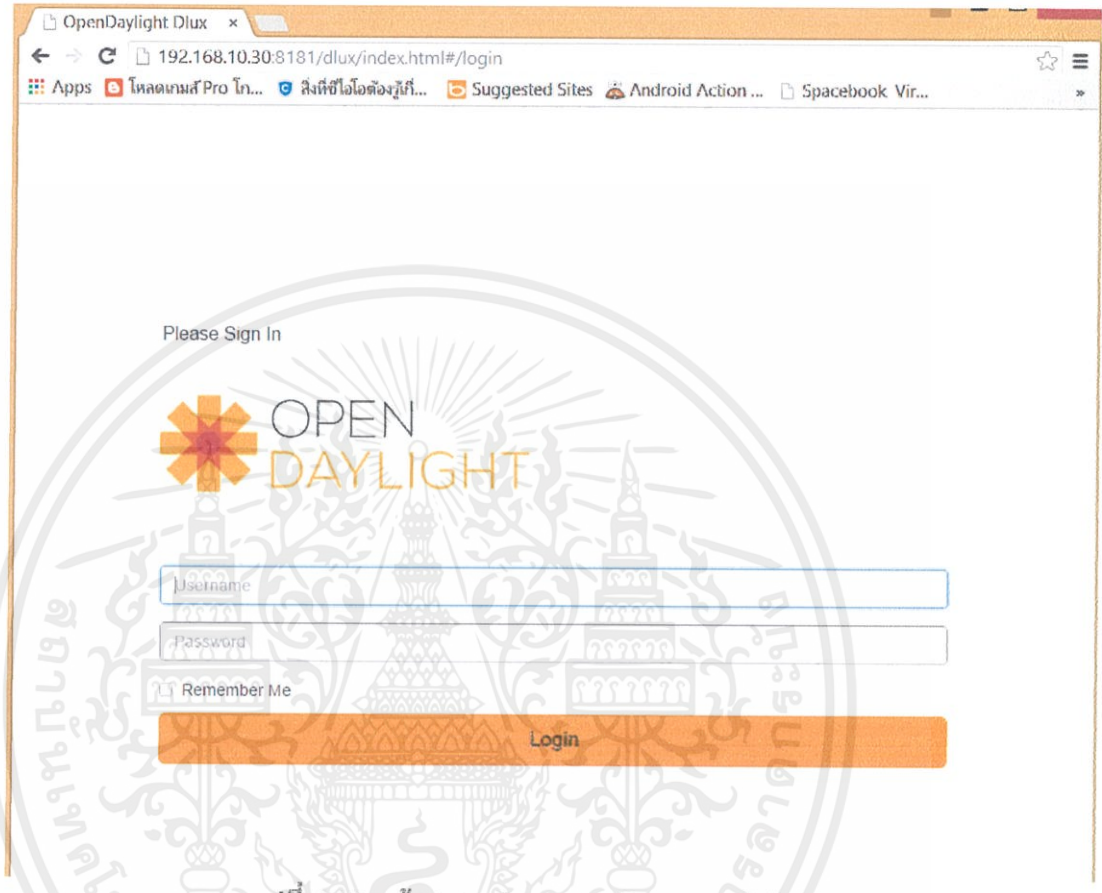
○ เชื่อมต่อไปยังคอนโซล ด้วยคำสั่งตามรูปที่ 4.71

รูปที่ 4.71 คำสั่งการเชื่อมต่อกับคอนโซลของ OpenDaylight

○ ตอนนี้มีการเชื่อมต่อกับคอนโซลของ OpenDaylight ต่อมาติดตั้งคุณสมบัติทั้งหมดที่จำเป็นต้องใช้ ด้วยคำสั่งตามรูปที่ 4.72

รูปที่ 4.72 ติดตั้งคุณสมบัติทั้งหมดที่จำเป็นต้องใช้

- หากทุกอย่างถูกติดตั้งอย่างถูกต้อง ตอนนี้จะสามารถเข้าสู่ระบบอินเตอร์เฟซบน dlux ตามรูปที่ 4.73



รูปที่ 4.73 หน้า Interface ของ OpenDaylight

ที่โหนด Controller

- ลบ instacnes, networks, routers และ ports ทั้งหมด
- หยุดบริการนิวตรอนเซิร์ฟเวอร์ระหว่างการตั้งค่า ด้วยคำสั่ง service neutron-server stop
- ที่โหลด Network และโหนด Compute
- หยุดบริการ Open vSwitch และเคลีย OVSDB ที่มีอยู่ ด้วยคำสั่งตามรูปที่ 4.74

```
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
tuned           0:off   1:off   2:off   3:on    4:on    5:on    6:off
udev-post       0:off   1:on    2:on    3:on    4:on    5:on    6:off
[root@compute01 ~]# service openvswitch stop
Killing ovs-vswitchd (12936)                               [  OK  ]
Killing ovsdb-server (12926)                               [  OK  ]
[root@compute01 ~]# rm -rf /var/log/openvswitch/*
[root@compute01 ~]# rm -rf /etc/openvswitch/conf.db
[root@compute01 ~]#  service openvswitch start
/etc/openvswitch/conf.db does not exist ... (warning).
Creating empty database /etc/openvswitch/conf.db           [  OK  ]
Starting ovsdb-server                                      [  OK  ]
Configuring Open vSwitch system IDs                        [  OK  ]
Starting ovs-vswitchd                                      [  OK  ]
Enabling remote OVSDB managers                             [  OK  ]
[root@compute01 ~]#
```

รูปที่ 4.74 การหยุดบริการ Open vSwitch และเคลีย OVSDB ที่มีอยู่

○ ในขั้นตอนนี้การกำหนดค่า Open vSwitch ของเปิดควรจะว่างเปล่า ตามรูปที่ 4.75

```
[root@compute01 ~]# ovs-vsctl show
12b74aca-3e26-445b-93c5-e0a9469f790d
    ovs_version: "2.1.3"
[root@compute01 ~]#
```

รูปที่ 4.75 ภาพแสดงการกำหนดค่า Open vSwitch

○ กำหนดค่า tunnel end-points ด้วยคำสั่งตามรูปที่ 4.76

```
[root@computer02 ~]# ovs-vsctl show
7542abf7-798b-4e4b-a70d-2aec06749303
    ovs_version: "2.1.3"
[root@computer02 ~]# ovs-vsctl set Open_vSwitch 7542abf7-798b-4e4b-a70d-2aec06749303 other_
config={'local_ip'='10.0.2.2'}
[root@computer02 ~]# ovs-vsctl list Open_vSwitch
_uuid               : 7542abf7-798b-4e4b-a70d-2aec06749303
bridges             : []
cur_cfg             : 1
db_version          : "7.4.0"
external_ids        : {system-id="255e040d-5894-477b-9513-3047208c2fc0"}
manager_options     : []
next_cfg            : 1
other_config        : {local_ip="10.0.2.2"}
ovs_version         : "2.1.3"
ssl                 : []
statistics          : {}
system_type         : unknown
system_version      : unknown
[root@computer02 ~]# ovs-vsctl set-manager tcp:192.168.10.30:6640
[root@computer02 ~]#
```

รูปที่ 4.76 ภาพแสดงการกำหนดค่า tunnel end-points

- สร้าง bridge br-ex ที่จำเป็นสำหรับเครือข่ายภายนอกเพื่อ OpenStack (ทำเฉพาะที่โหนด Network) ด้วยคำสั่งตามรูปที่ 4.77

```
[root@networkNode ~]# service openvswitch-switch stop
openvswitch-switch: unrecognized service
[root@networkNode ~]# clear
[root@networkNode ~]# service openvswitch stop
Killing ovs-vswitchd (17834)                               [  OK  ]
Killing ovsdb-server (17824)                               [  OK  ]
[root@networkNode ~]# rm -rf /var/log/openvswitch/*
[root@networkNode ~]# rm -rf /etc/openvswitch/conf.db
[root@networkNode ~]# service openvswitch start
/etc/openvswitch/conf.db does not exist ... (warning).
Creating empty database /etc/openvswitch/conf.db           [  OK  ]
Starting ovsdb-server                                      [  OK  ]
Configuring Open vSwitch system IDs                        [  OK  ]
Starting ovs-vswitchd                                      [  OK  ]
Enabling remote OVSDB managers                             [  OK  ]
[root@networkNode ~]# ovs-vsctl show
de6c9b28-7b72-4eff-8416-9989bd77685f
    ovs_version: "2.1.3"
[root@networkNode ~]# ovs-vsctl set Open_vSwitch de6c9b28-7b72-4eff-8416-9989bd77685f other
_config={'local_ip'='10.0.2.3'}
[root@networkNode ~]# ovs-vsctl add-br br-ex
[root@networkNode ~]# ovs-vsctl add-port br-ex eth0
[root@networkNode ~]# ovs-vsctl set-manager tcp:192.168.10.30:6640
```

รูปที่ 4.77 ภาพแสดงการสร้าง bridge br-ex

- เชื่อมต่อทุก openvswitch ด้วยตัวควบคุม OpenDaylight ด้วยคำสั่งตามรูปที่ 4.78

```
system_type         : unknown
system_version      : unknown
[root@computer02 ~]# ovs-vsctl set-manager tcp:192.168.10.30:6640
[root@computer02 ~]#
```

