Київський національний університетімені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Дипломна робота бакалавра

за спеціальністю: 123 Комп'ютерна інженерія

на тему:

Дослідження засобів побудови віртуальної лабораторної інфраструктури для роботи з електронно-обчислювальними машинами

Виконав:

студент 4-го курсу бакалаврату

Максим КОНДРАТЕНКО _____
(підпис)

Науковий керівник:

Асистент, кандидат технічних наук

Олександр БОРЕЦЬКИЙ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2022

Зміст

Реферат

Обсяг роботи 58 сторінок, 3 розділи, 19 ілюстрацій, 11 джерел посилань.

Об’єктом дослідження є програмний комплекс віртуальної інфраструктури.

Метою роботи є аналіз особливостей віртуальної лабораторної інфраструктури та створення такої інфраструктури за допомогою наявного обладнання та спеціалізованого програмного забезпечення.

Після цього отриманий результат вдало пройшов перевірку на відповідність сформульованим вимогам.

Вступ

Наразі сервіси, які надають послуги хмарних обчислень стали невід'ємною частиною арсеналу it спеціалістів. Це підтверджується щорічним зростанням прибутків та користувацької бази таких хмарних провайдерів як AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud Services та інших. Основною причиною такої популярності є зручність використання, пришвидшення робочого процесу та запобігання репетативності завдань для обслуговчого персоналу.

Гнучкість використання технологій віртуальних машин дозволяє розміщувати будь-які програмні рішення у хмарі. При цьому користувачу не потрібно витрачати час на обслуговування фізичного обладнання. Також, маючи доступ до зручних інтерфейсів керування, користувач економить максимальну кількість людських ресурсів, які направляються на покращення програмного продукту, замість обслуговування інфраструктури.

Також розвиток технологій контейнеризації застосунків дозволяє ефективно розподіляти навантаження по доступних обчислювальних потужностях, що дозволяє запускати велику кількість ізольованих сервісів на наявному обладнанні.

Одним з найпопулярніших на сьогодні програмних комплексів для створення хмарного середовища без використання ресурсів сторонніх сервісів є OpenStack. Проект OpenStack має відкритий код та модульну архітектуру. Така модель надає більшу гнучкість застосування, спрощує розширення функціоналу шляхом написання власних програмних модулів та дозволяє обізнаним професіоналам прибирати помилки і вразливості коду в максимально короткий проміжок часу.

Такий програмний комплекс полегшить, пришвидшить та покращить навчання студентів в галузі it технологій, оскільки більшість лабораторних робіт проводяться з використанням комп’ютерів, комп’ютерних мереж та різноманітного програмного забезпечення, яке на них запускається. Віртуальна лабораторна інфраструктура дозволить студентам та викладачам запускати обчислення та сервіси з високими потребами у обчислювальних потужностях та апаратному забезпеченні. Також скоротиться період часу, який використовується на підготовку необхідного середовища перед початком виконання завдань. Віртуальна інфраструктура надає можливість кожному мати свою власну лабораторію з комп’ютерних систем та мереж в хмарі. З цього випливає багато можливостей, які недоступні студентам без доступу до такої інфраструктури. Тим самим підвищиться якість навчання.

1 Дослідження особливостей завдання дипломної роботи

Віртуальна лабораторна інфраструктура — це самостійний сервіс хмарних обчислень, який надає засоби віртуалізації та контейнеризації, а також віртуальну мережу, для запуску проектів будь-якого розміру. Такими проектами можуть бути як мінімалістичний застосунок у вигляді контейнеру з маленькою програмою так і великі проекти з приватною віртуальною мережею і десятками контейнерів та віртуальних машин.

Метою такої інфраструктури є як пришвидшення та полегшення як створення вище описаних проектів, так і їхнього закриття, автоматизація цих процесів, і в наслідок цього — ефективніше використання обладнання без компромісів.

Така віртуальна лабораторна інфраструктура може використовуватись для рішення великого списку завдань, одними з найважливіших є такі застосування:

- Виконання лабораторних робіт
- Дослідження принципу роботи комп’ютерних мереж
- Використання у вигляді тестового середовища
- Платформа для студентських проектів
- Виконання практичних навчальних демонстрацій
- Тощо

Опираючись на такий список використань, виникає необхідність у задоволенні певних потреб, відповідно до застосування. Для виконання лабораторних робіт інфраструктура повинна надавати такі можливості:

- Створення віртуальних віртуальних вузлів як з ОС Linux так і з Windows. Додатково бажана підтримка інших ОС.
- Створення приватної мережі між вузлами.
- Віддалений доступ до консолі та графічного інтерфейсу вузлів.
- Доступ до мережі інтернет з вузла.
- Розгортання системи з образу.

Застосування у вигляді тестового середовища потребує:

- Можливість автоматичного запуску команд на вузлі
- Автоматичне створення і видалення вузла для запуску тестів.
- Сервіс збирання артефактів та логів.
- В деяких випадках доступ до консольного інтерфейсу вузла.

В разі виконання студентських проектів, користувачу необхідно:

- Віддалений доступ до консолі та графічного інтерфейсу вузлів.
- Доступ до мережі інтернет з вузла.
- Доступ до вузла з мережі інтернет.
- Розгортання системи з образу.

І для використання з ціллю створення навчальних демонстрацій потрібно:

- Розгортання системи з образу.
- Доступ до вузла з мережі інтернет.
- В деяких випадках доступ до консольного інтерфейсу вузла.

Навчальна програма студентів комп’ютерної інженерії передбачає виконання лабораторних робіт з програмування, алгоритмів, веб-

програмування, організації баз даних, адміністрування unix систем та інші. Так, кожен з типів лабораторних робіт отримає певну вигоду від наявності віртуальної лабораторної інфраструктури.

Лабораторні з програмування потребують наявності комп'ютера з операційною системою, середовища розробки, запуску та відлагодження програми. Віртуальна лабораторна інфраструктура може забезпечити всі ці потреби.

Лабораторні роботи з алгоритмів потребують середовища виконання програм. Такі лабораторні роботи отримають вигоду від можливості розподіленого запуску обчислень на декількох вузлах для пришвидшення обрахунків. Також використання віртуальної лабораторної інфраструктури дозволить здобути навички в сфері розподілених обчислень.

Лабораторні роботи з адміністрування комп'ютерних систем також потребують комп'ютера з операційною системою та встановленим програмним забезпеченням. Для таких лабораторних робіт віртуальна лабораторна інфраструктура дозволить забезпечити розгортання готових зображень операційної системи з встановленими залежностями, щоб студенти могли відразу приступити до виконання робіт, а також могли просто почати заново у разі невдачі.

Лабораторні роботи з unix систем потребують для виконання комп'ютера з встановленою unix системою. Віртуальна лабораторна інфраструктура дозволить кожному студенту швидко розгорнути віртуальний комп'ютер з необхідною операційною системою для виконання робіт.

Лабораторні роботи з організації баз даних потребують комп'ютера з встановленим програмним забезпеченням. Це також можна просто

задовольнити за допомогою віртуальної лабораторної інфраструктури, як і в попередніх випадках.

Лабораторні роботи з веб-програмування вимагають наявності веб-хостингу для наочного процесу виконання роботи, який передбачає запуск веб-сайту в мережі інтернет. Саме це можна забезпечити за допомогою публічних ір адрес, віртуальної мережі та віртуальних вузлів, які надає віртуальна лабораторна інфраструктура.

Не залежно від типу лабораторних робіт також необхідними є функції створення віртуальної мережі, запуск програм на декількох вузлах одночасно, створення власних зображень операційних систем, запуск контейнерів замість віртуальних машин, віддалений доступ до графічного інтерфейсу вузлів, клонування та масове створення нових віртуальних вузлів, тощо.

Аналіз програмних засобів інфраструктури віртуалізації

Система віртуалізації oVirt

oVirt — це безкоштовна платформа для управління віртуалізацією з відкритим кодом, яка базується на технології KVM (Kernel-based Virtual Machine). oVirt був створений Red Hat як cutting-edge проект, на стабільних версіях якого базується віртуалізація Red Hat Enterprise. Він дозволяє централізовано керувати віртуальними машинами, обчислювальними, сховищами та мережевими ресурсами з простого у використанні веб-інтерфейсу з незалежним від ОС доступом а також за допомогою api libvirt. kvm на архітектурі x86-64, PowerPC64 та s390x єдиними підтримуваними гіпервізорами, але в майбутніх випусках продовжуються зусилля щодо

підтримки архітектури arm. oVirt складається з двох основних компонентів, oVirt-engine і oVirt-node.

Бек-енд oVirt-engine написаний на java, тоді як інтерфейс розроблено за допомогою веб-інструментів gwt. oVirt-engine працює на сервері застосунків WildFly. Доступ до інтерфейсу можна отримати через портал веб-адміністратора для адміністрування або портал користувача з привілеями та функціями, які можна налаштовувати. Адмініструванням користувачів можна керувати локально або шляхом інтеграції oVirt із ldap або ad. Механізм oVirt зберігає дані в базі даних postgresql. Веб інтерфейс також надає rest api доступний для налаштування функцій серверу. Бек-енд oVirt-engine може бути встановлений на автономному сервері або може бути розміщений на кластері вузлів у віртуальній машині.

Вузол oVirt-node — це сервер, на якому працює rhel, CentOS, та інші дистрибутиви (які підтримуються в тестовому режимі), з гіпервізором kvm і написаним на Python демоном vdsm. Налаштування системи відбувається з веб-порталу адміністратора, який через бек-енд здійснює відповідні виклики до демона vdsm. vdsm контролює всі ресурси, доступні для вузла сховище, пам’ять, обчислювальні ресурси, мережу і віртуальні машини, що працюють на ньому, а також відповідає за надання зворотного зв'язку до oVirt-engine про всі здійснені операції. oVirt-node зазвичай керується мінімальним образом ОС на основі Fedora, який може бути або встановлений на носій даних в машині, або завантажуватися з мережі.

oVirt побудований на кількох інших проектах: PatternFly, libvirt, Gluster, та Ansible. Сховище організовано в об’єктах, які називаються доменами зберігання, і може бути локальним або спільним. Домени сховища можна створити за допомогою протоколів зберігання даних:

- NFS
- iSCSI
- Fibre Channel
- GlusterFS
- Posix Filesystem (локально)

Управління мережею дозволяє визначити кілька vlan-ів, які можуть бути підключені до мережевих інтерфейсів вузлів. Конфігурація зв'язаних інтерфейсів, IP-адрес, масок підмережі та шлюзів на керованих вузлах налаштовується через веб-портал адміністратора або api.

Функції керування для обчислювальних ресурсів включають закріплення процесу за ядром ЦП, визначення топології numa, kernel same-page merging, та інші.

Керування віртуальними машинами дає змогу вибирати пріоритет високої доступності, живу міграцію, живі знімки, клонувати віртуальні машини зі знімків, створювати шаблони віртуальних машин, використовувати cloud-init для автоматичного налаштування під час надання та розгортання віртуальних машин. Підтримувані гостьові операційні системи включають Linux, Microsoft Windows і FreeBSD. Доступ до віртуальних машин можна отримати з порталу веб-адміністратора за допомогою протоколів spice, vnc і rdp.

oVirt можна інтегрувати з багатьма проектами з відкритим вихідним кодом, включаючи OpenStack Glance і Neutron для надання сховища та мережі, Foreman/Katello для створення віртуальної машини.

Функції аварійного відновлення включають можливість імпортувати будь-який домен сховища в різні екземпляри oVirt-engine, а реплікацією можна керувати з oVirt за допомогою функції геореплікації GlusterFS або за

допомогою реплікації, наданої постачальниками обладнання для зберігання даних. Резервне копіювання oVirt-engine можна автоматизувати та періодично переносити у географічно віддалене місце.

Результат: + – так; - – ні; Враховується лише вбудований функціонал.

Забезпечення відмовостійкості	+
Система автентифікації, авторизації та обліку	+
Веб інтерфейс	+
Розгортання системи з образу	+
Створення віртуальних вузлів як з ОС Linux так і з Windows	+
Віддалений доступ до консолі та графічного інтерфейсу вузлів	+
Налаштування мережі вузла	-
Сервіс збирання артефактів та логів	-
Створення приватних мереж	-
Можливість автоматичного запуску команд на вузлі	-
Автоматичне створення та видалення вузла	-

Табл. 1.1.1.1 Таблиця відповідності oVirt критеріям відбору програмного комплексу для віртуальної лабораторної інфраструктури

Система віртуалізації Proxmox

Proxmox Virtual Environment базується на Debian GNU/Linux разом з спеціалізованим ядром. Так як і в oVirt для віртуалізації використовується Kernel-based Virtual Machine. Вихідний код Proxmox VE є безкоштовним, випущений під загальною публічною ліцензією GNU Affero, версія 3, це означає, що ви можете вільно використовувати програмне забезпечення, переглядати вихідний код і робити свої внески в проект. Proxmox гарантує повний доступ до всіх функцій безкоштовно, а також високий рівень надійності та безпеки.

Proxmox також підтримує технологію віртуалізації на основі контейнерів за допомогою Linux Containers. LXC — це середовище віртуалізації на рівні операційної системи для запуску кількох ізольованих систем Linux на одному хості управління Linux. LXC працює як запускає контейнери в просторі користувача і використовує функції ядра Linux для забезпечення ізоляції.

Веб-інтерфейс заснований на JavaScript-фреймворку ExtJS і доступний з будь-якого сучасного браузера. Окрім завдань керування, він також надає огляд історії завдань та системних журналів для кожного вузла. Цей журнал включає інформацію про виконання завдань резервного копіювання, активну міграцію, використання сховища та інші.

Proxmox VE використовує унікальну кластерну файлову систему Proxmox pmxcfs. Фізичне сховище може бути підключено за допомогою:

- iSCSI
- NFS
- SMB/CIFS
- Ceph RBD
- Direct to iSCSI LUN
- GlusterFS
- CephFS
- Posix Filesystem (локально)

Для досвідчених користувачів, які звикли до оболонки Unix або Windows Powershell, Proxmox надає інтерфейс командного рядка для керування всіма компонентами віртуального середовища. Ще один спосіб керування — rest api, який використовує json як основний формат даних та

має формальне визначення у вигляді json схеми. Таким чином забезпечується швидка та проста інтеграція зі сторонніми додатками.

