**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**КАФЕДРА** **КОМП’ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ**

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.В. Казмірчук

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

На правах рукопису

УДК 004.056.5:510.22(043.3)

**МАГІСТЕРСЬКА АТЕСТАЦІЙНА РОБОТА**

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ**

**«МАГІСТР»**

**Тема**:Відмовостійка кластерна система на основі оптимального резервування

|  |  |
| --- | --- |
| **Автор:** | М.В. Копитовський |
| **Науковий керівник:** к.т.н., доц. | В.Г. Павлов |
| **Нормоконтролер:** асист. | С.В. Егоров |

**Київ 2020**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**Факультет:** Кібербезпеки, комп’ютерної та програмної інженерії

**Кафедра:** Компютеризованих систем захисту інформації

**Освітній ступінь:** Магістр

**Спеціальність:** 125 «Кібербезпека»

**Освітньо-професійна програма**: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_ С.В. Казмірчук

«\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_ р.

**ЗАВДАННЯ**

**на виконання магістерської атестаційної роботи**

**магістранта Копитовського Максима Валерійовича**

1. Тема: *Відмовостійка кластерна система на основі оптимального резервування*

затверджена наказом ректора від «02» жовтня 2019 № 2265/ст*.*

1. Термін виконання з 14.10.2019р. по 09.02.2020р*.*
2. Вихідні дані: дані про інформаціно-комунікаційну систему для аналізу, визначення рівня захищеності системи та можливих вдосконалень.
3. Зміст пояснювальної записки: аналітичний огляд літературних джерел з тематики диплому. Аналіз компонентів інформаційно-комунікаційної системи та визначення її захищенності та можливих удосконалень. Знаходження та реалізація найбільш оптимального рішення для відмовостійкості.

**КАЛЕНДАРНИЙ ПЛАН**

**виконання магістерської роботи**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Етапи виконання магістерської роботи** | **Термін виконання етапів** | **Примітка** |
|  | Уточнення постановки задачі | 03.10.2019 | *Виконано* |
|  | Аналіз літературних джерел | 10.10.2019 | *Виконано* |
|  | Обґрунтування вибору рішення | 20.10.2019 | *Виконано* |
|  | Збір інформації | 21.10.2019 | *Виконано* |
|  | Аналіз даних та їх класифікація | 15.11.2019 | *Виконано* |
|  | Ознайомлення із документацією протоколів, операційних систем та серверу додатків | 25.11.2019 | *Виконано* |
|  | Побудова різних рішень серверних архітектур та їх аналіз | 03.12.2019 | *Виконано* |
|  | Оцінка ефективності побудованої системи | 29.12.2019 | *Виконано* |
|  | Перевірка на антиплагіат |  |  |
|  | Оформлення і друк пояснювальної записки |  |  |
|  | Оформлення презентації |  |  |
|  | Отримання рецензій від рецензента |  |  |
|  | Захист в ЕК |  |  |

Магістрант М. Копитовський

(підпис, дата)

Науковий керівник В. Павлов

(підпис, дата)

**РЕФЕРАТ**

Магістерська атестаційна робота складається зі вступу, двох розділів, загальних висновків, списку використаних джерел, додатків і має 75 сторінки основного тексту, 36 рисунка, 4 таблиць. Список використаних джерел містить 25 найменування і займає 4 сторінок. Загальний обсяг роботи 88 сторінок.

Метою роботи є є підвищення доступності і цілісності інформації за рахунок створення відмовостійкого кластеру на базі UNIX-серверів, на базі Oracle Glassfish Open Source Server 4.0, порівняння поведінки кожної з операційних систем, реалізація відмовостійкого кластеру з балансувальником навантаження та резервним копіюванням до хмарного сховища.

Методи дослідження: моделювання, удосконелення та аналіз створенної інформаційно-комунікаційної мережі.

Технічні та програмні засоби: Ubuntu, Oracle Glassfish, NGINX, Dropbox, AWS

Отримані результати та їх новизна: захищена система на базі Oracle Glassfish Open Server.