รูปที่ 4.78 ภาพแสดงการเชื่อมต่อทุก openvswitch ด้วยตัวควบคุม OpenDaylight

ที่โหนด Controller

- แก้ไขไฟล์ /etc/neutron/plugins/ml2/ml2_conf.ini และใส่การตั้งค่าตามรูปที่ 4.79

```
[ml2]
# (ListOpt) List of network type driver entrypoints to be loaded from
# the neutron.ml2.type_drivers namespace.
#
# type_drivers = local,flat,vlan,gre,vxlan
type_drivers = vxlan
# Example: type_drivers = flat,vlan,gre,vxlan

# (ListOpt) Ordered list of network_types to allocate as tenant
# networks. The default value 'local' is useful for single-box testing
# but provides no connectivity between hosts.
#
# tenant_network_types = local
tenant_network_types = vxlan
# Example: tenant_network_types = vlan,gre,vxlan

# (ListOpt) Ordered list of networking mechanism driver entrypoints
# to be loaded from the neutron.ml2.mechanism_drivers namespace.
# mechanism_drivers =
mechanism_drivers =opendaylight
# Example: mechanism_drivers = openvswitch,mlnx
# Example: mechanism_drivers = arista
# Example: mechanism_drivers = cisco,logger
/ tenant_network_types
# Configuration for the OpenDaylight MechanismDriver

[ml2_odl]
# (StrOpt) OpenDaylight REST URL
# If this is not set then no HTTP requests will be made.
#
url = http://192.168.56.1:8081/controller/nb/v2/neutron
# Example: url = http://192.168.56.1:8081/controller/nb/v2/neutron

# (StrOpt) Username for HTTP basic authentication to ODL.
#
username = admin
# Example: username = admin

# (StrOpt) Password for HTTP basic authentication to ODL.
#
password = admin
# Example: password = admin

# (IntOpt) Timeout in seconds to wait for ODL HTTP request completion.
# This is an optional parameter, default value is 10 seconds.
#
# timeout = 10
```

รูปที่ 4.79 ภาพแสดงการแก้ไขไฟล์

/etc/neutron/plugins/ml2/ml2_conf.ini

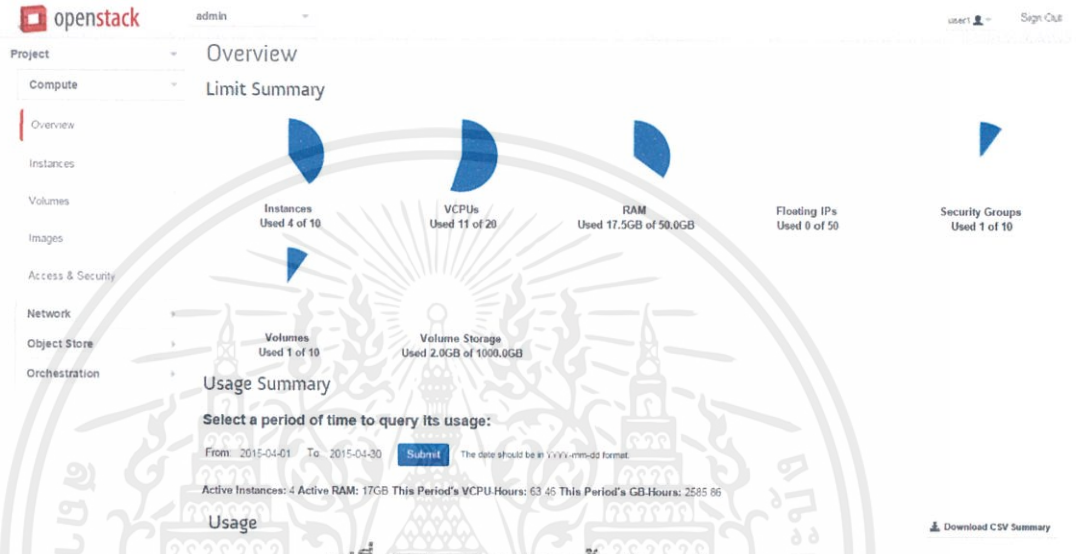
- Restart neutron-server ด้วยคำสั่ง service neutron-server start

4.9 การใช้งานระบบการประมวลผลแบบกลุ่มเมฆ

4.9.1 ส่วน Users Interface (Dashboard)

Overview

จะเป็นหน้าเว็บที่บอกการใช้งานของทุกโปรเจคที่อยู่บนคลาวด์ว่าใช้งานไปเท่าไรบ้าง



รูปที่ 4.80 ภาพแสดงหน้า Overview

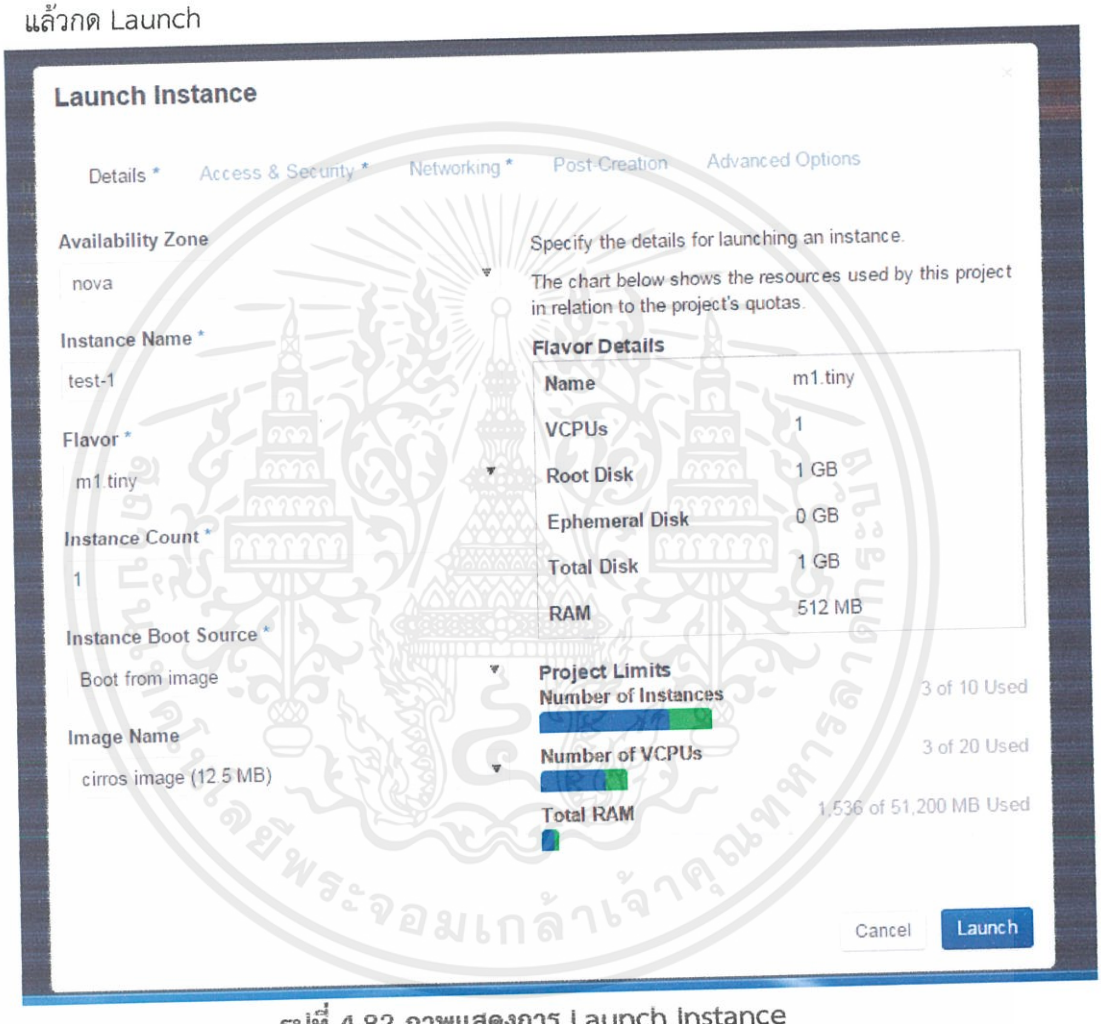
Instance

เป็นหน้าเว็บที่จะแสดง,launch,สร้างsnapshot,หยุดและ รีบูต instance

รูปที่ 4.81 ภาพแสดงหน้า Instance

วิธีการ Launch Instance

1) คลิกที่ luanch instance ในหน้าInstance จะขึ้นหน้าต่างให้กรอกรายละเอียดว่าจะให้อยู่ในโซนไหน มีชื่ออะไร มีลักษณะinstanceแบบไหน ให้บูทจากไหน และใช้ Image อะไร แล้วกด Launch