Рольова система доступу дозволяє визначити детальний доступ до всіх об’єктів (наприклад, віртуальних машин, сховища, вузлів тощо) за допомогою системи керування дозволами. Proxmox VE підтримує кілька джерел автентифікації, наприклад Linux pam, інтегрований сервер аутентифікації Proxmox VE, LDAP, Microsoft Active Directory і OpenID Connect.

Результат: + – так; - – ні; Враховується лише вбудований функціонал.

Забезпечення відмовостійкості	+
Система автентифікації, авторизації та обліку	+
Веб інтерфейс	+
Розгортання системи з образу	+
Створення віртуальних вузлів як з ОС Linux так і з Windows	+
Віддалений доступ до консолі та графічного інтерфейсу вузлів	+
Налаштування мережі вузла	-
Сервіс збирання артефактів та логів	-
Створення приватних мереж	-
Можливість автоматичного запуску команд на вузлі	-
Автоматичне створення та видалення вузла	+

Табл. 1.1.2.1 Таблиця відповідності Proxmox критеріям відбору програмного комплексу для віртуальної лабораторної інфраструктури

Програмний комплекс для створення приватної обчислювальної хмари OpenStack

OpenStack — це безкоштовна відкрита стандартна платформа хмарних обчислень. OpenStack складається із компонентів, що обробляють, зберігають і контролюють мережеві ресурси. Керування сервісами OpenStack відбувається за допомогою веб-інтерфейсу, інструментів командного рядка

або rest api. Ввесь вихідний код OpenStack відкритий і опублікований, кожен може його завантажити для власного використання, або запропонувати покращення. Архітектура OpenStack модульна, що дозволяє забезпечити всі потреби користувача, або за допомогою офіційних сервісів OpenStack, або за допомогою власних модулів.

Програмний комплекс OpenStack розрахований на розподіленість, велику кількість запитів та користувачів. Веб інтерфейс надає потужний функціонал як адміністратору, так і звичайному користувачу. При цьому наявна можливість гнучкого налаштування прав та обмежень кожного з користувачів, а також груп.

Також OpenStack має сервіси, які в комплексі дозволять зробити потужний self-hosted аналог Amazon ec2, який зарекомендував себе як зручна платформа для користувачів всього спектру професіоналізму. Так, можливо створити свої регіони розміщення, типи віртуальних вузлів, кількість виділених ресурсів, власні системні зображення та багато іншого.

Результат: + – так; - – ні; Враховується лише вбудований функціонал.

Забезпечення відмовостійкості	+
Система автентифікації, авторизації та обліку	+
Веб інтерфейс	+
Розгортання системи з образу	+
Створення віртуальних вузлів як з ОС Linux так і з Windows	+
Віддалений доступ до консолі та графічного інтерфейсу вузлів	+
Налаштування мережі вузла	+
Сервіс збирання артефактів та логів	+
Створення приватних мереж	+
Можливість автоматичного запуску команд на вузлі	-
Автоматичне створення та видалення вузла	+

Табл. 1.1.3.1 Таблиця відповідності Proxmox критеріям відбору програмного комплексу для віртуальної лабораторної інфраструктури

Система контейнерної оркестрації Kubernetes

kubernetes — це система оркестрування контейнерів з відкритим вихідним кодом. kubernetes автоматизує розгортання, масштабування та керування додатків в інфраструктурі. Він підтримує поширені середовища виконання контейнерів: cri-o, containerd та docker, але існують і неофіційні реалізації драйверів інших середовищ (наприклад rkt).

Сервер kubernetes керує вузлами та забезпечує спілкування з/між ними. Сам сервер складається з декількох компонентів, кожен з яких може бути розміщений на різній обчислювальній машині, а також в розподіленій інфраструктурі на декількох машинах для забезпечення надійності та доступності.

kubernetes api-сервер – ключовий компонент і обслуговує kubernetes api. Саме api використовує протокол http і використовує мову серіалізації даних. kubernetes api-сервер обробляє rest запити та оновлює дані у etcd.

etcd — легка розподілена база даних типу ключ-значення, яка зберігає налаштування кластеру та стан вузлів в будь-який момент часу. Etcd віддає перевагу узгодженості даних замість доступності, тобто при розділенні мережі між екземплярами etcd, ті екземпляри, які були відділені будуть видавати помилку замість тієї версії даних, яка збережена в них локально. Цей принцип відіграє критичну роль в правильному плануванні завдань та роботі вузлів. Api kubernetes використовує etcd для усунення розбіжностей стану конфігурації вузлів та їхнього моніторингу.

Планувальник — це сервіс, який контролює розміщення подів. Поди – це найменші, найпростіші об’єкти, які можна розгортати в kubernetes. Один под – це один додаток, запущений в інфраструктурі kubernetes. Кожен под місить один або більше контейнерів, які мають спільну мережу та можуть

мати спільні томи зберігання даних. Всі контейнери поду будуть розміщені на одному вузлі. Коли pod запускає кілька контейнерів, контейнери керуються як один суцільний об’єкт. Планувальник відстежує споживання ресурсів вузла і повинен гарантувати, що навантаження не буде більшим за потужність вузла, на якому воно запускається. Планувальник обчислює розміщення поду на вузлі відповідно до критеріїв, які включають як характеристики вузла так і користувацькі налаштування.

Менеджер контролерів — це сервіс, який порівнює поточний стан кластера з тим, що налаштований користувачем і виконує дії, для того, щоб обидва стани стали ідентичними. Менеджер контролерів використовує api-сервер для створення, зміни та видалення ресурсів, якими він керує. Одним із видів контролерів є контролер реплікації, який здійснює реплікацію та масштабування, запускаючи певну кількість копій поду на різних вузлах. Він також замінює поди, які вийшли з ладу новими. Інші контролери, які є частиною системи kubernetes, включають контролер демонів, який запускає по одному демону на кожному вузлі та контролер завдань для запуску подів, які виконують обчислення і завершуються наприкінці.

На вузлах в свою чергу запущені сервіси, до яких звертається сервер kubernetes для керування. Вони включають: kubelet – сервіс, який відповідає за стан вузла і повинен гарантувати, що всі поди, розміщені на ньому, працюють справно. Він займається запуском, зупинкою та підтримкою подів відповідно до вказівок серверу. kubelet автоматично перерозгортає под на тому ж самому вузлі у випадку внутрішньої помилки в екземплярі поду. Стан кожного поду повідомляється серверу кожні декілька секунд у вигляді heartbeat повідомлення. kube-proxy – це реалізація проксі та балансувальника,

він маршрутизує трафік до потрібного поду відповідно до ір адреси та порту вхідного запиту.

Результат: + – так; - – ні; Враховується лише вбудований функціонал та веб інтерфейс від розробників.

Забезпечення відмовостійкості	+
Система автентифікації, авторизації та обліку	-
Веб інтерфейс	+
Розгортання системи з образу	+
Створення віртуальних вузлів як з ОС Linux так і з Windows	+
Віддалений доступ до консолі та графічного інтерфейсу вузлів	-
Налаштування мережі вузла	+
Сервіс збирання артефактів та логів	+
Створення приватних мереж	-
Можливість автоматичного запуску команд на вузлі	-
Автоматичне створення та видалення вузла	+

Табл. 1.1.4.1 Таблиця відповідності Kubernetes критеріям відбору програмного комплексу для віртуальної лабораторної інфраструктури

В результаті проведеного аналізу можливо сформувати таку порівняльну таблицю:

	oVirt	Proxmox	OpenStack	kubernetes
Забезпечення відмовостійкості	+	+	+	+
Система автентифікації, авторизації та обліку	+	+	+	-
Веб інтерфейс	+	+	+	+
Розгортання системи з образу	+	+	+	+
Створення віртуальних вузлів як з ОС Linux так і з Windows	+	+	+	+
Віддалений доступ до консолі та графічного інтерфейсу вузлів	+	+	+	-
Налаштування мережі вузла	-	-	+	+
Сервіс збирання артефактів та логів	-	-	+	+
Створення приватних мереж	-	-	+	+
Можливість автоматичного запуску команд на вузлі	-	-	-	-
Автоматичне створення та видалення вузла	-	+	+	+

Табл. 1.1.1 Порівняння функціоналу програмних комплексів

Провівши аналіз функціоналу цих програмних комплексів виявилося, що для задоволення всіх потреб доцільно використати програмний комплекс OpenStack, оскільки він максимально покриває список вимог до віртуальної лабораторної інфраструктури. Такий вибір дозволить задовольнити всі

потреби з мінімальною кількістю доповнень. Відповідно до наведених вимог та обраного програмного комплексу, необхідно розробити та розгорнути систему, яка дозволить:

- Задовольнити всі потреби наведених шляхів використання
- Забезпечити безвідмовність системи
- Надати можливість подальшого оновлення системи
- Створити резервні копії важливих системних даних

2 Архітектура віртуальної лабораторної інфраструктури

Виходячи з розуміння архітектури взаємозв'язку між сервісами OpenStack, доцільно обрати таку архітектуру:

- 3-5+ (кворум) вузлів ectd
- 3-5+ (кворум) вузлів pgpoolII+postgresql
- 3-5+ (кворум) вузлів rabbitmq
- 2+ http, api-сервісів Horizon Dashboard, Glance Image, swift-proxy, swift-account, swift-container, neutron-server.
- 2+ вузли зберігання даних swift-object, залежно від кількості вузлів з зберігаючими пристроями, або з мережевим доступом до них.
- 2+ neutron-l3-agent з відмовостійкістю за допомогою vrrp.
- nova-compute, neutrol-l2-agent, Nova Compute на кожному обчислювальному вузлі.
- Локальний екземпляр Haproxy на кожному вузлі, який звертається до rest api задля забезпечення балансування навантаження та відмовостійкості та відсутності єдиної точки відмови.

Така архітектура дозволить забезпечити:

- Віртуальну vxlan мережу з віртуальними інтерфейсами та мережевим обладнанням за допомогою сервісу Neutron Networking та системи Open Virtual Network.
- Зручний веб-інтерфейс для адміністрування віртуальної інфраструктури за допомогою сервісу Horizon Dashboard.
- Створення віртуальних машин та контейнерів за допомогою сервісу Nova Compute.
- Обмеження використання ресурсів та їх статистика за допомогою сервісу Nova Compute.
- Потужна система ідентифікації користувачів та сервісів Keystone Identity.
- Висока доступність в разі виходу з ладу обладнання або вимкнення його на обслуговування.
- Модульна архітектура програмного комплексу OpenStack надає можливість додавати інші сервіси для розширення функціоналу.

Для розгортання віртуальної лабораторної інфраструктури було обрано таку конфігурацію:

- 5 вузлів контролерів, які містять etcd, pgpoolII+gostgresql, etcd, swift-proxy, Horizon Dashboard, Glance Image та Nova Compute.
- 4 мережевих вузли з Neutron Networking та Nova Compute.
- 4 вузли сховища, які містить лише Swift Object Storage.
- 2 обчислювальні вузли, які містить лише Nova Compute.

Забезпечення відмовостійкості

Однією з важливих вимог є відмовостійкість, забезпечення якої є фактором, який вносить свої корективи в побудову системи. Так, програмний комплекс OpenStack містить велику кількість незалежних stateless сервісів, а також певний набір основних statefull сервісів, які зберігають всю інформацію про налаштування та стан інфраструктури.

До statefull сервісів, необхідних для роботи OpenStack відносяться [1]:

- SQL база даних
- Система обміну повідомленнями
- etcd
- memcached

Всі інші сервіси входять в програмний комплекс OpenStack, і або є stateless сервісами, або мають вбудовані налаштування для забезпечення високої доступності (на приклад Neutron), або заздалегідь передбачено встановлення декількох екземплярів сервісу.

Задля забезпечення відмовостійкості stateless сервісів можливо використовувати такі засоби:

- Vrrp
- HTTP load balancing
- kubernetes

Враховуючи попередні спостереження, найкращим варіантом є використання http балансування за допомогою Haproxy. kubernetes також

підходить у якості інструменту для забезпечення високої доступності, оскільки він також виконує балансування між репліками контейнеру, але більшість додаткового функціоналу не буде використано в описаному застосуванні.

Розподілене сховище etcd

Розподілене сховище конфігурації etcd використовується для зберігання інформації типу ключ-значення сервісами OpenStack. OpenStack зберігає в ньому інформацію про блокування, конфігурацію, дані про працездатність сервісів, тощо [1]. Цей сервіс є одним з базових, без якого не можливе функціонування OpenStack.

Відмовостійкість etcd забезпечується за допомогою алгоритму консенсусу raft. Таким чином сховище стійке до будь-яких проблем з мережею чи відключення вузлів. Функціонування кворуму між вузлами можливе за правильної роботи більшості вузлів.

Розмір кластеру	Більшість	Відмовостійкість
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

Табл. 2.2.1 Відношення кількості вузлів до відмовостійкості кластера [9]

Виходячи з правил кворуму, доцільно використати як мінімум три вузли etcd. Таке використання забезпечить можливість вимкнення одного вузла на обслуговування зі збереженням працездатності системи. Розробниками etcd рекомендується розгортання системи з п’ятьма вузлами, і відповідно можливістю одночасної несправності (або обслуговування) двох з них.

Використання парної кількості вузлів не приносить користі, при цьому робить мережу вразливою до розділення навпіл, тому не варто його застосовувати.

SQL кластер з використанням pgpoolII

Більшість служб OpenStack використовують базу даних SQL для зберігання інформації [1]. База даних зазвичай працює на вузлі контролера. Програмний комплекс OpenStack підтримує такі популярні SQL бази даних як mariadb, mysql, postgesql та інші. SQL база даних необхідна для функціонування сервісів OpenStack.

Для всіх вище наведених баз даних існують системи кластеризації, а також програмне забезпечення для балансування запитів між вузлами кластеру.

Найбільше для забезпечення потреби у відмовостійкій SQL базі даних підходить postgresql з балансувачем pgpoolII. Таке поєднання забезпечує відмовостійкість, синхронну реплікацію між вузлами а також балансування навантаження. На додачу PgpoolII забезпечує кешування запитів та збереження відкритих з'єднань, що дозволяє оптимізувати доступ до бази даних.