В роботі був вдосконален відмовостійкий кластер на базі Oracle Glassfish Open Server для підвищення захисту від загроз та інших факторів, що впливають на працездатність усієї системи.

Розроблений метод та програмне забезпечення відносяться до галузі інформаційної безпеки і можуть бути використані для підвищення рівня захищеності.

Рекомендації щодо використання результатів: дану систему можна використовувати у більшості великих компаній, завдяки захищеності та аналізу найкращої продуктивності та швидкодії системи.

Галузь застосування та ступінь впроваження матеріалів дипломної роботи: будь-яке підприємство, де є необхідність в неперервній роботі системи

Ключові слова: Відмовостійкий кластер, балансувальник навантаження, резервне копіювання, хмарне сховище, інформаційна безпека, інформаційно-комунікаційна система.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 6](#_Toc30352813)

[ВСТУП 7](#_Toc30352814)

Розділ 1. [ОПИС ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ 9](#_Toc30352815)

1.1. [Аналіз останніх досліджень і публікацій 9](#_Toc30352816)

[1.1.1 Відокремлення невирішених раніше частин загальної проблеми. 9](#_Toc30352817)

[1.2. Відмовостійкість 10](#_Toc30352818)

[1.2.1 Функціонування відмовостійкості 11](#_Toc30352819)

[1.2.2 Необхідні компоненти для високої доступності 13](#_Toc30352820)

[1.2.3 Причини простою 14](#_Toc30352821)

[1.2.4 Конфігурації відмовостійкого кластеру 21](#_Toc30352822)

[1.3. Оптимальне резервування 23](#_Toc30352823)

[1.3.1 Завдання оптимального резервування комп’ютерних систем 23](#_Toc30352824)

[1.3.2 Оптимальний розподіл резервів чисельними методами 25](#_Toc30352825)

[1.4. Балансування навантаження 27](#_Toc30352826)

[1.4.1 Необхідність балансувальника навантаження 27](#_Toc30352827)

[1.4.2 Види балансувальників навантаження 28](#_Toc30352828)

[1.5. Кластеризація серверів 29](#_Toc30352829)

[1.5.1 Необхідність застосування кластеризації серверів 29](#_Toc30352830)

[1.5.2 Види кластеризації 30](#_Toc30352831)

[1.6. Висновки до розділу 31](#_Toc30352832)

Розділ 2. [РЕАЛІЗАЦІЯ ТА НАЛАШТУВАННЯ ВІДМОВОСТІЙКОГО КЛАСТЕРУ 32](#_Toc30352833)

[2.1. Конфігурація обладнання 32](#_Toc30352834)

[2.2. Вибір операційної системи 33](#_Toc30352835)

[2.2.1 Ключові відмінності Windows та Linux 34](#_Toc30352836)

[2.2.2 Тестування GF серверу на кожній з ОС 37](#_Toc30352837)

[2.2.3 Висновок по вибору ОС 39](#_Toc30352838)

[2.3. Реалізація та оптимізація серверної архітектури 39](#_Toc30352839)

[2.3.1 Односерверна архітетура 40](#_Toc30352840)

[2.3.2 Багатосерверна архітектура 41](#_Toc30352841)

[2.3.3 Кластерна архітектура 42](#_Toc30352842)

[2.3.4 Кластерна архітектура з використанням БН 55](#_Toc30352843)

[2.3.5 Удосконалення кластерної архітектури на базі Glassfish 68](#_Toc30352844)

[2.4. Висновки до розділу 80](#_Toc30352845)

[ВИСНОВКИ 81](#_Toc30352846)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 85](#_Toc30352847)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

SSH (англ. Secure SHell — «безпечна оболонка») — мережевий протокол рівня застосунків;

SSL (англ. Secure Sockets Layer — рівень захищених сокетів) — криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером;

GF (англ. Glassfish) — сервер додатків з відкритим вихідним кодом;

Відмовостійкість (Fault Tolerance, FT) — здатність системи до подальшої роботи після виходу з ладу будь-якого її елемента;