รูปที่ 4.82 ภาพแสดงการ Launch Instance

2) ก็จะได้Instanceที่ทำการlaunch ขึ้นมา

รูปที่ 4.83 ภาพแสดง Instance ที่ถูก launch

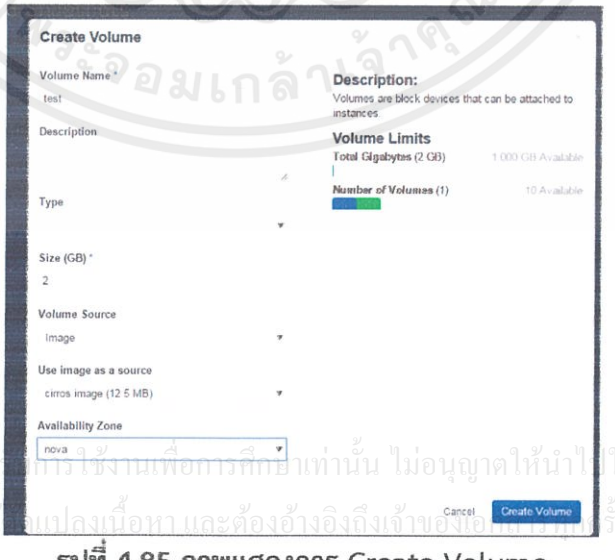
Volume

เป็นหน้าเว็บที่แสดง volumeที่ถูกสร้าง และสามารถสร้างvolume แก้ไขและลบ volumeได้

รูปที่ 4.84 ภาพแสดงหน้า Volume

วิธีการสร้าง Volume

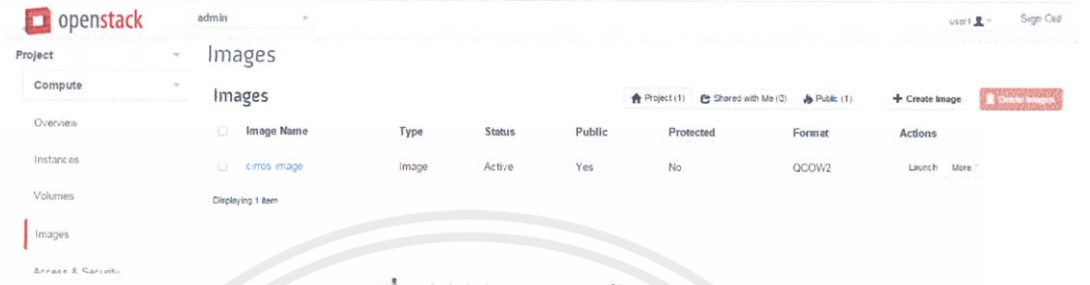
ให้คลิกที่ Create Volumes ในหน้า Volume ก็จะขึ้นหน้าต่างให้กรอกรายละเอียดว่าให้มีชื่อ volume อะไร มีขนาดเท่าไร และกำหนดว่า volume จะสร้างมาจากที่ไหน และประเภทของ Image และบอก Availability Zone ที่จะให้ไปอยู่



รูปที่ 4.85 ภาพแสดงการ Create Volume

Images

เป็นหน้าที่จะแสดง image และ image snapshot ที่ถูกสร้างโดย project user และสามารถที่จะสร้าง แก้ไขและลบ imageได้



รูปที่ 4.86 ภาพแสดงหน้า Images

Acess&Security

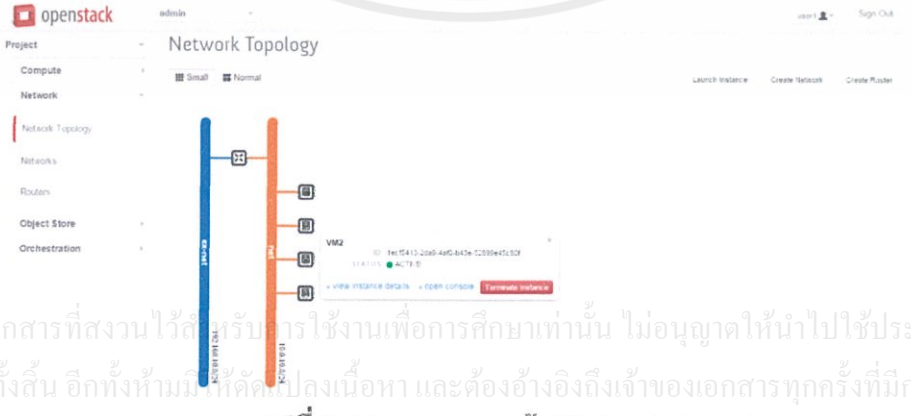
Security Group

เป็นในส่วนของการเปิดและปิดพอร์ตที่ต้องการเช่น การเปิดพอร์ต 22 (ssh)

รูปที่ 4.87 ภาพแสดงหน้า Security Group

Network Topology

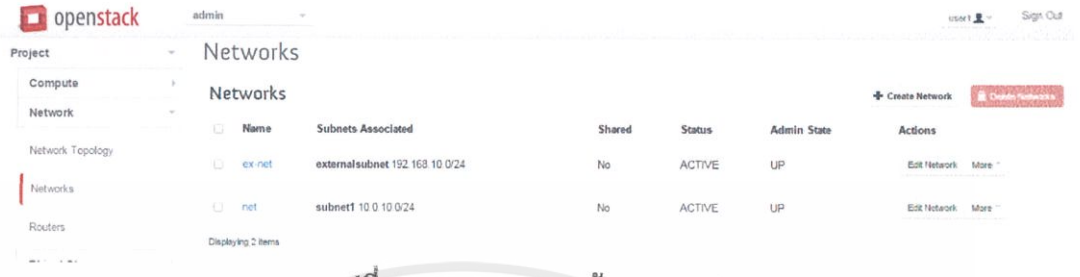
ส่วนแท่งสีส้มเป็นส่วนของไพรเวทเน็ตเวิร์ค ส่วนสีฟ้าเป็นส่วนของพับบลิคเน็ตเวิร์คและมี router1 เชื่อมต่อกับเน็ตเวิร์คทั้งสองเน็ตเวิร์ค และ สามาถสร้าง network และ router ได้ในหน้านี้



รูปที่ 4.88 ภาพแสดงหน้า Network Topology

Networks

โดยหน้าเว็บนี้จะสามารถที่จะสร้าง ลบ และจัดการที่เกี่ยวกับ network



รูปที่ 4.89 ภาพแสดงหน้า Networks

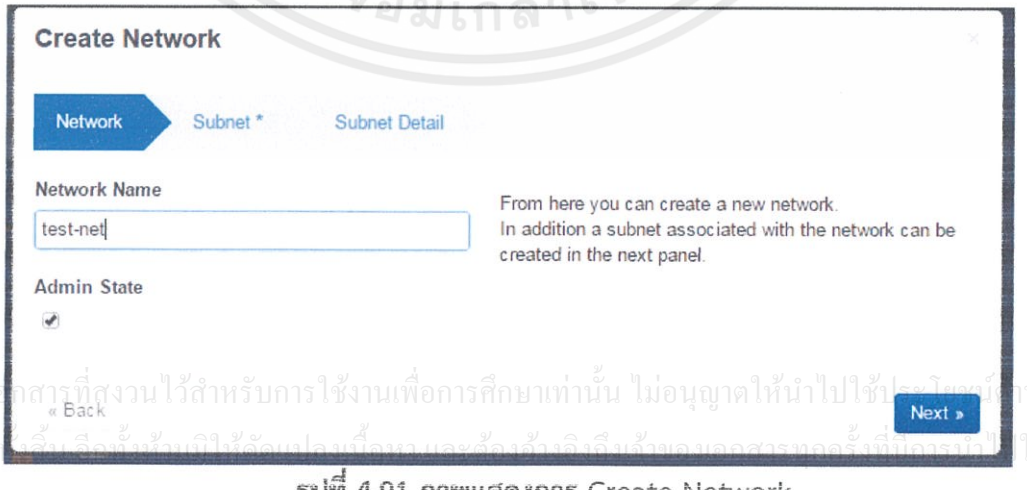
Routers

โดยหน้าเว็บนี้จะสามารถที่จะสร้าง ลบ และจัดการที่เกี่ยวกับ router

รูปที่ 4.90 ภาพแสดงหน้า Routers

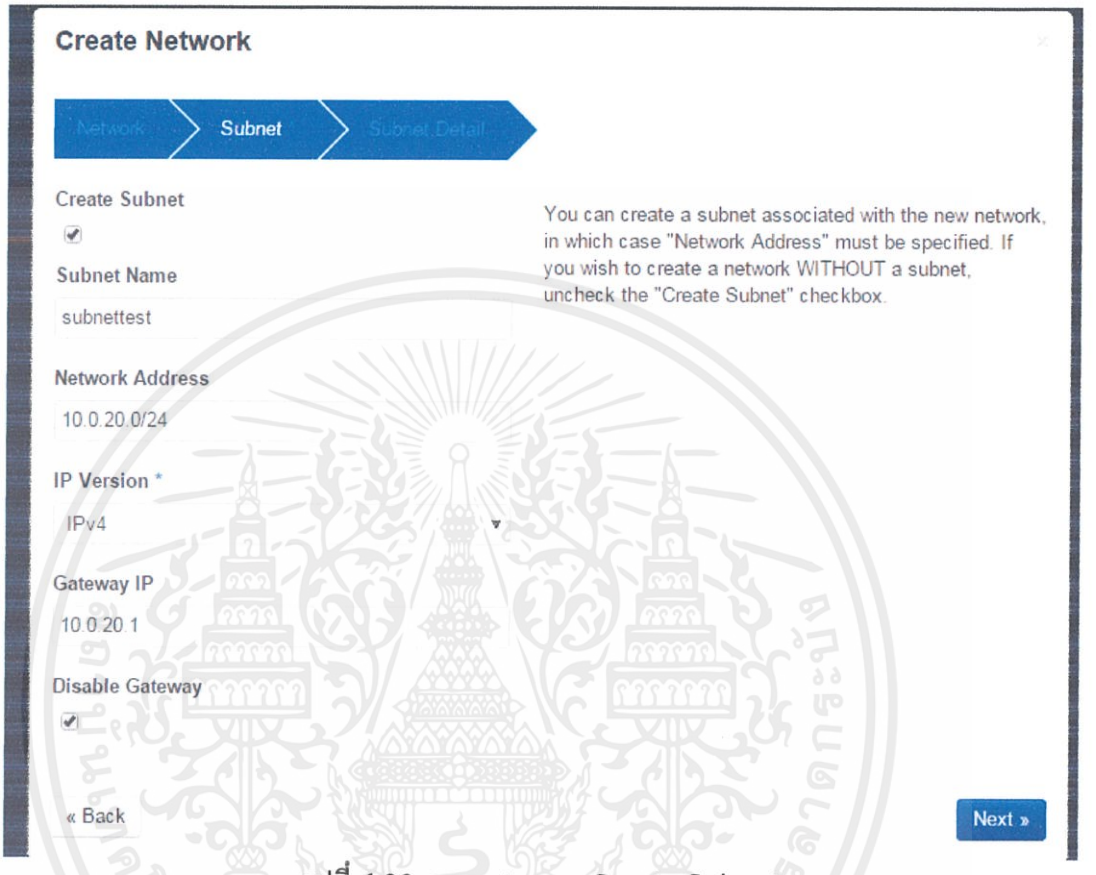
วิธีการสร้าง Network

1) ผู้ใช้สามารถคลิกปุ่ม Create Network ได้ที่หน้าของ Network Topology หรือ Networks โดยเมื่อคลิกเสร็จแล้วจะขึ้นหน้าต่างให้กรอกรายละเอียดว่า ชื่อ network นี้จะชื่ออะไร แล้วกด next