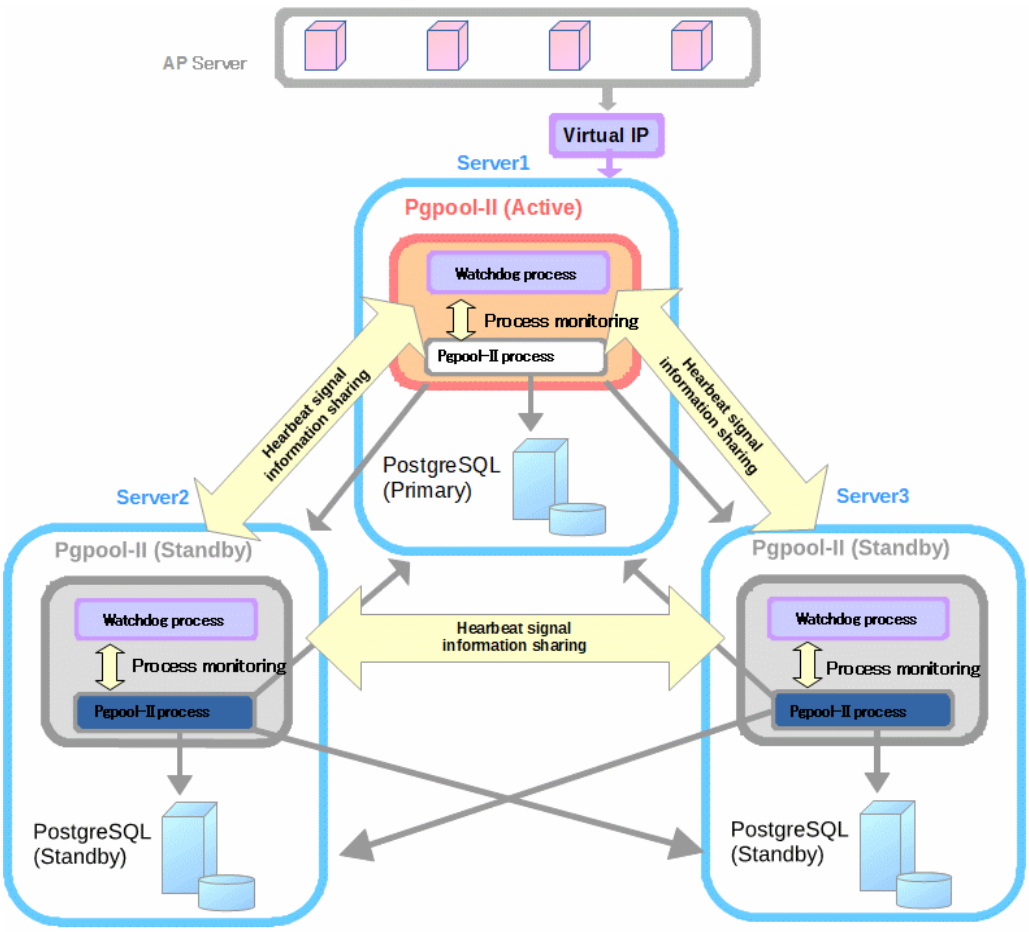

Рис 2.3.1 Архітектура кластера pgpoolII [7]

Лідер кластеру pgpoolII отримує встановлену в конфігурації ip адресу за допомогою vrrp і отримує всі запити, які приходять до кластеру. Запити на читання балансуються між другорядними вузлами, а запити на запис завжди виконуються лише на вузлі лідері з подальшою реплікацією на інші вузли.

PgpoolII використовує сторожовий процес, який спостерігає за процесом балансувача pgpool на локальній машині, а також отримує інформацію про стан балансувача на інших вузлах. Коли сторожовий процес бачить несправність головного вузла, він ініціює вибори нового лідера [7].

Вибори лідера відбуваються за правилами кворуму, тому застосування pgpool має ті самі умови по кількості вузлів що і etcd. Головна проблема, яку

вирішує використання кворуму — це split-brain. За допомогою кворуму забезпечується узгодженість даних. При цьому самі дані не перевіряються за допомогою кворуму, оскільки після узгодження лідера

Система обміну повідомленнями Rabbitmq

OpenStack використовує чергу повідомлень для координації операцій та інформації про стан між службами. Програмний комплекс OpenStack підтримує такі системи обміну повідомленнями: rabbitmq, qpid та zeromq. Рекомендованою розробниками є rabbitmq. OpenStack не може функціонувати без системи обміну повідомленнями, тому вона є необхідним компонентом [1].

Rabbitmq має функціонал для забезпечення високої доступності. Наявний режим кворуму, який є відносно новим алгоритмом для даного програмного забезпечення. Кворум заміняє раніше використовуваний алгоритм віддзеркалених черг. Кворум в rabbitmq базується на тому самому алгоритмі raft, що і кворум в etcd [8]. Тому всі умови використання etcd також застосовуються і до rabbitmq. Застосування raft дозволило кардинально спростити реплікацію в rabbitmq, при цьому також збільшилася надійність та узгодженість даних.

Також наявна можливість роботи у випадку split-brain, але при з’єднанні кластеру всі повідомлення з однієї частини будуть знищені. Таке застосування дозволяє збільшити доступність кластеру зменшивши при цьому узгодженість даних та виключає гарантовану доставку.

Сервіс кешування даних Memcached

В програмному комплексі OpenStack сервіс кешування Memcached використовується для зберігання токенів авторизації в системі ідентифікації [1]. На відміну від інших stateful сервісів, для memcached збереження даних не є необхідністю, оскільки вони не є критичними. Доступ до memcached здійснюється за допомогою бібліотеки openstack oslo.cache, яка автоматично вибирає із списку серверів перший робочий, і обробляє запит на ньому. Висока доступність забезпечується за допомогою бібліотеки та двома або більше серверами memcached.

Веб-інтерфейс користувача Horizon Dashboard

Horizon — це стандартна реалізація веб інтерфейсу для OpenStack. Цей інтерфейс надає можливість зручного налаштування і моніторингу інших сервісів, таких як: Keystone, Swift, Nova, тощо [5].

Horizon надає корисний веб інтерфейс, за допомогою якого користувачі та адміністратори можуть, відповідно до наданих їм прав, створювати, видаляти та налаштовувати віртуальні машини/контейнери. Також багато адміністративних налаштувань доступні з веб інтерфейсу. При цьому це все організовано в зручному форматі в стилі Amazon ec2.

За замовчуванням Horizon надає три центральні інформаційні панелі: інформаційна панель користувача, Системна інформація та Налаштування. Ці панелі охоплюють основний функціонал OpenStack. Horizon Dashboard складається з одного сервісу з підтримкою плагінів.

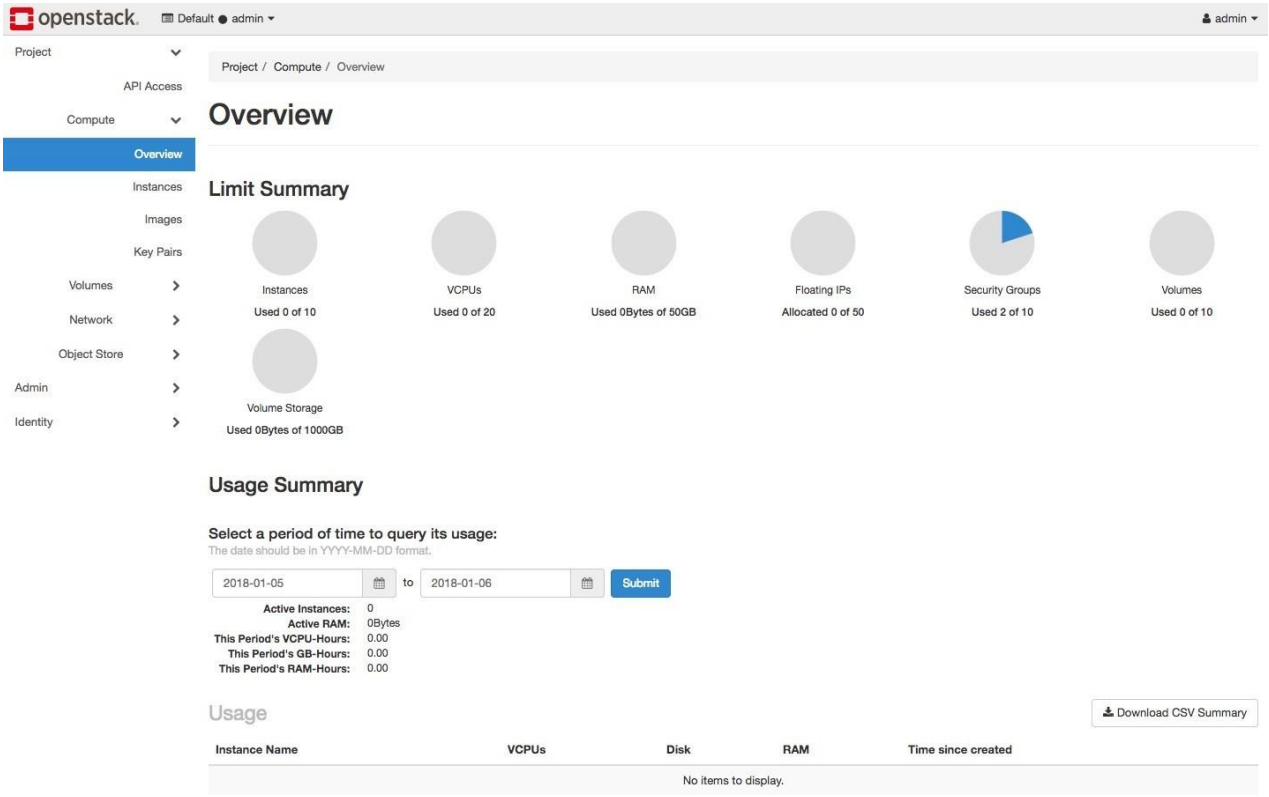

Рис 2.6.1 Веб інтерфейс OpenStack Horizon

Сервіс зображень Glance Image

Сервіс Glance Image забезпечує виявлення, реєстрацію та отримання зображень віртуальних машин та контейнерів. Glance має api, яке дозволяє запитувати метадані зображення, а також завантажувати самі зображення. Усі операції з метаданими та зображеннями виконуються за допомогою бібліотеки glance_store, яка відповідає за взаємодію із зовнішніми сховищами або локальною файловою системою. Бібліотека glance_store забезпечує єдиний інтерфейс для доступу до серверних сховищ. Glance використовує центральну базу даних Glance DB, яка використовується спільно всіма компонентами системи [4].

Glance може бути використаний для створення зображеннь-заготовок для навчальних демонстрацій, зображень стандартних конфігурацій

програмного забезпечення для практики його адміністрування та чистих зображень з популярними операційними системами для створення віртуальних вузлів.

Glance Image складається з двох сервісів, glance та glance-store [4]. Така архітектура дозволяє змінювати сервіс, за допомогою якого здійснюється доступ до сховища даних. Хоч сервіс і розділений, але доцільно використовувати обидва компоненти на одному вузлі, оскільки glance-store просто забезпечує інтерфейс до сховища.

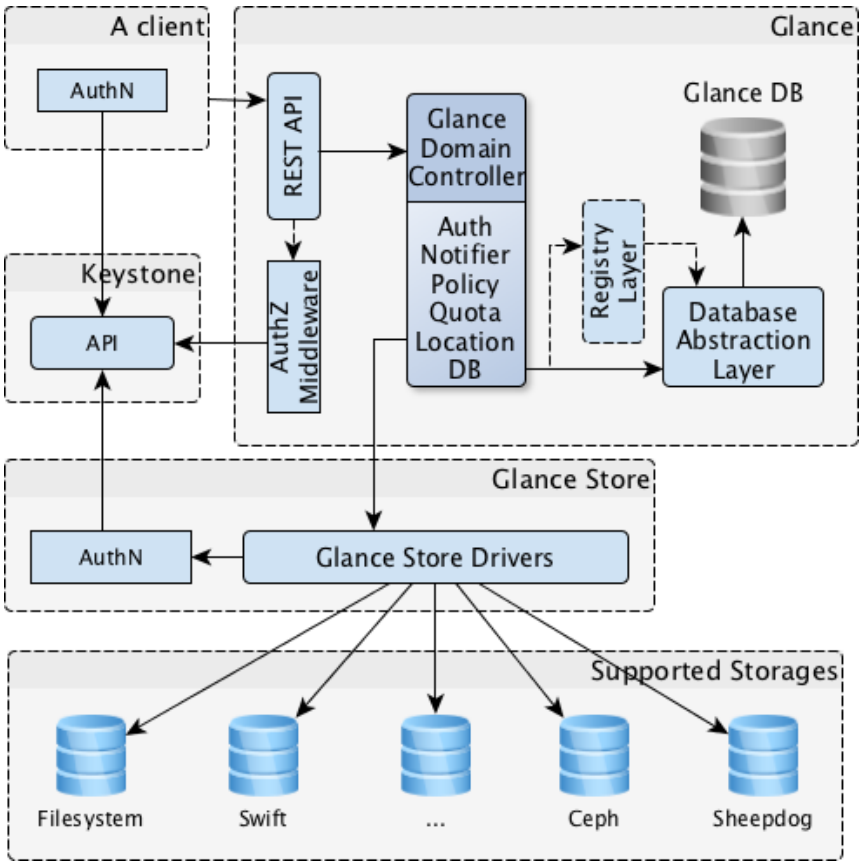

Рис 2.7.1 Архітектура OpenStack Glance Image [4]

Мережева система Neutron Networking

Neutron Networking є окремим компонентом модульної архітектури OpenStack. Він має визначальну роль у створенні віртуальної інфраструктури. Neutron розгортання декількох екземплярів сервісів на різних хостах.

Neutron стане корисним у виконанні робіт, які вимагають специфічну мережеву конфігурацію або приватну мережу з декількома вузлами. Також за допомогою Neutron Networking можливо видавати віртуальним вузлам публічні ip адреси, що дозволить мати доступ до розроблених веб-застосунків через стандартний інтернет-браузер.

Neutron Networking використовує neutron-server, для того шоб надати rest api для налаштування інших сервісів-компонентів Neutron Networking. neutron-server використовує плагін для доступу до бази даних для Neutron DB.

Neutron складається з таких сервісів [6]:

- neutron-server — rest api, яке налаштовує агенти у відповідності з запитами.
- plug-in agent — запускається на кожному гіпервізорі для виконання локальної конфігурації vSwitch.
- neutron-dhcp-agent — надає функціонал dhcp.
- neutron-l3-agent — забезпечує L3-forwarding та NAT для доступу до зовнішньої мережі з віртуальних машин Nova Compute.
- neutron-metering-agent — забезпечує облік трафіку для віртуальних мереж

Взаємозв’язок між окремими компонентами відбувається за допомогою rpc. Для управління конфігурацією використовується rest api. На кожен обчислювальний вузол встановлюється plug-in agent. Можливо встановити декілька dhcp та l3 агентів для забезпечення відмовостійкості.