Кластер — група серверів (обчислювальних одиниць), об'єднаних каналами зв'язку;

Відмовостійкий кластер (Fault Tolerant Cluster, FTC) — кластер, відмова сервера в якому не призводить до повної непрацездатності всього кластера. Завдання вийшла з ладу машини розподіляються між однією або кількома залишилися нодамі в автоматичному режимі;

Безперервна доступність (CA) - користувач може використовувати сервіс в будь-який час, без перерв у наданні. Скільки часу пройшло з моменту відмови вузла, не має значення;

Висока доступність (High Availability, HA) — в разі виходу з ладу вузла користувач якийсь час не отримуватиме послугу, проте відновлення системи відбудеться автоматично; час простою мінімізується;

КБД — кластер безперервної доступності, CA-кластер;

КВД — кластер високої доступності, HA-кластер;

# ВСТУП

Інтернет дозволяє підприємствам будь-якого розміру і в будь-якому місці виходити на нові і більш великі ринки і дає можливість працювати більш ефективно за допомогою комп'ютерних інструментів. Незалежно від того, чи планує компанія прийняти хмарні обчислення або просто використовувати електронну пошту та підтримувати веб-сайт, інформаційна безпека повинна бути частиною плану. Крадіжка цифрової інформації стала найпоширенішим шахрайством, випередив фізичне злодійство. Кожен бізнес, який використовує Інтернет, несе відповідальність за створення культури безпеки, яка підвищить довіру бізнесу і споживачів.

Простої призводять до втрати близько 900 000 доларів за сім днів для компанії, яка має більше десяти тисяч клієнтів. Згідно зі статистикою, більше половини компаній мають як мінімум півтори години простою щотижня. Прямі втрати значні, але вони також представляють високий ризик втрати довіри клієнтів. Це набагато складніше оцінити, але, тим не менш, це проблема утримання клієнтів і істотний бар'єр для зростання. Коротше кажучи, будь-яка форма незапланованих простоїв є токсичною для успіху бізнесу, незалежно від ролі, яку відіграє бізнес-додаток.

Найкращий спосіб запобігти простоям та усунути ці втрати - це прийняти ряд найкращих практик, які допоможуть вам досягти високої доступності для вашої послуги чи програми. Методології з відмовостійкістью, або HA, мають на меті підтримувати безперебійний сервіс якомога довше - як правило, допускаючи простої близькі до нуля.

Метою даної роботи є підвищення доступності і цілісності інформації за рахунок створення відмовостійкого кластеру. Для цього вирішуються наступні завдання:

* Проаналізувати стан даного питання за підсумками останніх досмліджень та публікацій;
* Виявити та класіфікувати основні чинники, що впливають на відмовостійкість та доступність кластерних систем;
* Розглянути стуктуру та основні складові кластерних систем;
* Проаналізувати балансування навантаження в умовах резервування;
* Побудувати та протестувати відмовостійкий кластер.

Саме ці задачі є головними для реалізації найбільш безпечної та ефективної системи, яка не підведе та зможе «витримати» низку атак у вигляді навантаження. Оскільки у одному додатку Oracle Glassfish Open Source Server 4.0, на якому ми будемо розгортати інфраструктуру, не має можливості усе це реалізувати, то ми будемо по ходу даної роботи вирішувати та аналізувати найефективніші способи та методи реалізації.

Об´єктом дослідження є процес побудови захищеної відмовостійкої кластерної системи зі застосуванням сучасних технологій.

Предмет дослідження – відмовостійка кластерна серверна система, яка забезпечує розподіл ресурсів та за рахунок цього має підвищену надійність зберігання інформації та захищеність, у тому числі предметом є балансувальник навантаження та хмарне резервування.

Наукова новизна одержаних результатів: удосконалена відмовостійка кластерна система на базі Oracle Glassfish Open Server із використанням балансувальником навантаження на базі NGINX та FTP сервер для спільного використання із хмарним резервним копіюванням.