รูปที่ 4.91 ภาพแสดงการ Create Network

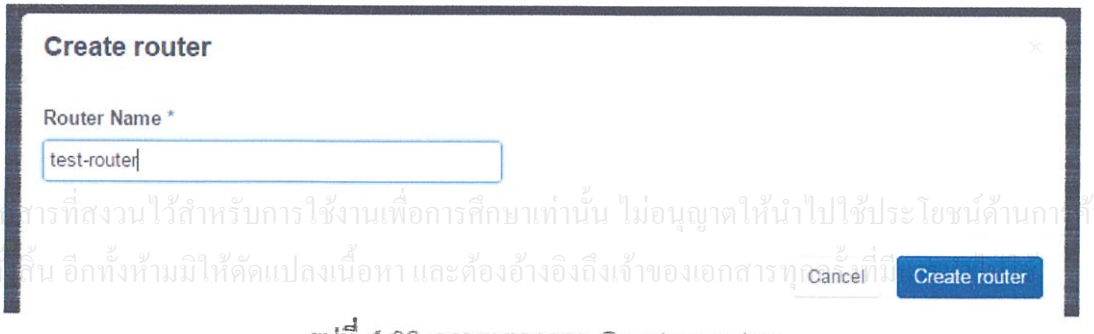
2) หน้าต่างนี้ก็จะให้กรอกรายละเอียดเกี่ยวกับ Subnet ว่าจะให้ชื่ออะไร มี network address ช่วงไหนเป็น IPversionไหน แล้วก็ระบุ Gateway IP



รูปที่ 4.92 ภาพแสดงการ Create Subnet

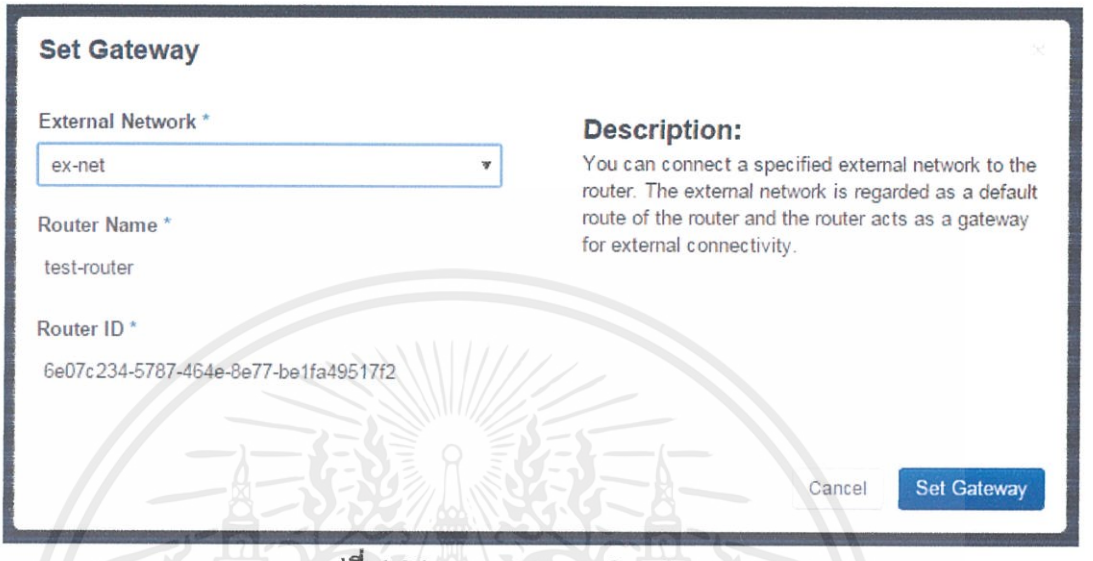
วิธีการสร้าง Routers

1) ผู้ใช้สามารถคลิกปุ่ม Create Router ได้ที่หน้าของ Network Topology หรือ Routers โดยเมื่อคลิกเสร็จแล้วจะขึ้นหน้าต่างให้กรอกรายละเอียดว่า ชื่อ Router นี้จะชื่ออะไร แล้วกด next



รูปที่ 4.93 ภาพแสดงการ Create router

2) หน้าต่างนี้จะให้กรอก Gateway ของ ว่าจะให้ External Network ออกไปวงไหนของ network



รูปที่ 4.94 ภาพแสดงการ Set Gateway

3) หน้าต่างนี้จะให้ทำการ add interface ว่าจะให้เชื่อมต่อกับ network ไหน

รูปที่ 4.95 ภาพแสดงการ add interface

4.9.2 ส่วน Admin Interface (Dashboard)

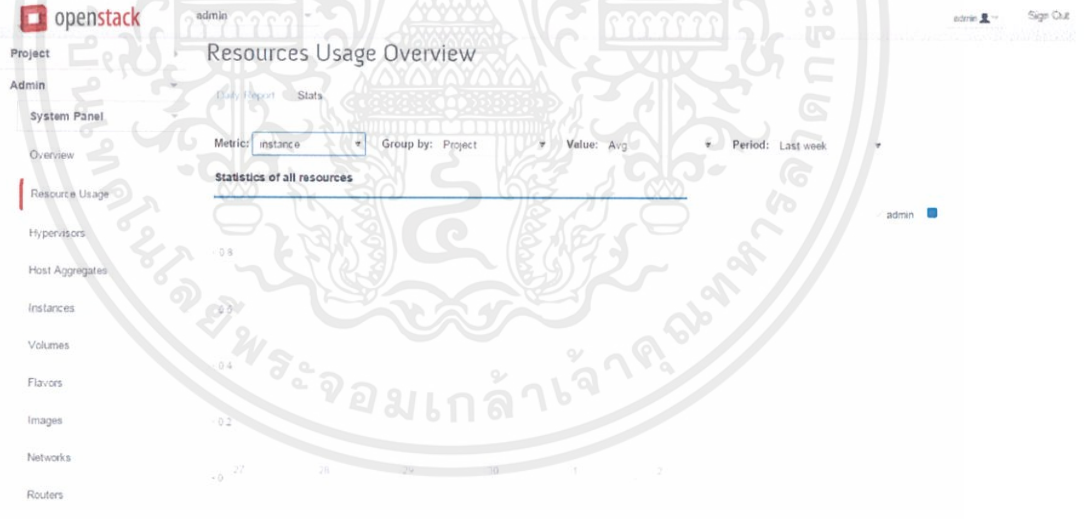
Overview

เป็นหน้าเว็บที่แสดงการใช้งานของทุก project ที่อยู่บนคลาวด์ ตามชื่อของ projectที่มี

รูปที่ 4.96 ภาพแสดงหน้า Overview

Resource Usage

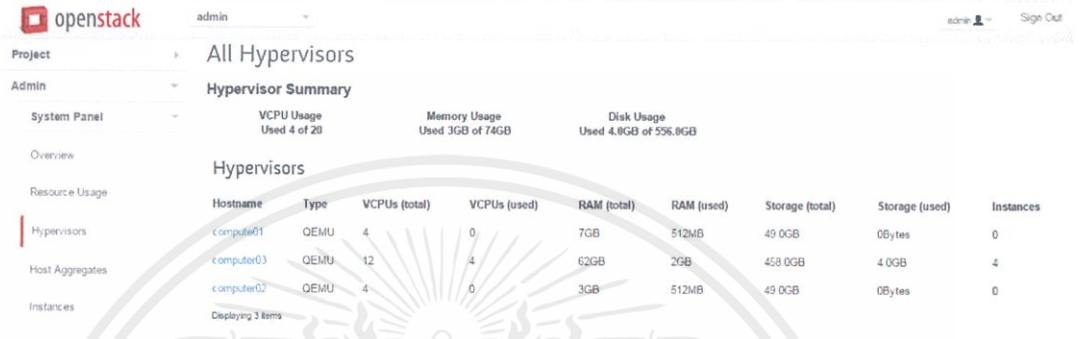
จะเป็นหน้าเว็บที่แสดงสถิติการใช้งาน resource ทุกอย่างที่อยู่บนคลาวด์