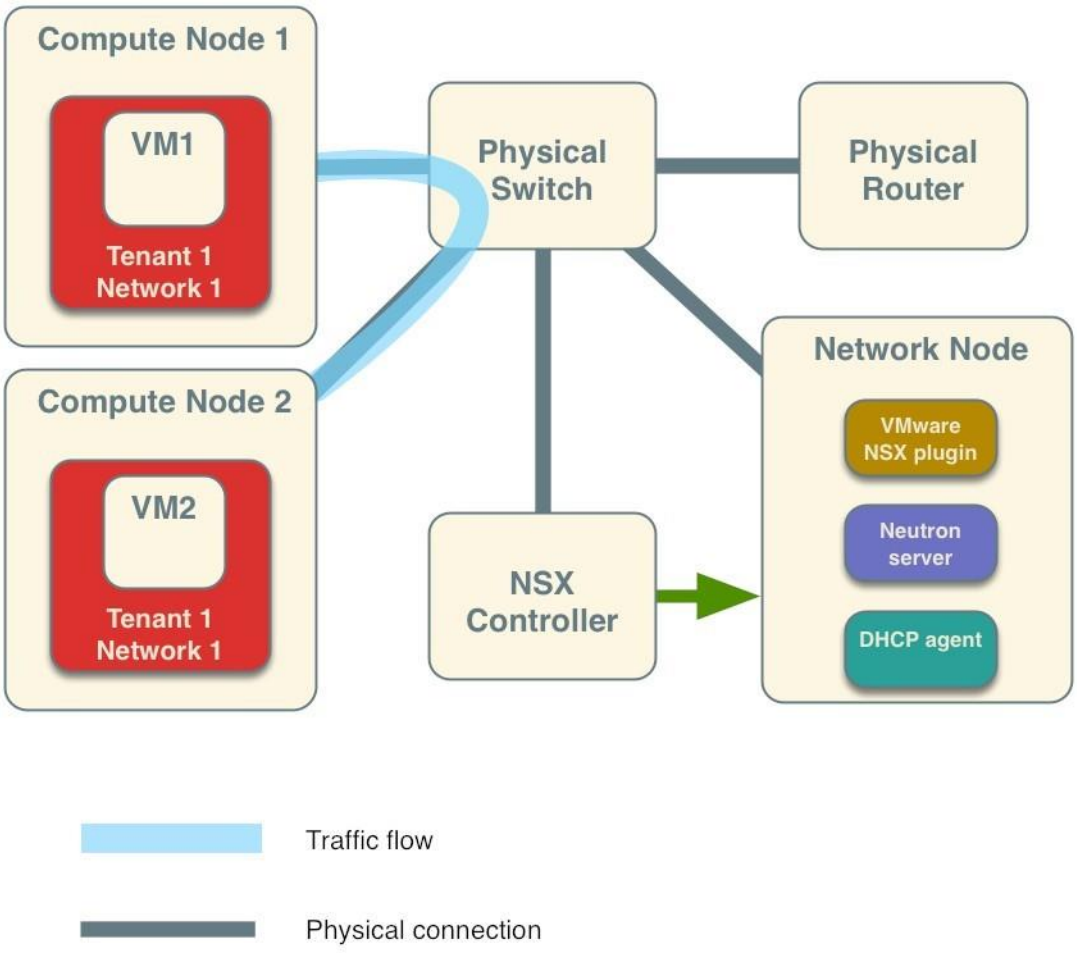

Рис 2.8.1 Архітектура Neutron Networking [6]

Сервіс збереження неструктурованих даних Swift Object Storage

Swift Object Storage — це високо-доступне, розподілене, узгоджене сховище об’єктів/блобів. Організації можуть використовувати Swift для ефективного, безпечного та дешевого зберігання великої кількості даних. Swift використовує Ring алгоритм, який забезпечує розподіленість системи. За допомогою цього алгоритму визначається, на якому з серверів міститься затребуваний файл.

Swift Object Storage слугує сховищем даних для різних сервісів, включно з Glance Image. Також за допомогою цього сервісу користувачі зможуть зберігати і отримувати будь-які файли.

Swift Object Storage складається з таких сервісів [3]:

- swift-proxy — обробляє вхідні запити від клієнта, наприклад читання, запис та видалення об’єктів. Swift-proxy визначає на яких серверах знаходяться дані за допомогою Ring алгоритму.
- swift-account — зберігає інформацію про акаунти.
- swift-container — зберігає інформацію про контейнери.
- swift-object — забезпечує читання та запис об’єктів. Також з певним інтервалом перевіряє хеш-суми об’єктів на диску, і виконує реплікацію в разі їх пошкодження.

Сервіс обчислень Nova Compute

Nova Compute забезпечує взаємодію з гіпервізором та двигуном контейнеризації на обчислювальних вузлах OpenStack. За допомогою Nova створюються віртуальні машини та контейнери, а також підключаються до віртуальної мережі Neutron Networking. Nova складається з декількох сервісів,

кожен з яких виконує різні функції. Nova має rest api, зі складовими середовища Nova api-сервер зв’язується за допомогою rpc.

Nova є основним сервісом, який надає можливість запускати віртуальні машини та контейнери, що є найголовнішою вимогою до віртуальної інфраструктури.

Nova Compute складається з таких сервісів [2]:

- nova-api — надає rest api для зв’язку з іншими сервісами Nova Compute.
- nova-compute — відповідає за створення образу диска, запуск його через базовий драйвер віртуалізації, повідомляє про стан запущених віртуальних вузлів. Також забезпечує підключення сховища до віртуальних вузлів.
- nova-conductor — забезпечує координацію та підтримку запитів до бази даних для Nova.
- nova-scheduler — відповідає за вибір обчислювального вузла для запуску віртуального обчислювального вузла.
- nova-serialproxy — надає websocket проксі, за допомогою якого можна отримати доступ до послідовного порта віртуального обчислювального вузла Nova.
- nova-libvirt — драйвер, який задовольняє запити nova-compute на створення, оновлення, видалення та налаштування віртуальних машин. Також існують інші драйвери, які базуються на технологіях, відмінних від libvirt.

Архітектура взаємозв'язків між компонентами сервісів OpenStack

В результаті аналізу вимог, які повинна задовольняти віртуальна лабораторна інфраструктура, а також умов, які створює необхідність забезпечення високої доступності, я пропоную використати таку архітектуру:

- Сервіси, від яких залежить програмний комплекс OpenStack (в цьому випадку postgresql+pgpoolII, rabbitmq, etcd та memcached) розмістити в контейнерах на кожному з керуючих вузлів.
  - etcd використовує кворум, звертатися можна до будь-якого вузла кластеру, відповідно можливе балансування навантаження.
  - pgpoolII використовує кворум, звертатися можна до будь-якого вузла кластеру, сам pgpoolII слугує баланувачем навантаження між вузлами postgresql
  - rabbitmq використовує кворум, звертатися можна до будь-якого вузла кластеру, відповідно можливе балансування навантаження.
  - memcahced не потребує балансування, оскільки воно здійснюється бібліотекою oslo.cache, яку використовують сервіси OpenStack.
- Запити до stateless сервісів OpenStack можуть бути балансовані за допомогою Haproxy. Ці сервіси включають: cinder-api, zun-api, vitrage-api, watcher-api, trove-api, sahara-api, octavia-api, nova-api, solum-api, placement-api, senlin-api, manila-api, monasca-api, mistral-api, murano-api, barbican-api, glance-api, masakari-api, magnum-api, ironic-api, gnocchi-api, freezer-api, heat-api, heat-api, designate-api, cyborg-api, cloudkitty-api, aodh-api, blazar-api та інші.
- Деякі stateful сервіси OpenStack мають вбудовану систему забезпечення високої доступності, наприклад neutron-l3-agent, який використовує keepalived для підтримання роботи сервісу в режимі active/pasive. Є

сервіси, розраховані на високу доступність, як Swift Object Storage, який має власну архітектуру серверів та проксі. Метод забезпечення високої доступності залежить від сервісу.

- Всі інші stateful сервіси OpenStack потребують або розміщення в кількості 1 шт. або кластеризацію в режимі active/pasive.

Така конфігурація дозволить забезпечити найбільш рівномірне розподілення навантаження між серверами інфраструктури та надасть переваги високої доступності для сервісів, для яких це можливо.

Повна цілісна схема архітектури системи дуже важка для розуміння, тому було прийнято рішення про розділення її на логічні частини. Всі схеми зображені з розрахунку на архітектуру з високою доступністю. Кількість розміщених сервісів, а також вузли, на яких вони розміщуються може варіюватися в залежності від застосування.

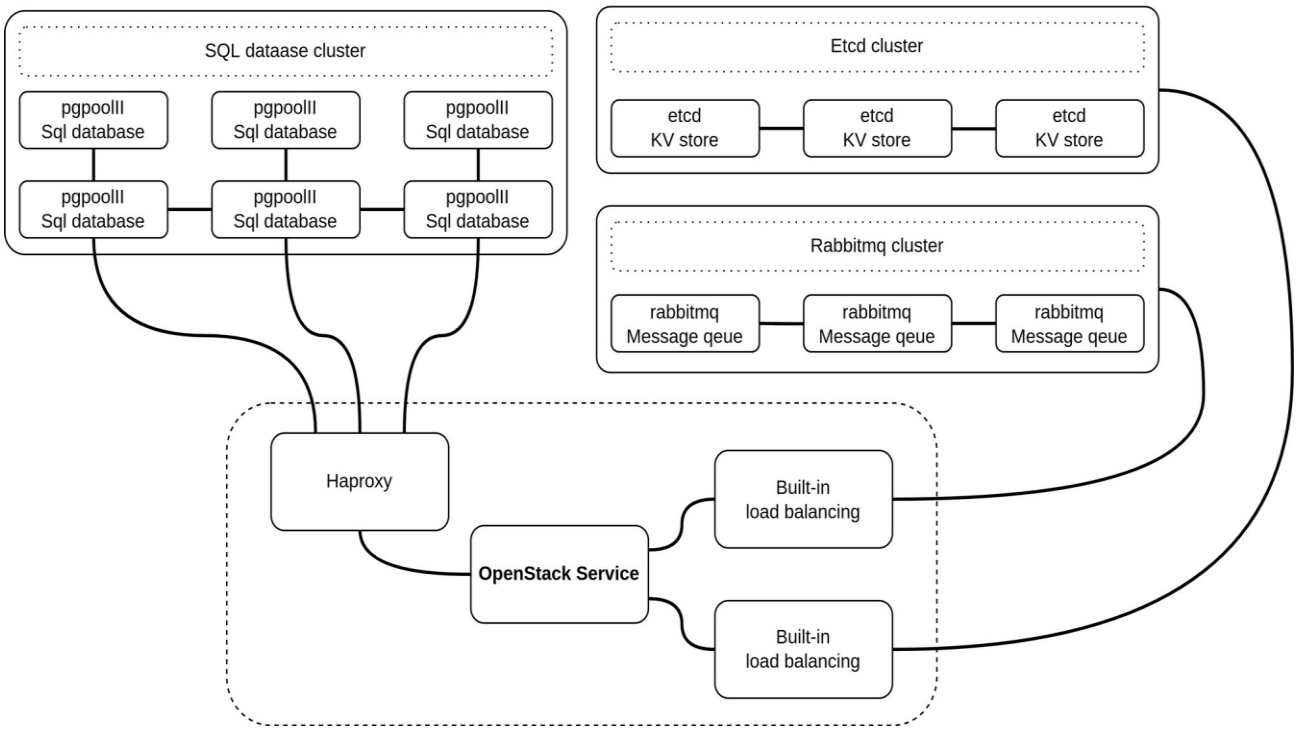

Рис 2.11.1 Типове підключення сервісу OpenStack

Для підключення до базових сервісів, сервіс OpenStack використовує вбудовані балансувачі rabbitmq та etcd, з'єднання з кластером SQL бази даних використовується локальний екземпляр Haproxy, який балансує навантаження між екземплярами pgpoolII.

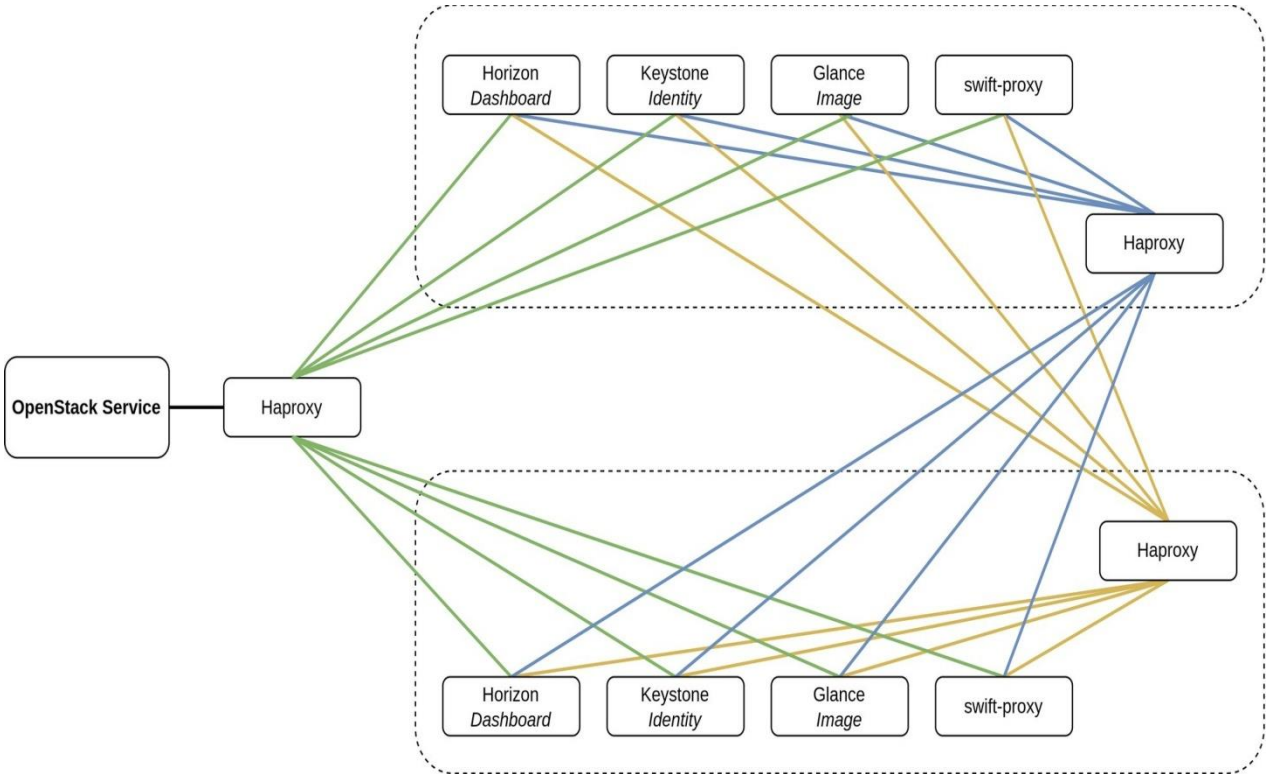

Рис 2.11.2 Балансування rest api за допомогою haproxy

Для забезпечення високої доступності rest api-сервісів використовується локальний екземпляр Haproxy, який балансує запити, а також перевіряє доступність вузлів призначення. Такий підхід дозволяє не використовувати централізований екземпляр Haproxy, як єдину точку відмови.