Галузь застосування і апробація результатів роботи: середні та великі підприємства будь-якої галузі, де є необхідність в безперервній роботі системи.

**РОЗДІЛ 1**

# ОПИС ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

## Аналіз останніх досліджень і публікацій

Відомо, що головними завданнями інформаційної безпеки є забезпечення конфіденційності, [доступності](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0) і [цілісності](https://uk.wikipedia.org/wiki/%D0%A6%D1%96%D0%BB%D1%96%D1%81%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97) інформації, а також захищеності [інформації](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%8F) від несанкціонованого доступу [1][2]. Зазначені вище атаки впливають насамперед на доступність інформаційних ресурсів, тобто на можливість отримати необхідну інформацію у будь-який час з максимальною швидкістю [3]. На сьогоднішній день доводиться констатувати, що надійного комплексного засобу протидії цим атакам немає, бо найчастіше вони мають комбінований характер [4].

Основуючись на статті [5], було виділено три основні типа атак:

* атаки на рівні мережних каналів;
* атаки на рівні мережних додатків;
* атаки на рівні мережних сервісів.

Останні види атак пов´язані з WEB, PROXY, SSL, DNS та іншими серверами, до яких можливій доступ ззовні мережі [6]. Тому зазвичай надаються рекомендації щодо контролю над запитами, що здійснюють користувачі, фільтрації трафіку тощо [7]. Але жодне з існуючих на цей момент обладнання не забезпечує 100% захист [8].

У той же час з´являються численні пропозиції, що надходять від таких потужних компаній, як Google, Xelent, Yandex та ін. щодо використання їхніх хмарних ресурсів для розміщення та резервування інформації. Але знову ж таки ці storage-рішення не вирішують проблему. Наприклад атака, яка була здійснена у сімнадцятому році на хмарну інфраструктуру одну із дуже великих компаній, за даними звіту, нанесла збитків більш ніж на два мільйона доларів США [9].

### 1.1.1 Відокремлення невирішених раніше частин загальної проблеми.

В якості одного з можливих рішень пропонується підвищити рівень безпеки у інформаційно-комунікаційної мережі шляхом побудови відмовостійкого кластеру з балансувальником навантаження та резервного копіювання. Потрібний ефект досягається за рахунок автоматичного перезапуску додатку на будь-якому іншому вузлі кластеру, якщо певний вузол внаслідок апаратного або програмного збою або атаки стає непрацездатним. У свою ж чергу, за рахунок балансувальника навантаження, трафік буде поступати лише на незагружені вузли кластеру. В додаток до всього цього буде реалізоване резервне копіювання до хмарного сховища, де інформація буде доступна вповноваженим людям навіть у разі виходу усіх вузлів зі строю. Таким чином, буде досягнута найбільш безпечна та ефективна інфраструктура серверної частини за мінімальні кошти.

## 1.2. Відмовостійкість

Обчислювальні середовища, налаштовані на забезпечення майже повноцінної доступності, відомі як системи високої доступності. У таких системах зазвичай є надлишкові апаратні та програмні засоби, що робить систему доступною, незважаючи на збої. Добре розроблені системи з високою доступністю не дозволяють мати окремі точки відмови. Будь-який апаратний або програмний компонент, який може вийти з ладу, має надлишковий компонент одного типу.

Коли трапляються збої, процес відмови переміщує обробку, виконану невдалим компонентом, на резервний компонент. Цей процес ремастеризує загальносистемні ресурси, відновлює часткові або невдалі транзакції та відновлює систему в нормі. Чим прозоріший цей відмов для користувачів, тим вище доступність системи [10].

Ви можете класифікувати системи та оцінювати їх очікувану доступність за типом системи. Найважливіші для місії та найважливіші для бізнесу програми, такі як сервери електронної пошти та Інтернет, ймовірно, потребують значно більшої доступності, ніж програми, які мають меншу кількість користувачів. Крім того, деякі системи можуть мати постійну потребу в режимі безперервної роботи, а інші, такі як система відстеження фондового ринку, матимуть майже постійні вимоги до безперервного часу для конкретних часових рамків, наприклад, коли фондовий ринок є відчинено.