รูปที่ 4.97 ภาพแสดงหน้า Resource Usage

Hypervisors

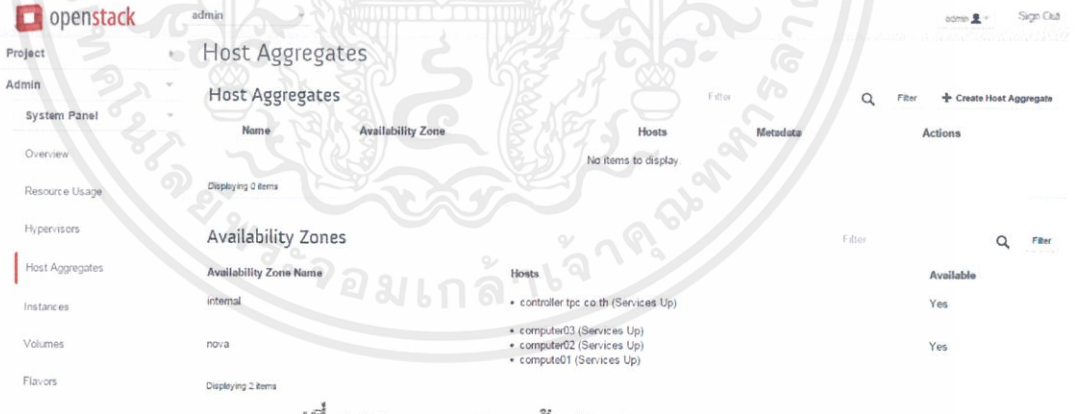
เป็นส่วนหน้าเว็บที่มีการแสดงทรัพยากรของเครื่องที่เป็นทั้ง Controller Node และ Compute Node ที่อยู่บนคลาว



รูปที่ 4.98 ภาพแสดงหน้า Hypervisors

Host Aggregates

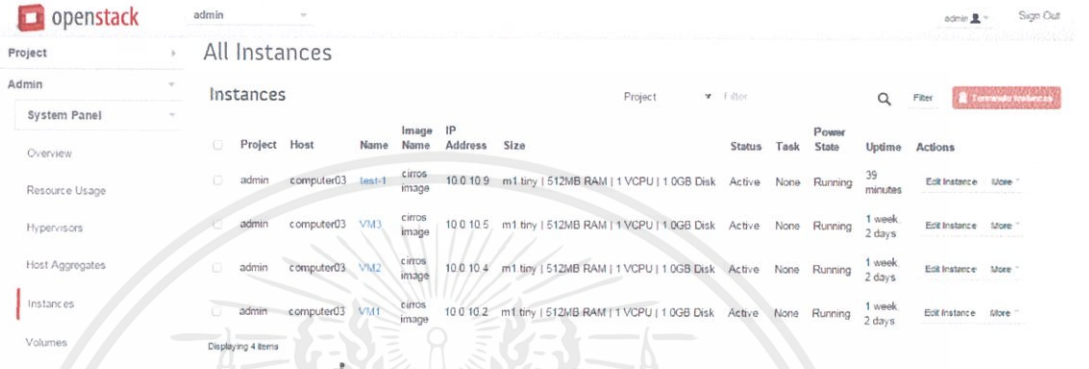
เป็นหน้าเว็บที่โชว์ สร้าง และแก้ไข ของ Host Aggregates และสามารถดูรายชื่อใน availability zones.



รูปที่ 4.99 ภาพแสดงหน้า Host Aggregates

All Instances

เป็นหน้าเว็บที่แอดมินจะเห็น Instance ทุกตัวและจะรู้ว่าแต่ละ Instance ได้ไปอยู่บนโฮสไหนบ้าง



รูปที่ 4.100 ภาพแสดงหน้า All Instances

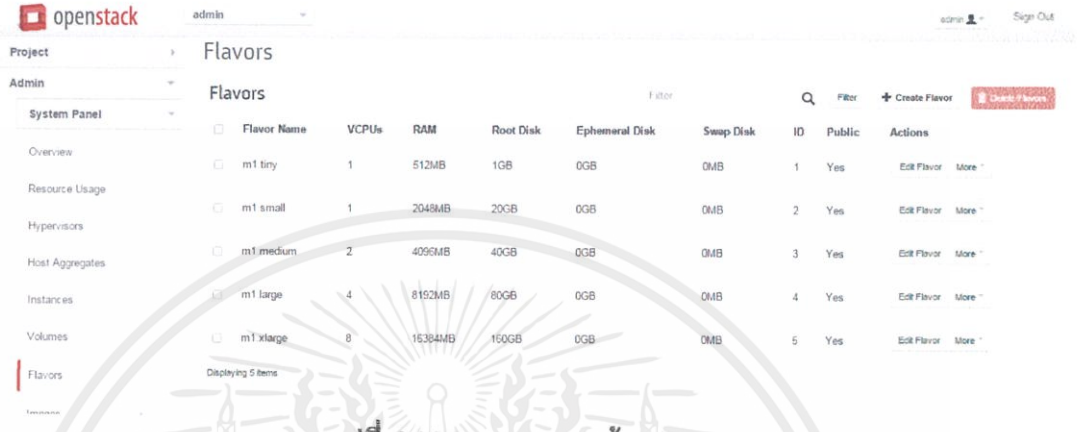
Volumes

เป็นหน้าเว็บที่จะโชว์ชื่อ Volume ทั้งหมด และสามารถสร้าง แก้ไข และลบ volumeและประเภท volume ได้

รูปที่ 4.101 ภาพแสดงหน้า Volumes

Flavors

เป็นส่วนของหน้าเว็บที่แสดง สร้าง แก้ไข และลบ flavor ซึ่งเป็นรูปแบบที่ใช้ในการเลือกขนาดของ instance



รูปที่ 4.102 ภาพแสดงหน้า Flavors

Image

เป็นส่วนของหน้าเว็บที่แสดง,สร้าง,แก้ไข และลบ image

รูปที่ 4.103 ภาพแสดงหน้า Image

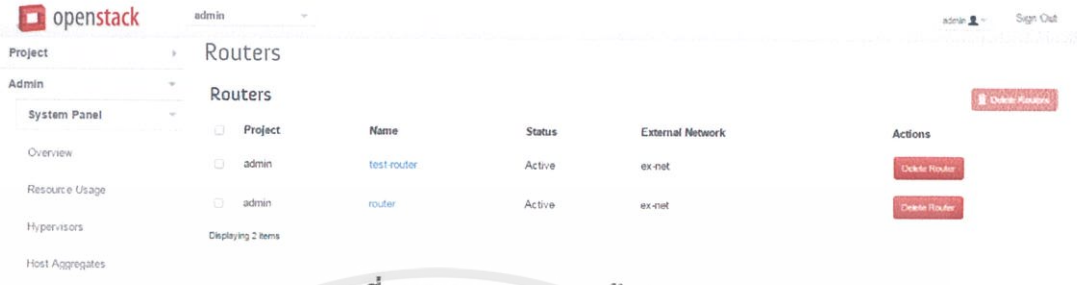
Network

เป็นส่วนของหน้าเว็บที่แสดง,สร้าง,แก้ไข และลบ network

รูปที่ 4.104 ภาพแสดงหน้า Network

Routers

เป็นส่วนของหน้าเว็บที่แสดง,สร้าง,แก้ไข และลบ router



รูปที่ 4.105 ภาพแสดงหน้า Routers

System Info

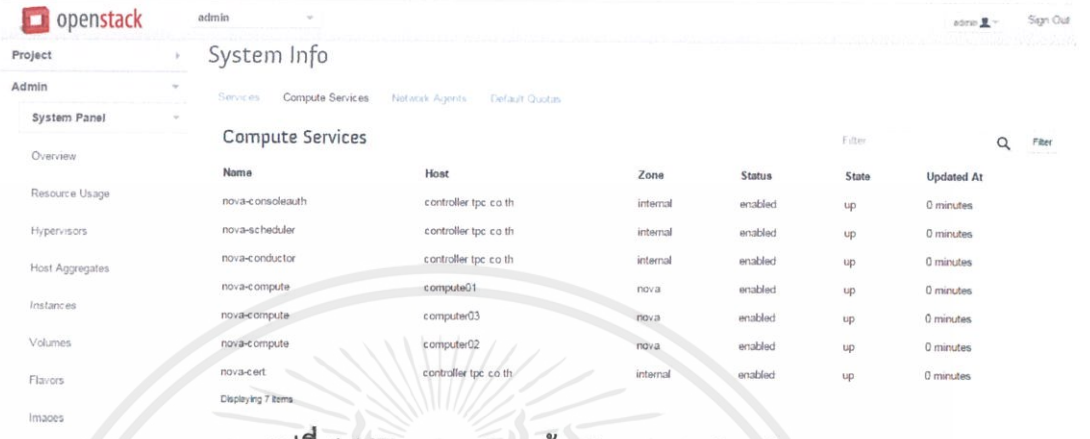
Services

เป็นส่วนแสดงสถานะเซอร์วิสของซอฟแวร์ส่วนต่างๆของโอเพนสแต็กในเครื่อง Controller Node

รูปที่ 4.106 ภาพแสดงหน้า Services

Compute Service

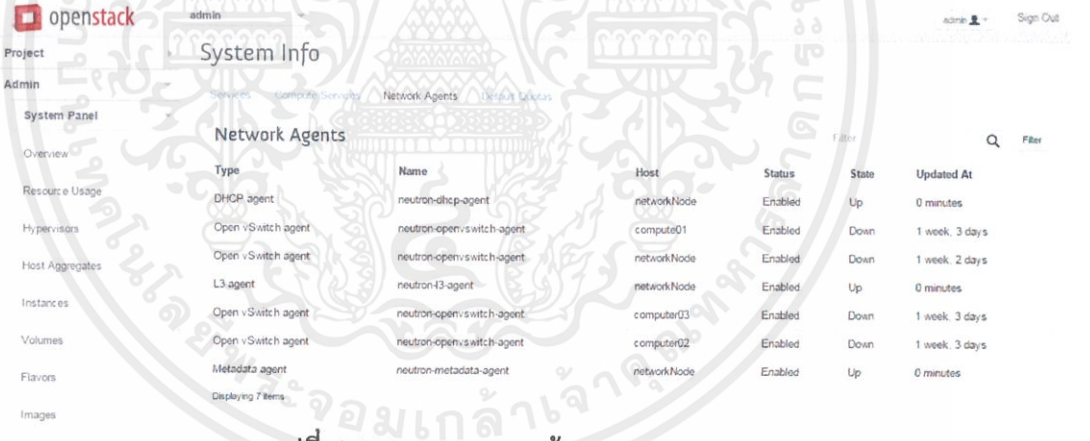
เป็นส่วนที่แสดงสถานะของ Compute service ในแต่ละเครื่องที่อยู่บนคลาวด์