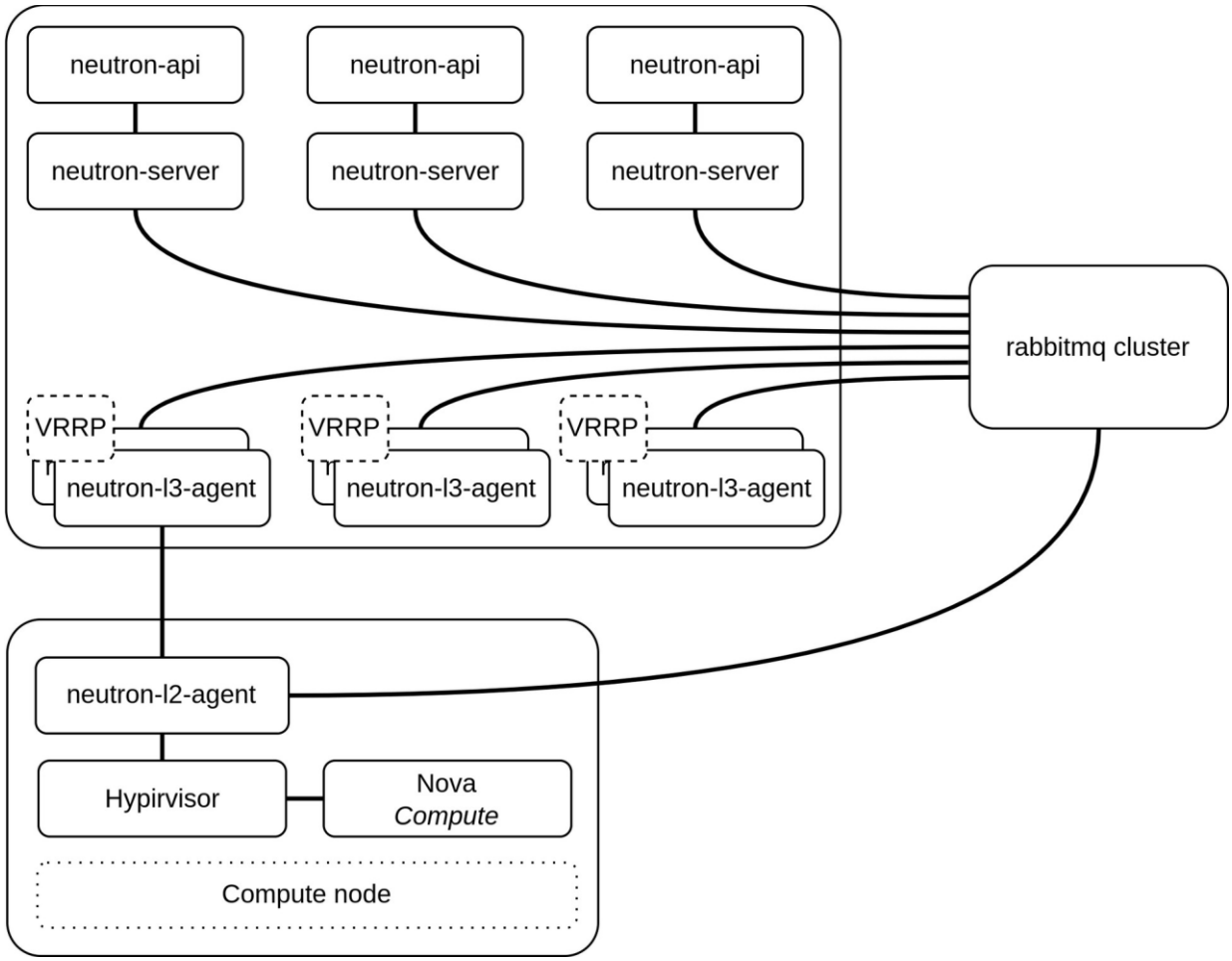

Рис 2.11.3 Архітектура сервісу Neutron Networking

Архітектура Neutron Networking вимагає встановлення L2 агентів на серверах з гіпервізором, який контролюється за допомогою Nova Compute. Сам L2 агент налаштовує віртуальний комутатор (наприклад Open vSwitch), який перенаправляє трафік до віртуального роутера — L3 агента. При цьому самих L3 агентів може бути декілька, а відмовостійкість кожного з них може забезпечуватися різними плагінами в режимі active/pasive. На даний момент існує лише один такий плагін, який використовує VRRP. З'єднання до neutron-api балансується екземпляром Haproxy, розташованим на клієнтській машині.

Neutron-server з'єднується з L3 та L2 агентами за допомогою rpc, надсилаючи повідомлення через rabbitmq кластер.

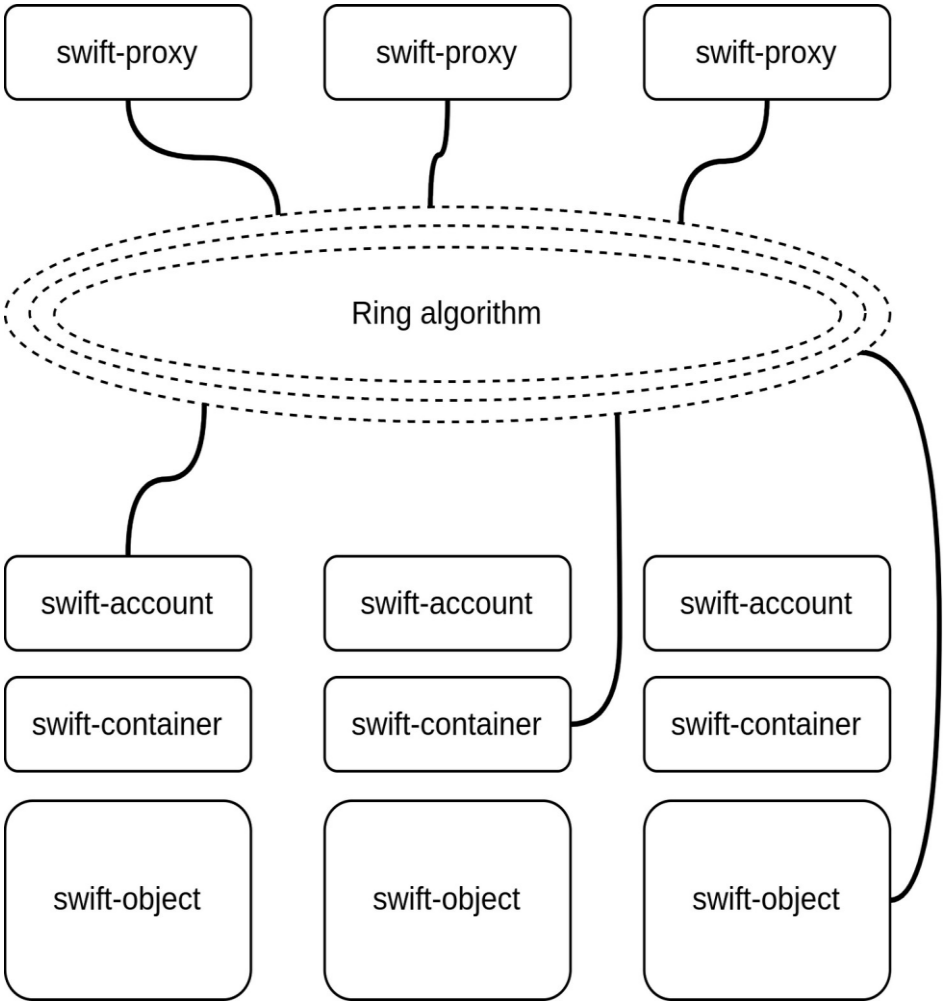

Рис 2.11.4 Архітектура сервісу Swift Object Storage

OpenStack Swift Object Storage розроблений з високою доступністю, реплікацією, балансуванням навантаження та масштабованість. Відповідно для забезпечення вимог віртуальної лабораторної інфраструктури не потрібно робити нічого особливого. Swift розгортається в стандартному виконанні, висока доступність забезпечується за допомогою декількох екземплярів кожного із сервісів-компонентів Swift Object Storage.

3 Розгортання та тестова реалізація інфраструктури

Особливості практичної реалізації

Встановлення програмного комплексу OpneStack детально описано на офіційному сайті (https://docs.openstack.org). При цьому також існує інструкція по забезпеченню високої доступності, але вона не повна, застаріла та використовує не оптимальні методи забезпечення доступності. Через це інформацію про забезпечення високої доступності доводиться брати з відкритих проектів, наприклад kolla-ansible (https://github.com/openstack/kolla-ansible) або Highly Available OpenStack: From Theory to Reality (https://www.youtube.com/watch?v=-ya-GMwP068). Обидва джерела інформації дають простір для креативності у імплементації власного високо-доступного середовища.

Сервіси OpenStack у вигляді контейнерів можна знайти на сайті quay.io/openstack.kolla або створити їх за допомогою openstack-kolla (https://github.com/openstack/kolla).

Фізична архітектура інфраструктури

В наявності маємо 16 серверів, симетрично розподілених між двома серверними стійками.

Сумарні характеристики:

- 120 обчислювальних ядер x86
- 832 ГБ пам’яті DDR2

Між двома серверами Dell PowerEdge 2950 повинно бути розміщено керуючі компоненти віртуальної інфраструктури та мережеве сховище постійного зберігання даних у високо-доступній відмовостійкій конфігурації. Всі інші сервери використовуватимуться як обчислювальні вузли віртуальної інфраструктури. Для забезпечення сховища постійного зберігання даних віртуальних вузлів використовуватиметься дисковий масив з інтерфейсом fibre channel.

Кожен сервер має однорідну конфігурацію підключення мережі, сховища та KVM, лише конфігурація підключення менеджмент портів різна, оскільки вони відсутні на серверах Supermicro.

Список обладнання:

- Dell PowerEdge 2950 2шт.
- Supermicro X7DBR-8 4шт.
- HP ProLiant DL360 G5 6шт.
- SuperMicro X7SBI 2шт.
- Cisco Catalyst WS-C2950T-24 2шт.
- Cisco Catalyst WS-C3750G-24TS-S1U 2шт.
- Brocade Silkworm-3200 2шт.
- D-Link DKVM-16 1шт.