### 1.2.1 Функціонування відмовостійкості

Відмовостійкість функціонує як механізм реагування на збій інфраструктури. Це працює досить просто в концептуальному плані, але зазвичай вимагає спеціального програмного забезпечення і конфігурації.

Мінімізація простоїв часто є пріоритетом при налаштуванні надійних виробничих систем. Незалежно від того, наскільки надійні ваші системи і програмне забезпечення, можуть виникнути проблеми, які можуть привести до збою ваших програм або серверів.

Реалізація високої доступності для вашої інфраструктури є корисною стратегією для зниження впливу подій такого типу. Системи високої доступності можуть автоматично відновлюватися після збою сервера або компонента.

Одна з цілей високої доступності - усунути окремі точки відмови у вашій інфраструктурі. Одиночна помилка - це компонент вашого технологічного стека, який може перервати обслуговування, якщо він недоступний. Отже, будь-який компонент, необхідний для правильної роботи вашої програми без надмірності, вважається єдиною точкою відмови.

Щоб усунути окремі точки відмови, кожен рівень вашого стека повинен бути підготовлений до надмірності. Наприклад, уявіть, що у вас є інфраструктура з двох ідентичних надлишкових веб-серверів на балансувальник навантаження. Клієнтський трафік буде рівномірно розподілятися між веб-серверами, але якщо один сервер вийде з ладу, балансувальник навантаження перенаправить весь трафік на іншу частину веб-сервера [11].

Шар веб-сервера в цьому сценарії не є єдиною точкою відмови, оскільки:

* є надлишки компонентів для одного завдання
* механізм зверху цього шару (балансир навантаження) здатний виявляти збої в компонентах і адаптувати його поведінку для своєчасного відновлення

Але що станеться, якщо балансир навантаження перейде в офлайн-режим?

При описаному сценарії, який не є рідкістю в реальному житті, сам рівень балансування навантаження залишається єдиною точкою відмови. Однак усунення цієї єдиної точки відмови може бути складним завданням; Хоча ви можете легко налаштувати додатковий балансувальник навантаження для досягнення надмірності, над балансувальник навантаження немає очевидної точки для виявлення і усунення несправностей.

Одне тільки перевищення не може гарантувати високу доступність. У вас повинен бути механізм для усунення неполадок і вжиття заходів, коли один з компонентів вашого стека стане недоступним.

Пошук та усунення несправностей і відновлення для надлишкових систем може бути виконано з використанням спадного підходу: рівень верхнього рівня стає відповідальним за моніторинг рівня безпосередньо під ним на предмет збоїв. У нашому попередньому прикладі сценарій балансування навантаження є верхнім шаром. Якщо один з веб-серверів стає недоступним, балансувальник завантаження перестане перенаправляти запити на певний сервер.

Цей підхід зазвичай простіше, але у нього є обмеження: у вашій інфраструктурі буде точка, в якій верхній рівень або не існує, або недоступний для рівня балансування навантаження. Створення служби відпрацювання відмови для балансування навантаження на зовнішньому сервері просто створить нову єдину точку відмови.

У цьому сценарії потрібно розподілений підхід. Кілька надлишкових вузлів повинні бути пов'язані один з одним як кластер, де кожен вузол повинен бути в рівній мірі здатний виявляти і усувати несправності.

Проте, в разі балансувальник навантаження, існує додаткова складність, пов'язана з роботою серверів імен. Відновлення з балансувальника навантаження зазвичай означає відмову від резервного балансувальника навантаження, що означає, що вам потрібно змінити DNS, щоб вказати доменне ім'я для IP-адреси балансувальника навантаження. Такі зміни можуть зайняти значний час для поширення в Інтернеті, що може привести до серйозного простою системи.

Можливе рішення - використовувати круглу балансування DNS для балансування навантаження. Однак такий підхід ненадійний, оскільки залишає невідповідну клієнтську програму.