รูปที่ 4.107 ภาพแสดงหน้า Compute Service

Network Agents

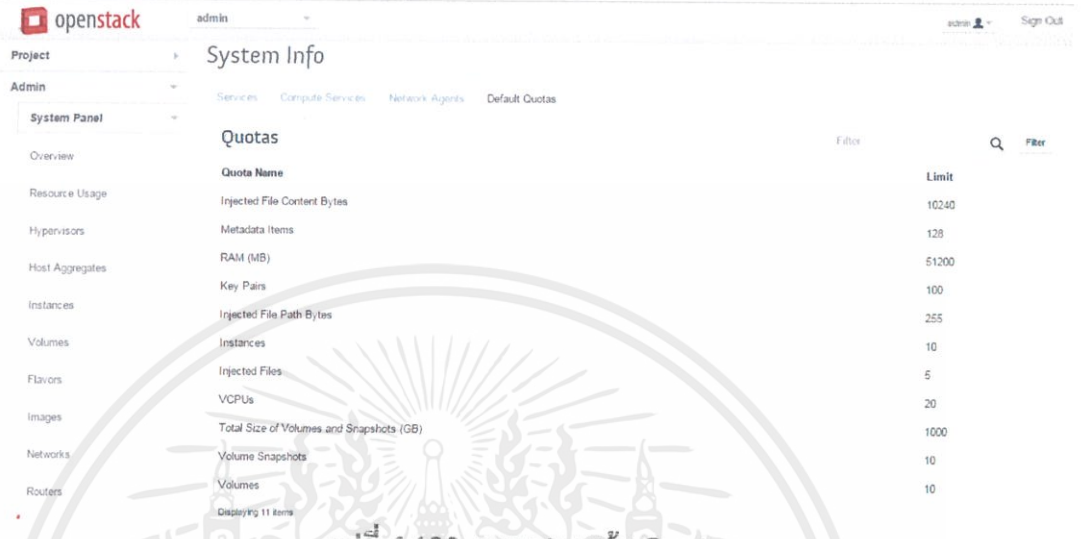
แสดงสถานะของ Network Agents ในแต่ละเครื่องที่อยู่บนคลาวด์



รูปที่ 4.108 ภาพแสดงหน้า Network Agents

Quotas

เป็นส่วนของสิทธิในการใช้งานทรัพยากรของผู้ให้บริการ



รูปที่ 4.109 ภาพแสดงหน้า Quotas

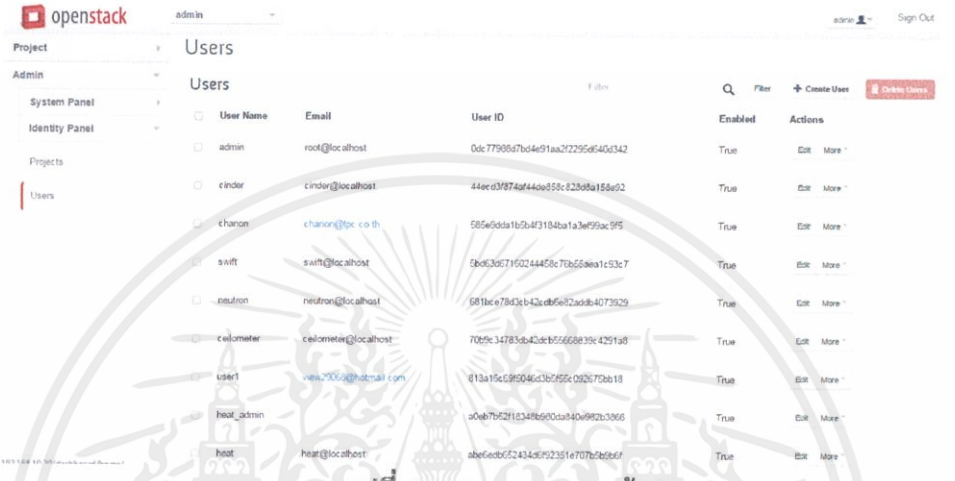
Project

ในส่วนนี้เป็นจัดการกับ project ซึ่งหนึ่งproject เปรียบเสมือนหนึ่ง tenant และหนึ่ง project สามารถรองรับผู้ใช้งานได้หลายคน ซึ่งสามารถที่จะแสดงชื่อ project ทั้งหมด และสามารถสร้าง และมอบหมายว่าจะให้ user ไหนใช้งาน project ไหนได้บ้าง และสามารถลบ Project ได้

รูปที่ 4.110 ภาพแสดงหน้า Projects

Users

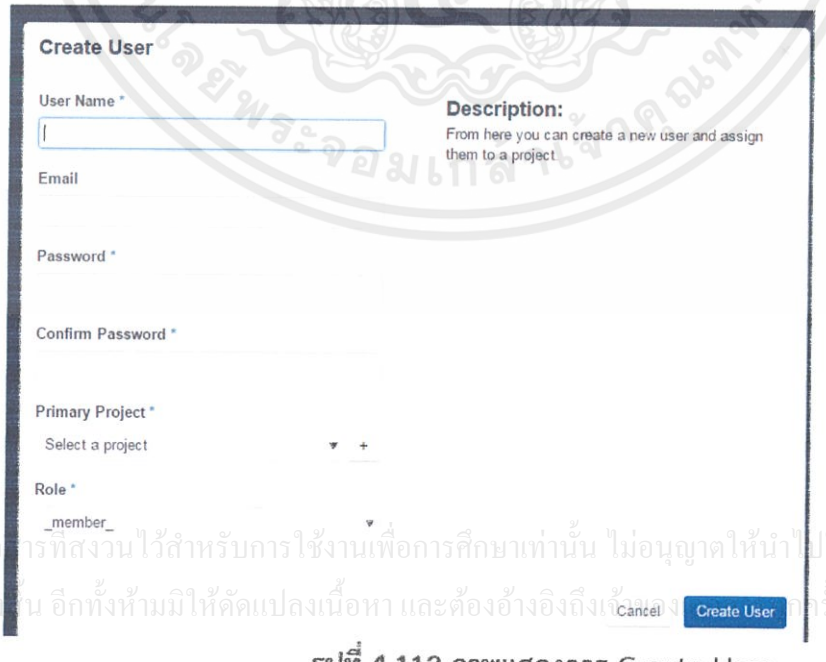
ในส่วนนี้จะจัดการกับผู้ใช้งานใน การสร้าง การลบ และการอนุญาตหรือไม่อนุญาติสิทธิของผู้ใช้งานได้ และจะแสดงรายชื่อของผู้ใช้งานทั้งหมด



รูปที่ 4.111 ภาพแสดงหน้า Users

วิธีการสร้าง User

ให้คลิกที่ปุ่ม Create User จะขึ้นหน้าต่างให้กรอกรายละเอียดว่าจะใช้ username อะไร e-mailที่ใช้ติดต่อได้ passwordที่ต้องการ แล้วก็เลือก project ว่าจะใช้ส่วนไหน และมีสิทธิและบทบาทอย่างไรกับ Project นั้น



รูปที่ 4.112 ภาพแสดงการ Create User

4.10 การใช้ Ceilometer ด้วย command line

ขั้นตอนเข้าใช้ Ceilometer (Telemetry)

ขั้นแรกเราต้องทำการเข้า keystone เพื่อบอกสิทธิการเข้าถึงและการใช้งานของการประมวลผลบนกลุ่มเมฆ จากรูป จะใช้คำสั่ง source keystone_admin เพื่อเข้าไปใช้สิทธิของ admin

```
[root@controller ~]# source keystonerc_admin
[root@controller ~(keystone_admin)]#
```

รูปที่ 4.113 ภาพแสดงการเข้าใช้งาน Ceilometer

คำสั่งเข้าใช้ Ceilometer (Telemetry)

ในการใช้ ceilometer นั้นจะต้องนำด้วย ceilometer ทุกครั้งเพื่อบอกถึงสิ่งที่เราจะใช้

1) ceilometer help เป็นคำสั่งที่ใช้สำหรับดูว่า ceilometer มีคำสั่งอะไรบ้าง แต่ละคำสั่งมีไว้ใช้ทำอะไร และต้องมีรูปแบบคำสั่งอย่างไร

```
[root@controller ~(keystone_admin)]# ceilometer help
usage: ceilometer [--version] [-d] [-v] [-k] [--cert-file CERT_FILE]
                  [--key-file KEY_FILE] [--os-cacert <ca-certificate-file>]
                  [--ca-file OS_CACERT] [--timeout TIMEOUT]
                  [--os-username OS_USERNAME] [--os-password OS_PASSWORD]
                  [--os-tenant-id OS_TENANT_ID]
                  [--os-tenant-name OS_TENANT_NAME]
                  [--os-auth-url OS_AUTH_URL]
                  [--os-region-name OS_REGION_NAME]
                  [--os-auth-token OS_AUTH_TOKEN]
                  [--ceilometer-url CEILOMETER_URL]
                  [--ceilometer-api-version CEILOMETER_API_VERSION]
                  [--os-service-type OS_SERVICE_TYPE]
                  [--os-endpoint-type OS_ENDPOINT_TYPE]
                  <subcommand> ...

Command-line interface to the OpenStack Telemetry API.

Positional arguments:
  <subcommand>
    alarm-combination-create
                          Create a new alarm based on state of other alarms.
```