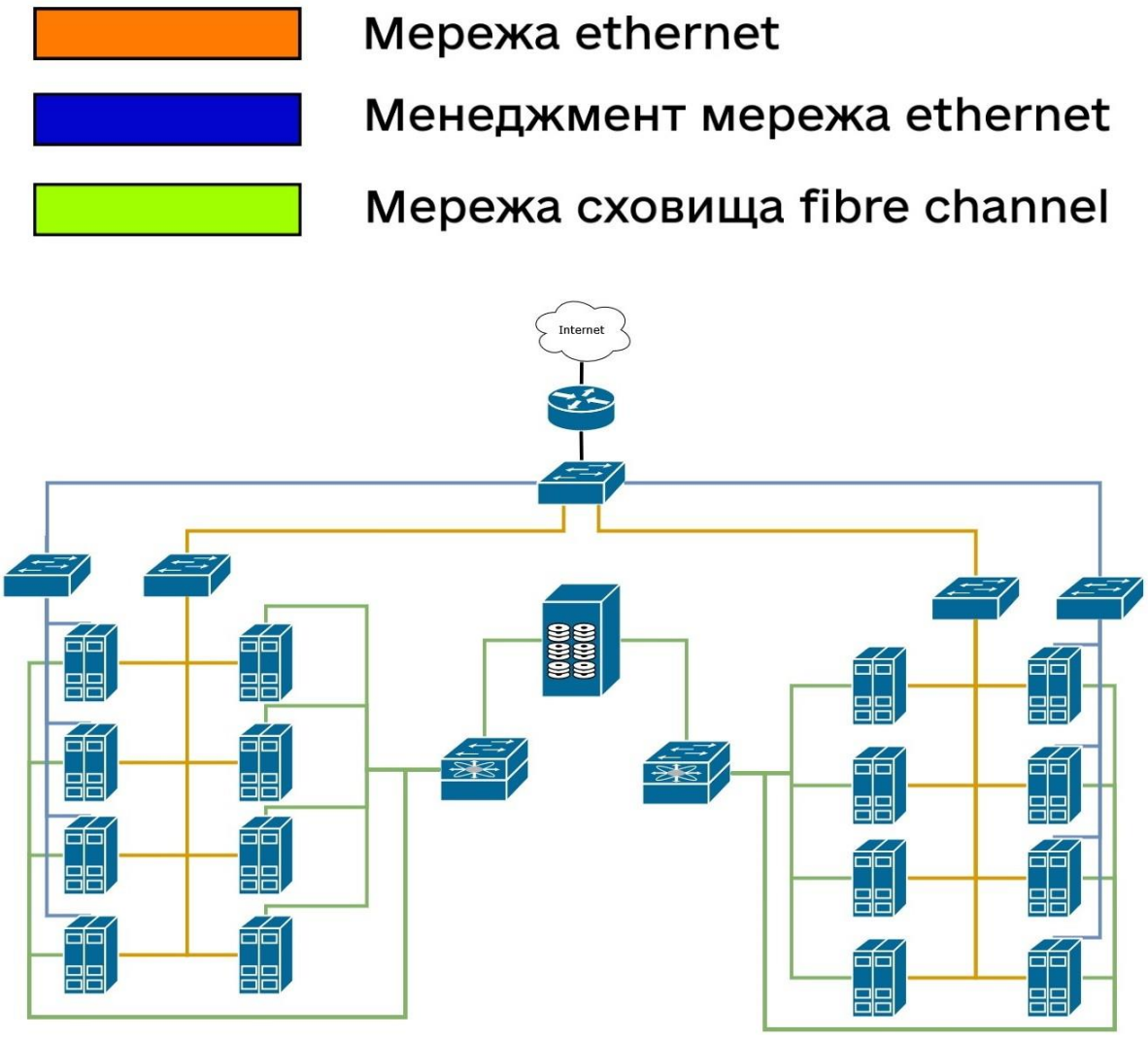

Рис 3.2.1 Логічна архітектура мережевих з'єднань інфраструктури

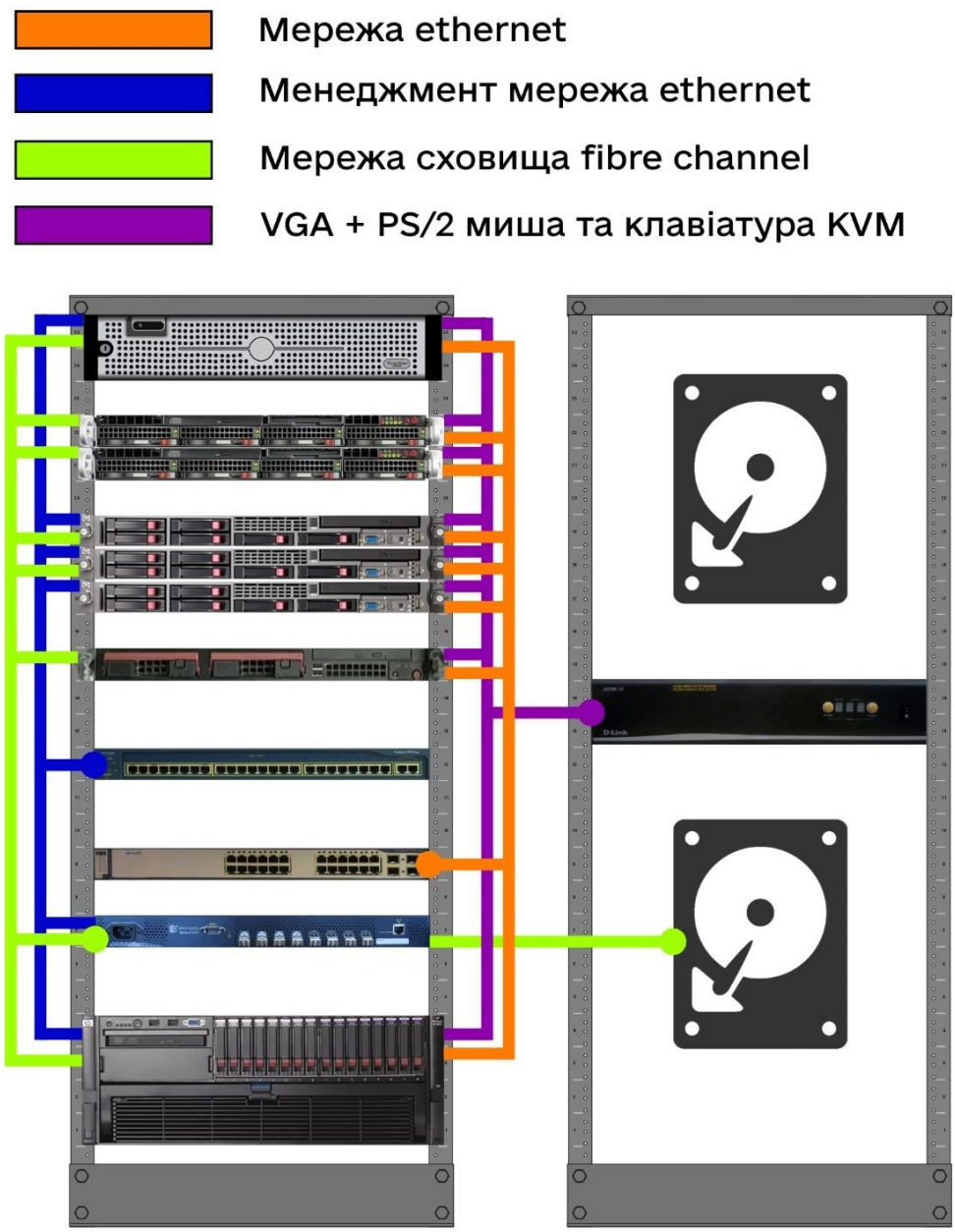

Рис 3.2.2 Фізичне обладнання інфраструктури, в наявності дві стійки з серверами з однаковою конфігурацією (на рисунку зображена одна стійка)

Встановлення та конфігурація

Для спрощення встановлення програмного комплексу OpeStack на сервери існує автоматизований інструмент на базі Ansible та Python — kolla-ansible (https://github.com/openstack/kolla-ansible). Даний інструмент надає можливість налаштування всього процесу конфігурації, вибрати сервери, на яких розміщуватимуться певні типи вузлів, наприклад compute чи storage. Вся конфігурація відбувається в файлі inventory, який слугує джерелом інформації, яке використовується разом з файлами завдань та файлом глобальних налаштувань програмним комплексом Ansible, для автоматизації процесу встановлення. Kolla-ansible надає інструменти для встановлення, оновлення та видалення кластерів OpenStack. Ці інструменти надають можливість зберігати всю конфігурацію кластера в двох файлах. За потреби можливо внести зміни в файли конфігурації і оновити кластер. Такий підхід дозволяє кардинально зменшити кількість часу, який оператор витрачає на його встановлення та налаштування.

Даний kolla-ansible доступний для rhel та debian подібних дистрибутивів [11]. Для встановлення на всіх вузлах використовувався дистрибутив Rocky Linux 8.5.

```
$ dnf install python3-devel libffi-devel gcc openssl-devel python3-libselinux
$ pip3 install ansible
$ pip3 install git+https://opendev.org/openstack/kolla-ansible@master
$ mkdir -p /etc/kolla
$ cp -r /usr/local/share/kolla-ansible/etc_examples/kolla/* /etc/kolla
$ cp /usr/local/share/kolla-ansible/ansible/inventory/* /etc/kolla
$ kolla-ansible install-deps
$ kolla-genpwd
# Налаштування файлу globals.yaml та multinode (файл inventory)
```

```
$ kolla-ansible -i ./multinode bootstrap-servers
$ kolla-ansible -i ./multinode prechecks
$ kolla-ansible -i ./multinode deploy
$ kolla-ansible post-deploy
```
[11]

В результаті маємо автоматично розгорнуту та налаштовану віртуальну лабораторну інфраструктуру на базі програмного комплексу OpneStack. Далі адміністратор створює необхідних користувачів та групи, вводить обмеження на використання ресурсів, додає базові зображення віртуальних машин та контейнерів, тощо.

Процес використання

Розглянемо процес використання віртуальної інфраструктури на базі програмного комплексу OpenStack. Так, через веб інтерфейс можливо зробити такі налаштування:

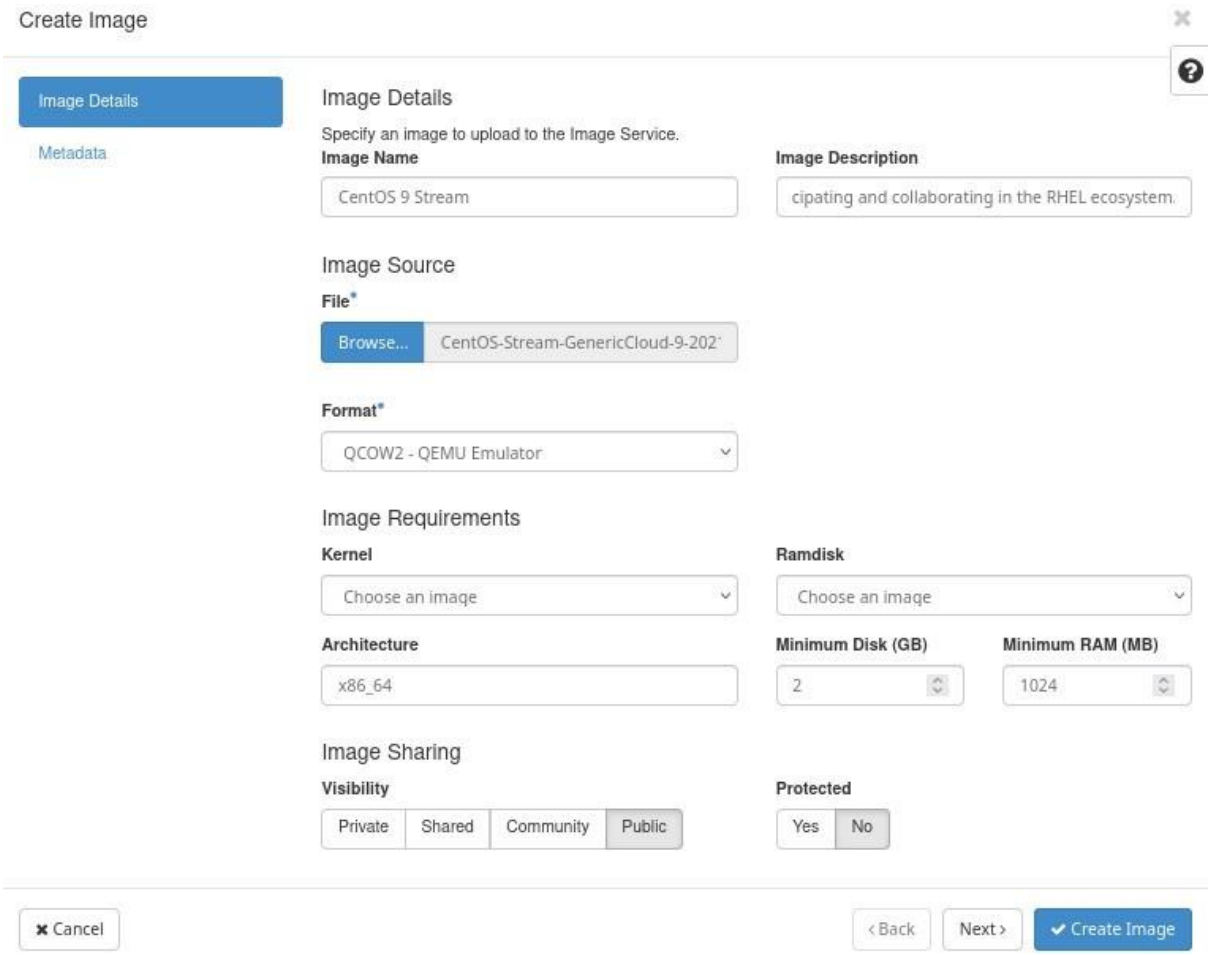

Рис 3.4.1 Створення зображення операційної системи

Наявна можливість створювати зображення операційної системи для подальшого використання. Заздалегідь заготовані зображення дозволять пришвидшити виконання лабораторних робіт, в яких вимагається адміністрування готової архітектури. Також повинен бути набір з стандартних

чистих операційних систем Windows, CentOS, Debian, тощо. Такі зображення знадобляться при виконанні майже всіх лабораторних робіт. Також створення зображень дозволить швидко розгортати навчальні демонстрації, які потрібні лише на короткий період часу.

В інтерфейсі створення зображення є такі налаштування:

- Name — назва зображення, яка відображатиметься користувачу.
- Description — опис зображення.
- File — саме зображення операційної системи.
- Format — формат файлу зображення. Варіантами є iso, ploop, qcow2, raw, vdi, vhd, vmdk, aki, ami або ari. Тип підтримуваного зображення залежить від гіпервізора.
- Kernel, ramdisk — ядро та зображення пам’яті, які використовуватимуться в разі запуску на bare-metal сервері, а не через гіпервізор.
- Наявні налаштування відображення зображення іншим користувачам. Можливо зробити зображення приватним, публічним, доступним тільки деяким користувачам та групам.
- Metadata — додаткова інформація, яка буде використовуватись для правильного розгортання зображення та його роботи.

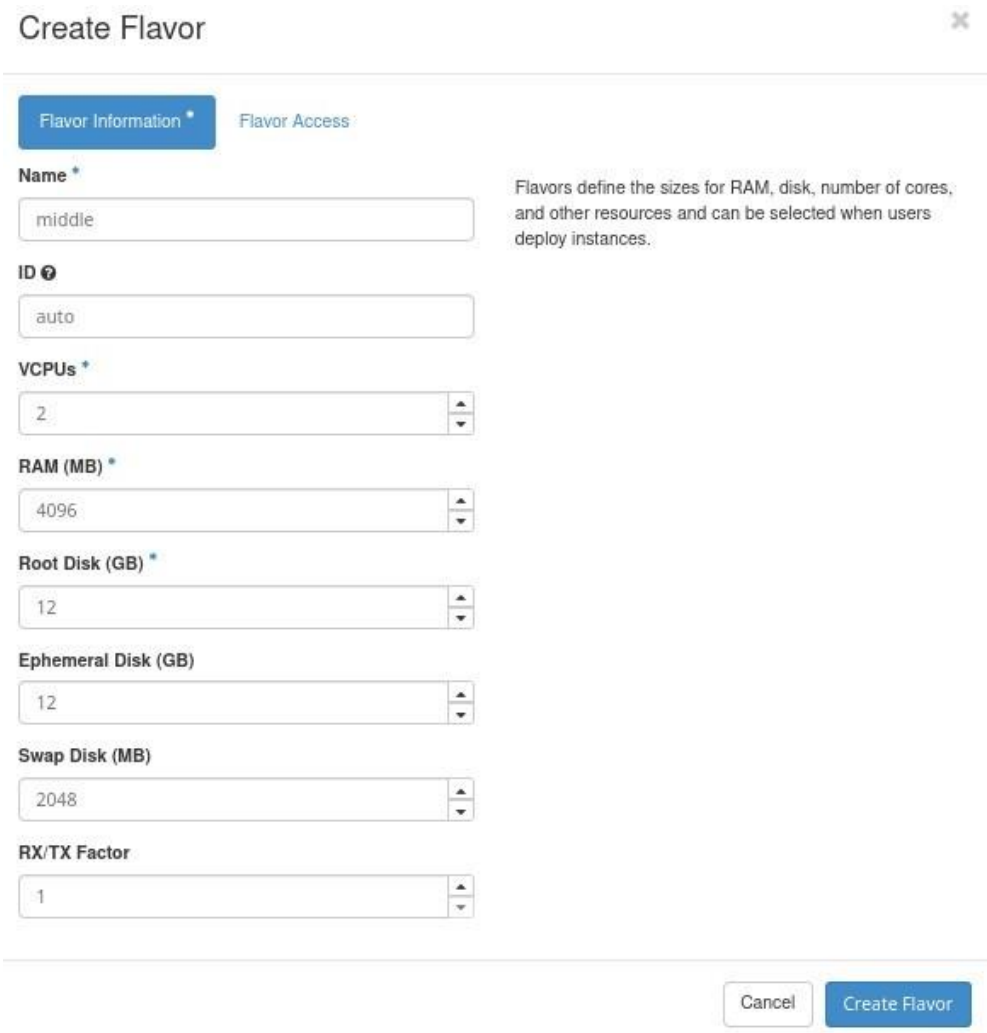

Рис 3.4.2 Створення типу віртуального вузла

Наявна можливість створення різних типів віртуальних вузлів, що дозволить користувачам з простотою та точністю підібрати тип вузла, який задовольнить всі потреби в обчислюваних потужностях та об’ємах сховища. Такий підхід дозволить більш ефективно використовувати ресурси віртуальної лабораторної інфраструктури.