Більш надійним і надійним рішенням є використання систем, які дозволяють гнучко переосмислювати IP-адреси, наприклад плаваючі IP-адреси. Перезавантаження IP-адреси на вимогу усуває проблеми з розповсюдженням і кешуванням DNS, надаючи статичний IP-адресу, який при необхідності можна легко перепакувати. Доменне ім'я може залишатися пов'язаним з тим же IP-адресою, в той час як сам IP-адреса переміщається між серверами.

### 1.2.2 Необхідні компоненти для високої доступності

Є кілька компонентів, які необхідно ретельно враховувати для впровадження високої доступності на практиці [12]. Набагато більше, ніж впровадження програмного забезпечення, висока доступність залежить від таких факторів, як:

* Навколишнє середовище: якщо всі ваші сервери розташовані в одній географічній зоні, умови навколишнього середовища, такі як землетрус або повінь, можуть зруйнувати всю вашу систему. Наявність надлишкових серверів в різних дата-центрах і географічних зонах підвищить надійність.
* Апаратне забезпечення. Сервери високої доступності повинні бути стійкі до перебоїв харчування і апаратних збоїв, включаючи жорсткі диски і мережеві інтерфейси.
* Програмне забезпечення: весь програмний стек, включаючи операційну систему і саму програму, повинен бути підготовлений до обробки непередбачених помилок, які можуть, наприклад, потенційно вимагати перезавантаження системи.
* Дані: втрата даних і невідповідність можуть бути викликані декількома факторами і не обмежуються збоями жорсткого диска. Системи високої доступності повинні враховувати безпеку даних в разі збою.
* Мережа: незаплановані відключення мережі є ще однією можливою точкою відмови для систем високої доступності. Важливо, щоб надлишкова мережева стратегія була створена для можливих збоїв.

Індустрія програмного забезпечення зазвичай вимірює доступність за допомогою двох типів показників (вимірювань):

* Середній час для відновлення
* Середній час між відмовами

У більшості сценаріїв збоїв галузь фокусується на середньому часу відновлення і вивчає способи оптимального проектування систем для їх зменшення. Як правило, це більше відноситься до доступності обладнання; ця глава НЕ деталізує середній час між відмовами. Однак, оскільки ви можете створювати реальні кластери додатків, щоб уникнути одиничних точок збою, збої компонентів не обов'язково роблять додаток недоступним. Отже, реальні кластери додатків можуть бути значно скорочені з точки зору доступності додатків.

Інший часто використовуваний показник – це, наприклад, вісім годин недоступності системи щороку призводять до дев'яносто дев'яти відсотків або трьом дев'яткам. Кілька хвилин недоступності системи щороку дають п'ять дев'яток.

Важко розглянути кілька десятків варіантів доступності, які не описавши при цьому основних правил і строгих процесів управління додатками, методологій тестування і процедур управління змінами. З цієї причини реальні кластери додатків можуть значно скоротити середній час до відмови при збоях. Це неминуче сприяє кращій доступності всієї системи.

Як уже згадувалося, добре розроблена система для відмов кластерів містить надлишкові компоненти, які захищають від більшості збоїв і створюють середовище без єдиного збою. Робота з вашим постачальником обладнання є ключем до створення повністю надлишкових кластерних середовищ для реальних групових застосунків..

### 1.2.3 Причини простою

Час простою може бути плановим або позаплановим. На рисунку 1.1 показано багато причин непередбачуваного простою. Вони можуть бути класифіковані як системні помилки, помилки даних та медіа-файлів та збої на сайті. Цей незапланований час простою є руйнівним, оскільки важко передбачити його терміни.