รูปที่ 4.114 คำสั่งที่ใช้สำหรับดูว่า ceilometer มีคำสั่งอะไรบ้าง

2) ceilometer help command เป็นคำสั่งที่จะคล้ายข้อแรกแต่สามารถระบุเจาะจงไปที่คำสั่งที่ต้องการได้ จากรูปตัวอย่างจะเป็นการดูคำสั่ง meter-list

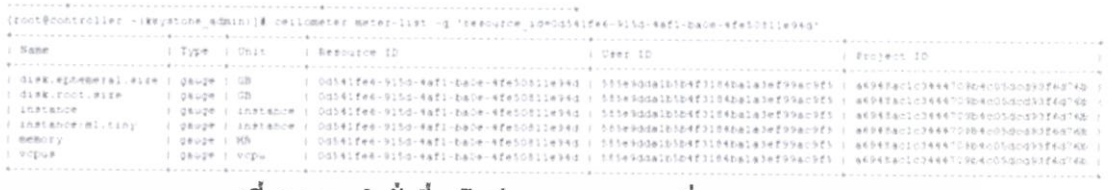
```
[root@controller ~(keystone_admin)]# ceilometer help meter-list
usage: ceilometer meter-list [-q <QUERY>]

List the user's meters.

Optional arguments:
  -q <QUERY>, --query <QUERY>
                        key[op]data_type::value; list. data_type is optional,
                        but if supplied must be string, integer, float, or
                        boolean.
```

รูปที่ 4.115 คำสั่งที่บอกการใช้งานของชื่อคำสั่งที่ระบุไป

3) ceilometer meter-list เป็นคำสั่งที่จะโชว์ทุก meter ที่มีอยู่บนระบบทั้งหมดและทุกๆ tenant

รูปที่ 4.116 คำสั่งที่จะโชว์ทุก meter ที่มีอยู่บนระบบทั้งหมด

4) ceilometer meter-list -q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d' เป็นคำสั่งที่จะโชว์เฉพาะ meter ที่เราระบุ resource_id



รูปที่ 4.117 คำสั่งที่จะโชว์เฉพาะ meter ที่เราระบุ resource_id

5) ceilometer sample-list -m vcpus เป็นคำสั่งที่ใช้ในการโชว์ sample ของ meter ที่ระบุไว้

Resource ID	Name	Type	Volume	Unit	Timestamp
3794c002-0162-4cf9-887c-60be5956d6b5	vcpus	gauge	1.0	vcpu	2015-04-23T10:37:21.289000
3794c002-0162-4cf9-887c-60be5956d6b5	vcpus	gauge	1.0	vcpu	2015-04-23T10:37:15.714000
1ecf5413-2da9-4af0-b43e-52899e45c80f	vcpus	gauge	1.0	vcpu	2015-04-23T10:29:42.814000
1ecf5413-2da9-4af0-b43e-52899e45c80f	vcpus	gauge	1.0	vcpu	2015-04-23T10:29:37.093000
a67e9139-ba80-43b6-8e23-cf69c8b36b16	vcpus	gauge	1.0	vcpu	2015-04-23T09:54:30.262000
a67e9139-ba80-43b6-8e23-cf69c8b36b16	vcpus	gauge	1.0	vcpu	2015-04-23T09:54:24.582000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:02:39.150000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:02:38.637000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:02:37.347000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:02:37.326000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:02:14.680000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:01:40.481000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:01:39.901000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:01:38.913000
bce8f065-4600-4426-aca3-667e1abd886a	vcpus	gauge	1.0	vcpu	2015-04-23T03:01:34.185000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:43.850000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:43.434000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:42.753000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:42.207000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:40.239000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:40.230000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:39.781000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:39:39.769000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:38:48.521000
8317b3d7-3b7a-44a8-8bb8-f591b14350ca	vcpus	gauge	1.0	vcpu	2015-04-23T02:38:43.283000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:37:00.384000
71bbc39b-fbdb-4414-84e0-ca3e75e7bc97	vcpus	gauge	1.0	vcpu	2015-04-23T02:36:55.146000
ecebf683-8ffe-4af0-a6ee-0686ee592cde	vcpus	gauge	1.0	vcpu	2015-04-23T02:26:24.624000
ecebf683-8ffe-4af0-a6ee-0686ee592cde	vcpus	gauge	1.0	vcpu	2015-04-23T02:26:24.151000
ecebf683-8ffe-4af0-a6ee-0686ee592cde	vcpus	gauge	1.0	vcpu	2015-04-23T02:26:22.848000
ecebf683-8ffe-4af0-a6ee-0686ee592cde	vcpus	gauge	1.0	vcpu	2015-04-23T02:26:22.836000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:19:09.459000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:19:08.915000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:19:07.272000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:19:07.267000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:16:45.169000
f1f1773a-d0a3-4bf4-ad6f-c3ff1246c2f9	vcpus	gauge	1.0	vcpu	2015-04-23T02:16:39.995000
40179e7d-eaf4-4b9f-928a-6cbb9ac108a1	vcpus	gauge	1.0	vcpu	2015-04-23T02:11:23.198000
40179e7d-eaf4-4b9f-928a-6cbb9ac108a1	vcpus	gauge	1.0	vcpu	2015-04-23T02:11:22.596000
40179e7d-eaf4-4b9f-928a-6cbb9ac108a1	vcpus	gauge	1.0	vcpu	2015-04-23T02:11:21.226000

รูปที่ 4.118 คำสั่งที่ใช้ในการโชว์ sample ของ meter ที่ระบุ

6) ceilometer sample-list -m instance \-q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d' เป็นคำสั่งที่ใช้ในการโชว์ sample ที่ระบุทั้ง meter และ resource_id

```
[root@controller ~(keystone_admin)]# ceilometer sample-list -m instance \
> -q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d'
+--------------------------------------+----------+-------+--------+----------+----------------------------+
| Resource ID                          | Name     | Type  | Volume | Unit     | Timestamp                  |
+--------------------------------------+----------+-------+--------+----------+----------------------------+
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:42:33.371000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:42:32.605000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:42:31.310000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:42:31.305000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:36:59.237000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:36:53.392000 |
+--------------------------------------+----------+-------+--------+----------+----------------------------+
```

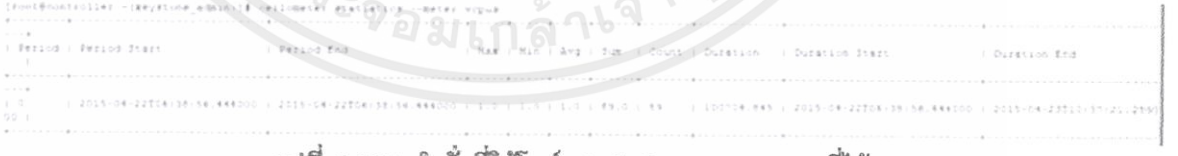
รูปที่ 4.119 คำสั่งที่ใช้ในการโชว์ sample ที่ระบุทั้ง meter และ resource_id

7) ceilometer sample-list -m instance \-q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d;timestamp>2015-04-23T01:35;timestamp<2015-04-23T01:40' เป็นคำสั่งที่ใช้ในการโชว์ sample ที่ระบุ meter , resource_id และที่กำหนดช่วง Timestamp ที่เราต้องการ

```
[root@controller ~(keystone_admin)]# ceilometer sample-list -m instance \
> -q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d;timestamp>2015-04-23T01:35;timestamp<2015-04-23T01:40'
+--------------------------------------+----------+-------+--------+----------+----------------------------+
| Resource ID                          | Name     | Type  | Volume | Unit     | Timestamp                  |
+--------------------------------------+----------+-------+--------+----------+----------------------------+
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:36:59.237000 |
| 0d541fe6-915d-4af1-ba0e-4fe50811e94d | instance | gauge | 1.0    | instance | 2015-04-23T01:36:53.392000 |
+--------------------------------------+----------+-------+--------+----------+----------------------------+
```

รูปที่ 4.120 คำสั่งโชว์ sample ที่ระบุ meter , resource_id และที่กำหนด Timestamp

8) ceilometer statistics --meter vcpus เป็นคำสั่งที่ใช้โชว์ statistic ของ meter ที่ได้ระบุไว้



รูปที่ 4.121 คำสั่งที่ใช้โชว์ statistic ของ meter ที่ได้ระบุ

9) ceilometer statistics -m vcpus -q 'resource_id=0d541fe6-915d-4af1-ba0e-4fe50811e94d' เป็นคำสั่งที่ใช้โชว์ statistic ที่ได้ระบุทั้ง meter และ resource_id

รูปที่ 4.122 คำสั่งที่ใช้โชว์ statistic ที่ได้ระบุทั้ง meter และ resource_id

บทที่ 5
บทสรุปและข้อแนะนำ

5.1 สรุปและวิจารณ์

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

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

Block Storage (Cinder)

Cinder นั้นเป็น Block Storage หรือฮาร์ดดิสก์เสมือนนั่นเอง ผู้ใช้สามารถสร้างฮาร์ดดิสก์เสมือนขึ้นมาและนำไปเชื่อมต่อกับ instance ใดที่กำลังทำงานอยู่ก็ได้ Cinder นั้นสามารถสร้างฮาร์ดดิสก์เสมือนจากFilesystemจริงหลายรูปแบบ เช่นจากLVM, NFS, หรือ GlusterFS เป็นต้น นอกจากนี้ Cinder ยังสามารถทำ Snapshot หรือ ภาพสถานะปัจจุบันของฮาร์ดดิสก์ และเก็บไว้เป็นแบ็คอัพบน Swift ได้ด้วย โมดูล Cinder นี้นั้นเทียบเท่ากับบริการ EBS ของ Amazon AWS