Вікно створення типу віртуального вузла має такі налаштування:

- Name — назва типу вузла, яка відображатиметься користувачу.
- Id — ідентифікатор типу вузла в буквенно-цифровій формі.

- vCpus — кількість віртуальних процесорів, які буде виділено вузлу.
- RAM — кількість оперативної пам’яті, яка буде виділена вузлу.
- Root Disk — об’єм постійного сховища, доступного вузлу. Таке сховище зберігається після видалення віртуального вузла.
- Ephemeral Disk — об’єм постійного сховища, яке видаляється у разі видалення віртуального вузла.
- Swap Disk — об’єм swap розділу.

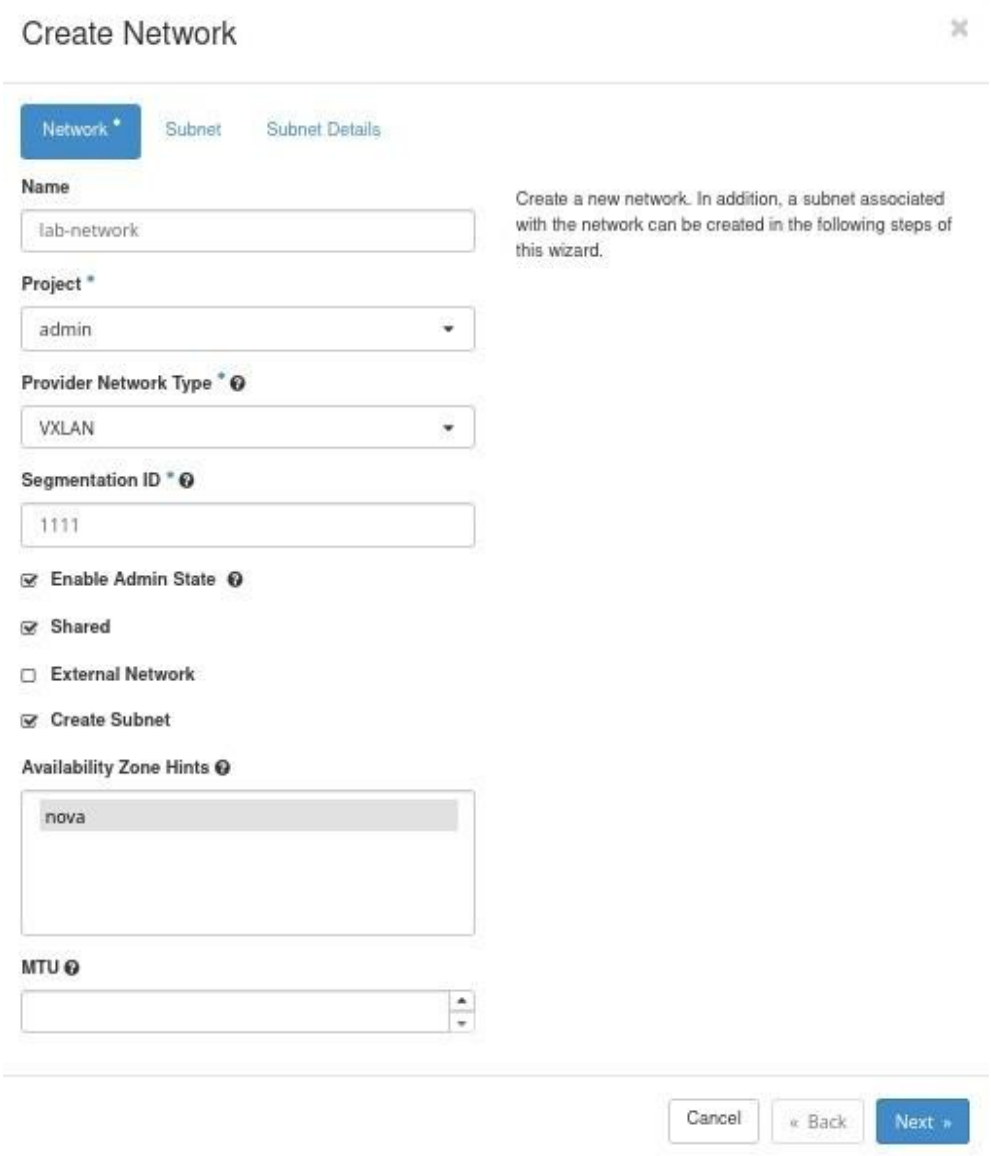

Рис 3.4.3 Створення віртуальної мережі

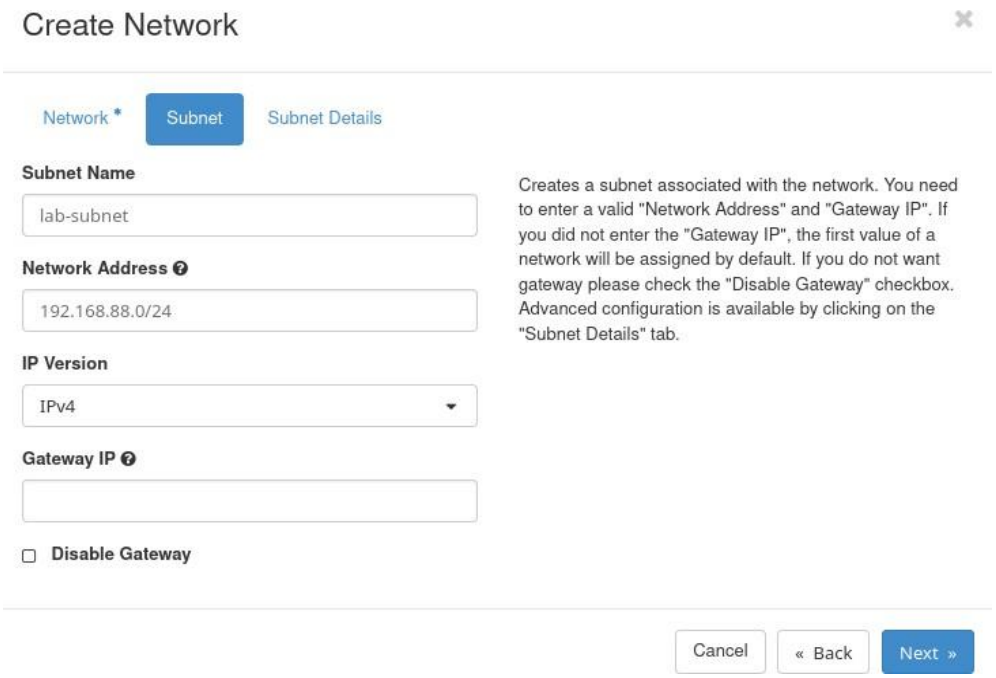

Рис 3.4.4 Створення віртуальної підмережі

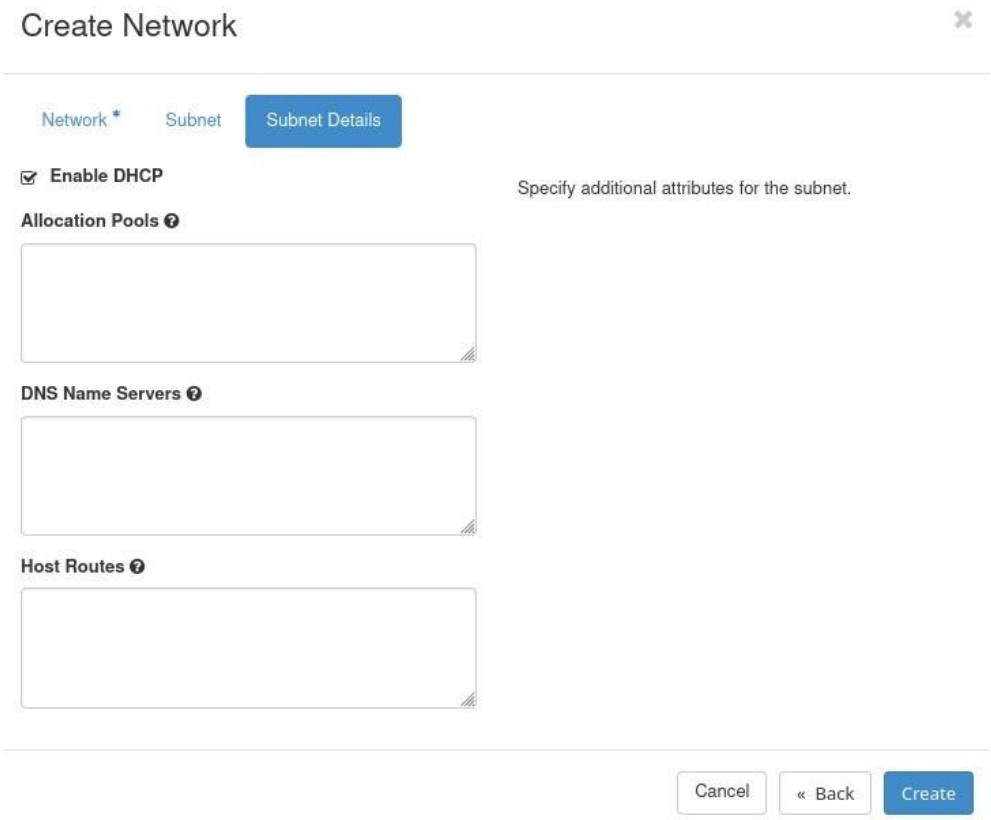

Рис 3.4.5 Налаштування dhcp віртуальної мережі

Також можливе створення віртуальних мереж. Це дозволить створювати ізольовані проекти з декількома віртуальними вузлами. Також віртуальні мережі мають дуже гнучкі налаштування, які включають доступ до віртуального вузла з мережі інтернет, що дозволить збільшити наглядність лабораторних робіт з веб-програмування.

В інтерфейсі налаштування віртуальної мережі наявні такі пункти:

* Network
    - Name — ім’я мережі, яку створюємо.
    - Project — проект, якому буде доступна дана мережа.
    - Provider Network Type — тип віртуальної мережі. Доступні варіанти: local, flat, vlan, gre,vxlan, geneve.
    - Segmentation Id - ідентифікатор, за допомогою якого виконується сегментація мережі на віртуальні частини.
    - Enable Admin State — перемикач, який вмикає та вимикає дану мережу.
    - Shared — дозволяє з’єднувати підмережі з різних проектів.
    - Create Subnet — чи створювати віртуальну підмережу.
    - Availability Zone Hints — вузли, на яких може розміщуватися dhcp сервер для даної віртуальної мережі.
    - MTU — maximum transmission unit для даної віртуальної мережі.
* Subnet
    - Name — ім’я підмережі, яку створюємо.
    - Network Address — адреса підмережі з маскою.
    - Ip Version — вибір між типом мережі Ipv4 та Ipv6.

- Gateway ip — поле для встановлення ip адреси шлюзу за замовчуванням. Якщо поле пусте, використовується перша адреса підмережі.

* Subnet Details

- Enable DHCP — перемикач, який вмикає dhcp сервер для даної підмережі.
- Allocation Pools — список меж, в яких повинні знаходитися адреси, видані dhcp сервером.
- DNS Name Servers — список dns серверів, які надаватимуться з допомогою dhcp.
- Host Routes — список статичних маршрутів. Які надаватимуться за допомогою dhcp.