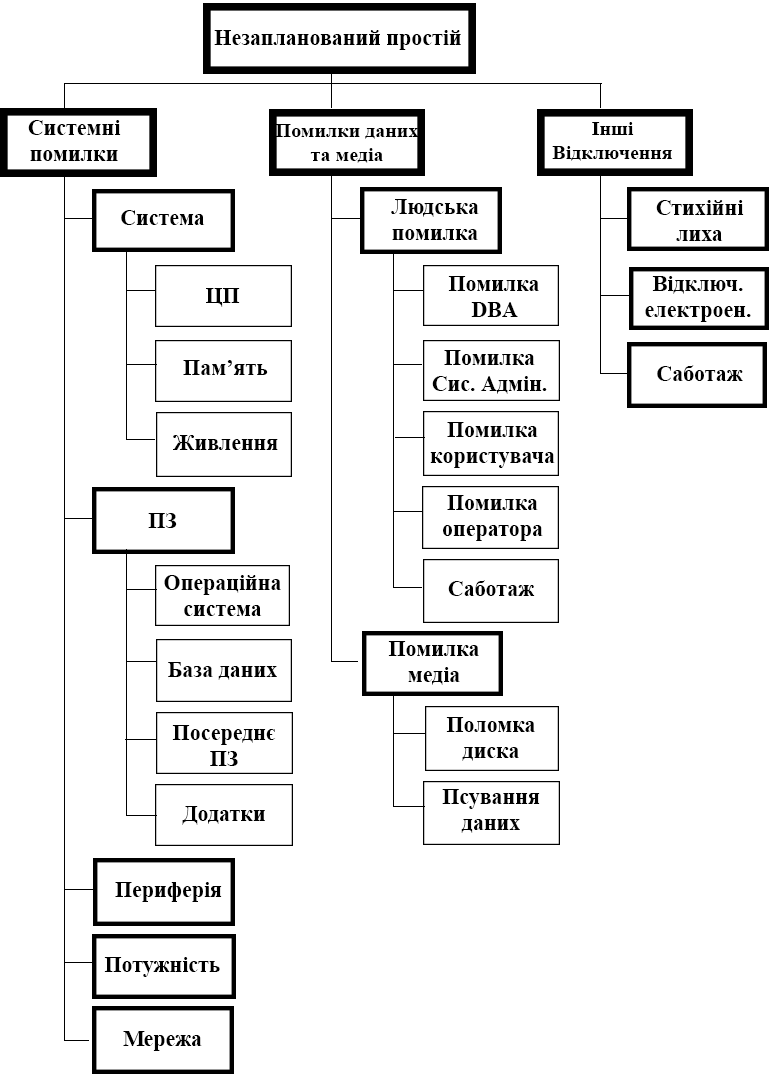


Рис. 1.1 Причини незапланового простою

На рис. 1.1 видно, що незаплановані простої діляться на 3 категорії: Системні помилки, помилка даних і медіа та інші відключення.

Із системних помилок можна виділити основні підкатегорії такі як: система та програмне забезпечення. До системи відносяться усі апаратні помилки, у томі числі помилки центрального процесора, пам’яті та живлення. У помилках програмного забезпечення можна виділити основні підкатегорії такі як: операційна система, база даних та додатки. Саме вони мають найбільш вразливі місця.

Помилки даних та медіа діляться на 2 типи: людська помилка та помилка медіа. До людських помилок відносять помилки користувача, помилки системних адміністраторів, тощо. Із помилок медіа можна виділити основну проблему – псування даних.

До інших помилок відносяться будь-які стихійні лиха, відключення електроенергії та інші моменти, які не зовсім контрольовані людиною чи системою.

Однак запланований час простою може бути настільки ж руйнівним для операцій, особливо на глобальних підприємствах, які підтримують користувачів в кількох часових поясах. На малюнку 1.2 показано кілька типів запланованих простоїв. Зазвичай їх можна класифікувати як рутинні операції, періодичне обслуговування і модернізацію чи оновлення.