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 ของระบบคลาวด์ ซึ่งมันสามารถนำมาใช้โดยผู้ดูแลระบบ

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 OpenStack Telemetry (Ceilometer)

Ceilometer เป็นโมดูลที่ใช้เก็บวัดจำนวนทรัพยากรที่ถูกใช้ไปโดยผู้ใช้แต่ละคน โดยเริ่มต้นนั้นผู้พัฒนาตั้งใจให้ Ceilometer นั้นทำหน้าที่เก็บข้อมูลเพื่อคิดค่าบริการเป็นหลัก แต่ปัจจุบันสามารถใช้ Ceilometer นำไปทำงานอื่นได้ เช่นเพื่อการปรับจำนวนทรัพยากรอัตโนมัติตามภาระงานเป็นต้น

5.2 ปัญหาอุปสรรคและแนวทางการแก้ไข

5.2.1 การใช้ GlusterFS ร่วมกับ OpenStack Storage

1) ปัญหาอุปสรรค - ไม่สามารถเชื่อมต่อ Storage หลายๆเครื่องเพื่อทำ pool สำหรับการใช้ GlusterFS
แนวทางการแก้ไข - ทำการปิด FireWall ของทุกเครื่อง เพื่อให้ทุกเครื่องเข้าถึงกันได้
2) ปัญหาอุปสรรค - เมื่อทำการใช้การ GlusterFS ร่วมกับบริการ Object Storage แล้วไม่สามารถเก็บข้อมูลที่เป็น Object ได้
แนวทางการแก้ไข - ตรวจสอบสิทธิการเข้าถึงไฟล์ โดยตั้งค่าให้ Swift เข้าถึงไฟล์ที่เก็บข้อมูลที่เป็น Object ได้ จึงจะทำให้บริการ Swift สามารถจัดการ Object Storage ได้
3) ปัญหาอุปสรรค - Block Storage ไม่สามารถใช้งานร่วมกับ GlusterFS
แนวทางการแก้ไข – จากการศึกษาพบว่าเป็น Bug ของ OpenStack เวอร์ชัน Icehouse จึงทำการเปลี่ยนมาใช้เวอร์ชัน Havana แทน

5.2.2 การติดตั้ง OpenStack Networking

1) ปัญหาอุปสรรค – Hard Disk ของเครื่อง Controll เสีย
แนวทางการแก้ไข – ติดตั้งระบบคลาวด์ใหม่ทั้งหมด และนำ GlusterFS มาใช้งานในการทำการ Back up ข้อมูล เพื่อป้องกันปัญหา Hard Disk ของเครื่อง Controll เสียอีกในอนาคต
2) ปัญหาและอุปสรรค – เมื่อนำ OpenDaylight มาใช้กับ OpenStack พบว่าทำให้ระบบคลาวด์ไม่มีความเสถียร ระบบคลาวด์เสียหายได้ทุกเมื่อโดยไม่มีสาเหตุ
แนวทางการแก้ไข – พบว่าเป็น Bug ที่เกิดจาก OpenDaylightจึงทำการนำ OpenDaylight ออกจากระบบคลาวด์

5.2.3 การติดตั้ง OpenStack Networking OpenStack Telemetry

1) ปัญหาอุปสรรค – Hard Disk ของเครื่อง Controll เสีย
แนวทางการแก้ไข – ติดตั้งระบบคลาวด์ใหม่ทั้งหมด และนำ GlusterFS มาใช้งานในการทำการ Back up ข้อมูล เพื่อป้องกันปัญหา Hard Disk ของเครื่อง Controll เสียอีกในอนาคต

5.3 แนวทางการพัฒนาต่อ

Software-Defined Anything with Cloud Computing

ในยุคของคลาวด์คอมพิวติง ศูนย์ข้อมูลของผู้ให้บริการรายสำคัญ ๆ มักจะเป็นศูนย์ขนาดใหญ่มาก มีอุปกรณ์คอมพิวเตอร์ ตู้บันทึกข้อมูล และตู้ควบคุมการทำงานของระบบเครือข่ายจำนวนมาก ไม่เพียง 10-20 ตู้ แต่อาจมีถึงพัน ๆ หมื่น ๆ หรือแสน ๆ ตู้ การบริหารจัดการให้ระบบไอซีทีที่มีอุปกรณ์ชนิดต่าง ๆ จำนวนมหาศาลนี้ให้ทำงานอย่างราบรื่น มีวิธีบำรุงรักษา และแก้ไขปัญหาเมื่อมีเหตุการณ์ผิดปกติเกิดขึ้นนั้น ไม่ใช่เรื่องทำได้ง่าย ๆ การบริหารศูนย์ข้อมูลขนาดใหญ่มากนี้ ไม่ง่าย ซอฟต์แวร์พิเศษจะถูกสร้างขึ้นเพื่อทำหน้าที่ควบคุม ประสานงานระหว่างส่วนต่าง ๆ ของระบบจำนวนมากที่กระจายอยู่ทั่วโลก พร้อมติดตามและจดจำงานทุกขั้นตอน เก็บข้อมูลไว้เพื่อช่วยการวินิจฉัยปัญหาและวิเคราะห์หาทางแก้ไขได้อย่างรวดเร็วและมีประสิทธิภาพเมื่อเกิดความจำเป็น ซึ่งหมายความถึงมีซอฟต์แวร์กำหนดวิธีการทำงานและติดตามการทำงานทุกขั้นตอนของอุปกรณ์ต่าง ๆ โดยเฉพาะการประสานงานและร่วมทำงานกันระหว่างกลุ่มทรัพยากรไอซีที Software-defined network (SDN) คือระบบเครือข่ายที่มีซอฟต์แวร์ทำหน้าที่จัดรูปแบบ (Configure) และควบคุมการทำงานของระบบเครือข่ายจากจุดเดียว

SDDC (Software-defined data center) ที่มีซอฟต์แวร์ทำหน้าที่สนับสนุนการจัดการการดำเนินงานของศูนย์ข้อมูลที่อาศัย Virtualization เพื่อบริหารจัดการทรัพยากรไอที นอกจากนี้ยังมีระบบซอฟต์แวร์ทำหน้าที่ดูแลรักษาความปลอดภัย การจัดการระบบเครือข่าย การจัดการระบบฐานข้อมูล ฯลฯ

SDS (Software-defined storage) ที่มีหลักการเดียวกันกับ SDN คืออาศัยซอฟต์แวร์จัดการและควบคุมการทำงานของเครื่องบันทึกข้อมูล และจัดการให้กลุ่มเครื่องบันทึกข้อมูลทำงานภายใต้หลักการของVirtual Storage

ดังนั้นการนำ Software-Defined Anything มาใช้กับระบบคลาวด์คอมพิวติง จะสามารถช่วยการจัดการบริการต่างๆในระบบคลาวด์ให้ง่ายต่อการควบคุมมากขึ้น

บรรณานุกรม

[1] So Good Web. "**Cloud Computing คืออะไร**" [Online]. Available : http://about.sogoodweb.com/ArticleList.aspx?ArticleTypeID=184

[2] Tutorialspoint. "**Cloud Computing Overview**" [Online]. Available : http://www.tutorialspoint.com/cloud_computing/cloud_computing_overview.htm

[3] Mat. "**Virtualization คืออะไร**" [Online]. Available : http://www.mat.co.th/th/products/si_consulting/virtual/

[4] Thaiopensource. "**GlusterFS**" [Online]. Available : http://thaiopensource.org/tag/glusterfs/

[5] Gluster. "**GlusterFS**" [Online]. Available : http://www.gluster.org/community/documentation/index.php/QuickStart

[6] Gluster. "**GlusterFS Cinder**" [Online]. Available : http://www.gluster.org/community/documentation/index.php/GlusterFS_Cinder

[7] Gsr-linux. "**GlusterFS integration with openstack-cinder and openstack-glance (Grizzly) using packstack --allinone**" [Online]. Available : http://gsr-linux.blogspot.com/2013/07/glusterfs-integration-with-openstack.html

[8] Server-world. "**Configure OpenStack Object Storage (Swift).**" [Online]. Available : http://www.server-world.info/en/note?os=CentOS_6&p=openstack_icehouse2&f=5

[9] Stalker. "**File Systems**" [Online]. Available : http://www.stalker.com/notes/SFS.html

[10] Gluster. "**GlusterFS iSCSI**" [Online]. Available : http://www.gluster.org/community/documentation/index.php/GlusterFS_iSCSI

[11] OpenDaylight. "**OpenStack and OpenDaylight**" [Online]. Available : https://wiki.opendaylight.org/view/OpenStack_and_OpenDaylight#Ensuring_OpenStack_network_state_is_clean

[12] Blognone. "**Software-defined Network (SDN)**" [Online]. Available : https://www.blognone.com/node/56144

[13] OpenDaylight. "**OpenDaylight**" [Online]. Available : http://www.opendaylight.org/

[14] Redhat. "**OPENSTACK NETWORKING INSTALLATION OVERVIEW**" [Online]. Available : https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/5/html/Installation_and_Configuration_Guide/sect-OpenStack_Networking_Installation_Overview.html

[15] OpenStack. "**Plug-in architecture**" [Online]. Available : http://docs.openstack.org/admin-guide-cloud/content/section_plugin-arch.html

[16] OpenStack. "**Open vSwitch concepts**" [Online]. Available : http://docs.openstack.org/havana/install-guide/install/yum/content/concepts-neutron.openvswitch.html