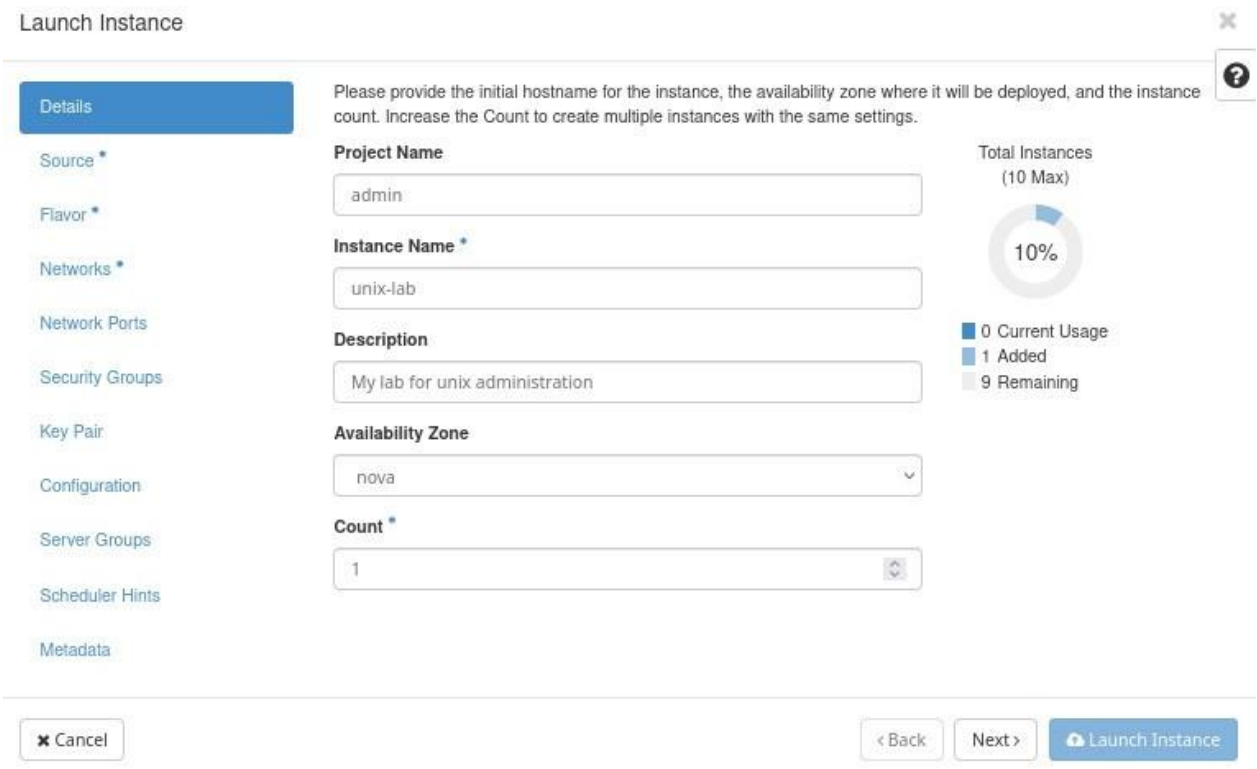

Рис 3.4.6 Створення віртуального вузла, пункт “Деталі”

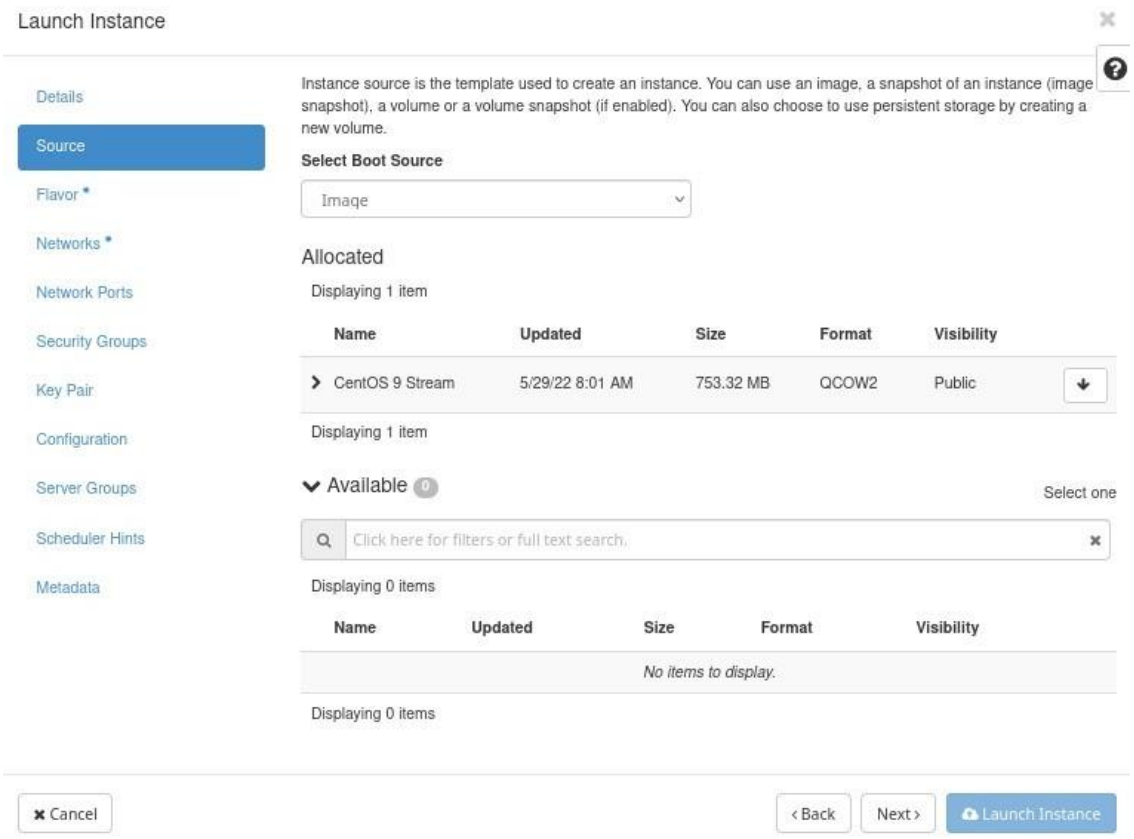

Рис 3.4.7 Створення віртуального вузла, пункт вибору зображення операційної системи

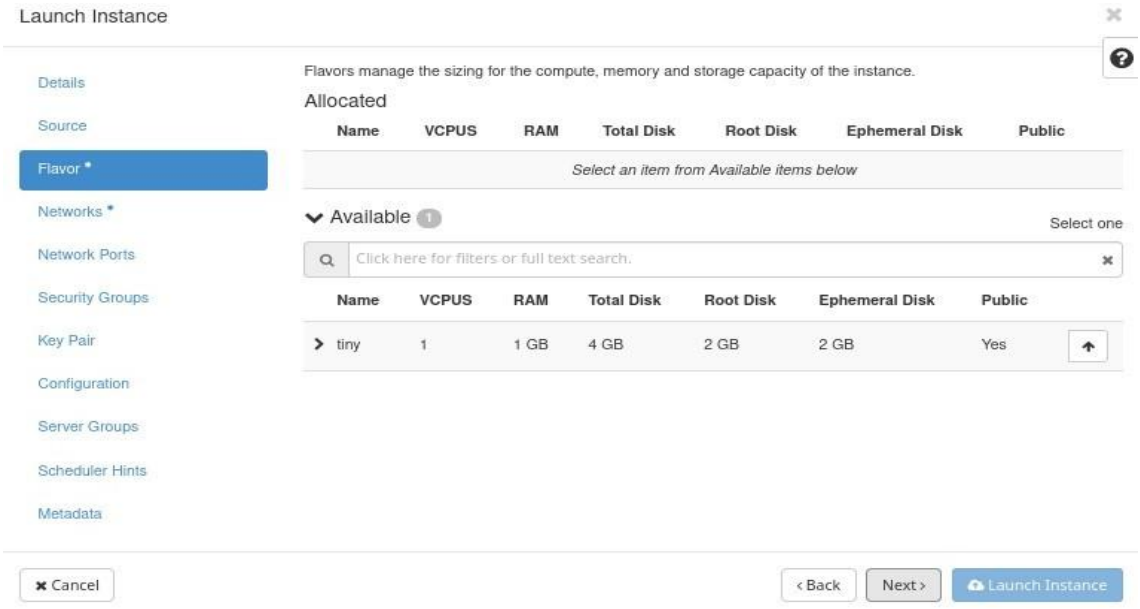

Рис 3.4.8 Створення віртуального вузла, пункт вибору типу вузла

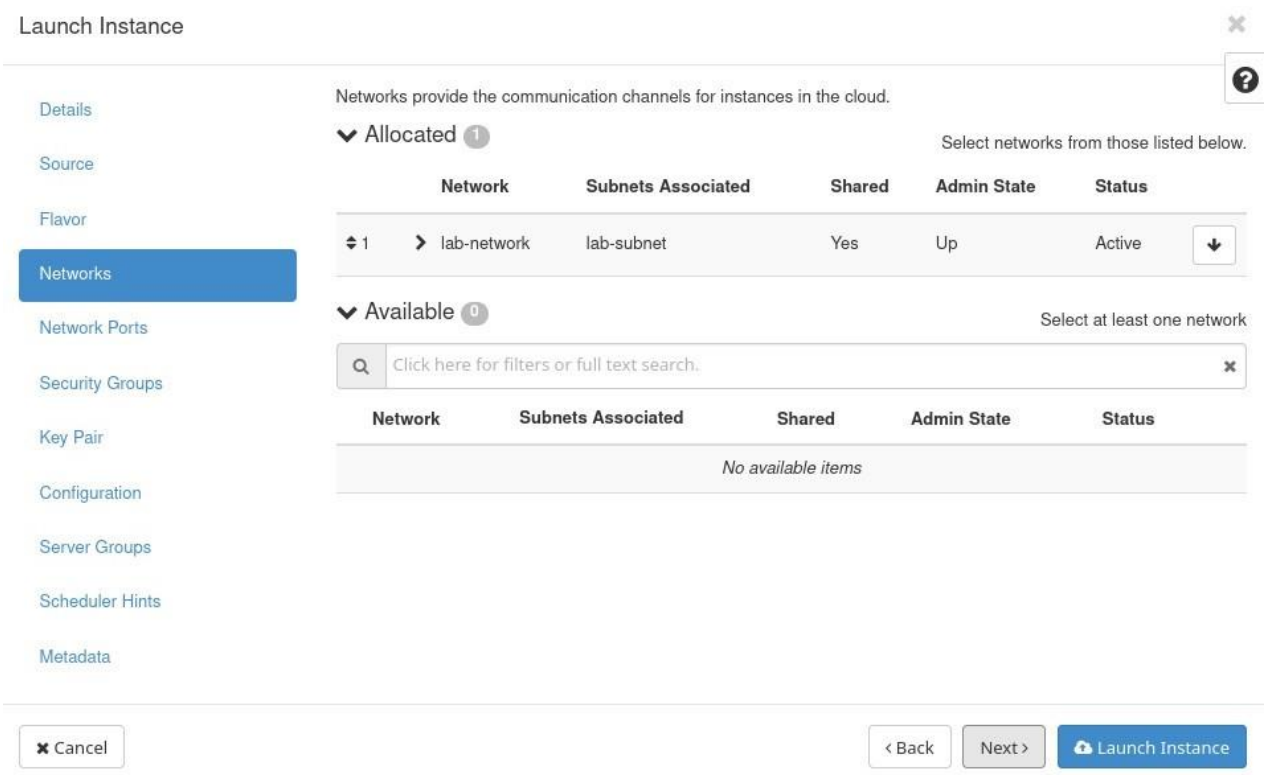

Рис 3.4.9 Створення віртуального вузла, пункт вибору мережі

В інтерфейсі створення віртуального вузла наявні такі пункти:

* Details

- Project Name — ім’я проекту, до якого належатиме віртуальний вузол.
- Instance Name — ім’я створюваного віртуального вузла.
- Description — опис вузла.
- Availability Zone — зона розміщення віртуального вузла, наприклад UA, CN, GE, тощо.
- Count — кількість вузлів, які буде створено.

Source — вибір зображення, яке використовуватиметься для створення вузла.

Flavor — вибір типу віртуального вузла.

Networks — вибір мережі, до якої належатиме вузол.

Network Ports — альтернативний до Networks метод налаштування мереж вузла. (Можливо використання обох одночасно)

Security Groups — безпекова група, до якої належатиме вузол.

Key Pair — вибір ключа ssh, за допомогою якого здійснюватиметься під’єднання до терміналу вузла.

Configuration — пункт, в якому можна обрати тип створення розділів диску та додати конфігураційний сценарій.

Server Groups — вибір груп, до яких належатиме вузол.

Scheduler Hints — налаштування розміщення обчислювальних потужностей вузла на доступних ресурсах.

Metadata — додаткова конфігураційна інформація.

Висновок

Дослідження особливостей виконання лабораторних робіт, студентських проектів та навчальних демонстрацій та дозволило сформувати вимоги сформувати особливості використання та вимоги до віртуальної лабораторної інфраструктури.

Аналіз програмних засобів побудови віртуальної інфраструктури на можливість забезпечення відмовостійкості, наявності системи автентифікації, авторизації та обліку, можливість створення віртуальних вузлів як з ОС Linux так і з Windows, тощо показав доцільність використання програмного комплексу OpenStack.

Запропонована в роботі програмно-апаратна конфігурація на основі OpenStack дозволяє задовольнити вимоги до віртуальної лабораторної інфраструктури для використання в навчальному процесі широкого спектру фахівців з комп'ютерних та мережевих технологій. Тестова експлуатація дозволила підтвердити, що запропонована архітектура дозволяє змінювати кількість фізичних ресурсів без необхідності переконфігурації інфраструктури. Тестова експлуатація дозволила підтвердити, що програмний комплекс OpenStack сконфігурований відповідно до запропонованої архітектури дозволяє використовувати віртуальну лабораторну інфраструктуру для виконання лабораторних робіт, як середовище для тестування програмного забезпечення, як платформа для студентських проектів та навчальних демонстрацій.

Протестувавши розгорнуту віртуальну лабораторну інфраструктуру вдалося підтвердити її відповідність до наведених вимог.

Список використаних джерел

1. OpenStack Documentation [Електронний ресурс] — Режим доступу:
URL: https://docs.openstack.org/yoga/#install-guides
2. OpenStack Compute (nova) [Електронний ресурс] — Режим доступу:
URL: https://docs.openstack.org/nova/latest/
3. Welcome to Swift's documentation [Електронний ресурс] — Режим доступу:
URL: https://docs.openstack.org/swift/latest/
4. Welcome to Glance's documentation [Електронний ресурс] — Режим доступу:
URL: https://docs.openstack.org/glance/latest/
5. Horizon: The OpenStack Dashboard Project [Електронний ресурс] — Режим доступу: URL: https://docs.openstack.org/horizon/latest/
6. Welcome to Neutron's documentation [Електронний ресурс] — Режим доступу: URL: https://docs.openstack.org/neutron/latest/
7. pgpool-II 4.0.9 Documentation [Електронний ресурс] — Режим доступу:
URL: https://www.pgpool.net/docs/pgpool-II-4.0.9/en/html/index.html
8. Quorum Queues [Електронний ресурс] — Режим доступу:
URL: https://www.rabbitmq.com/quorum-queues.html
9. v3.5 docs [Електронний ресурс] — Режим доступу:
URL: https://etcd.io/docs/v3.5/
10. Kolla Overview [Електронний ресурс] — Режим доступу:
URL: https://github.com/openstack/kolla
11. Kolla Ansible [Електронний ресурс] — Режим доступу:
URL: https://github.com/openstack/kolla-ansible