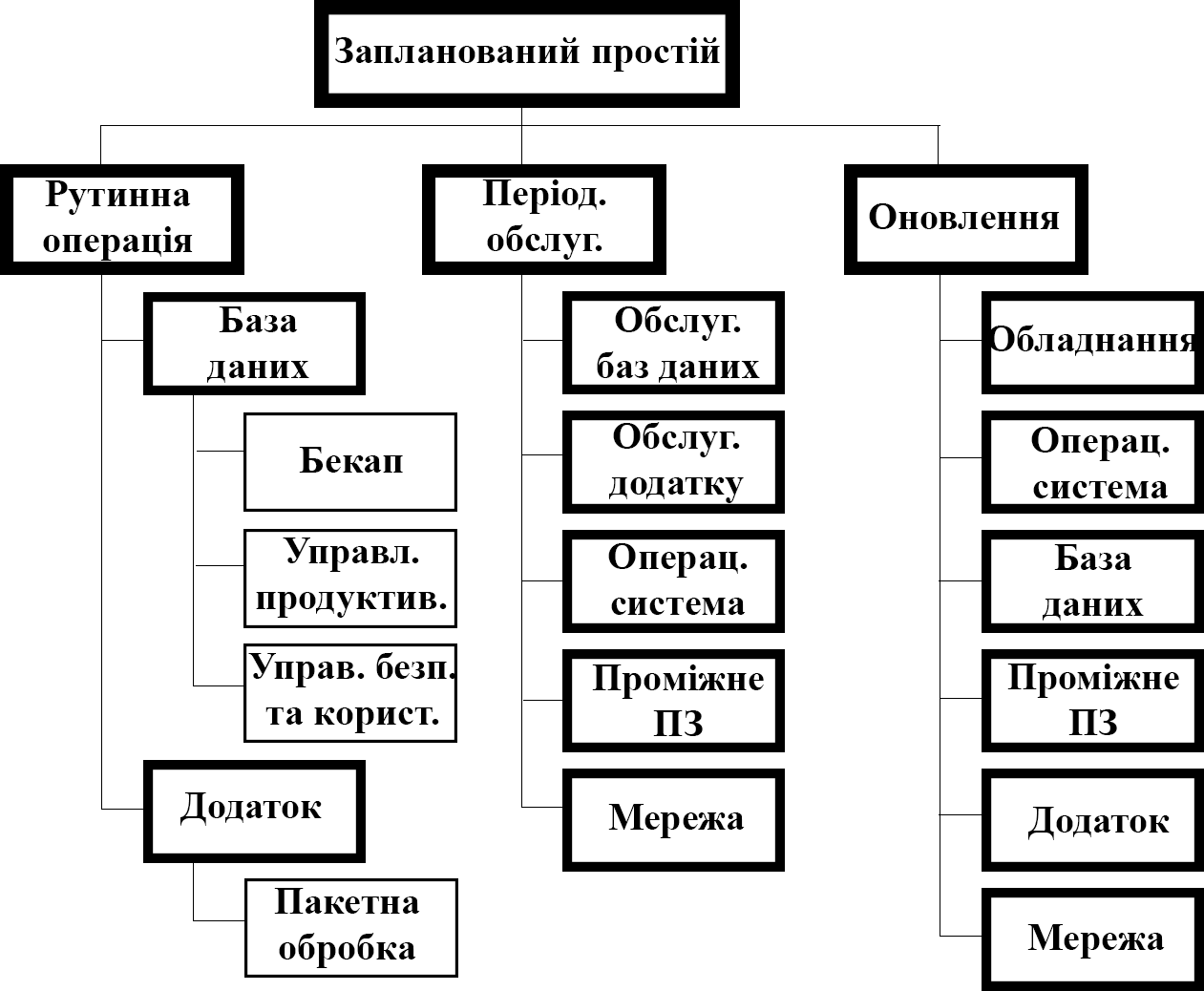


Рис. 1.2 Причини запланованого простою

На рис. 1.2 ми бачимо класифікацію типів запланованого простою. Запланований простій можна поділити на 3 типи: рутинна операція, періодичне обслуговування та оновлення.

До рутинних операцій відносять базу даних та додатки. Що стосується підкатегорії бази даних, то основною причиною простою є бекап чи резервне копіювання та управління безпекою та користувачами. Саме у цей час частина системи може переходити у простій.

Із періодичного обслуговування можна виділити такі підкатегорії: обслуговування баз даних, обслуговування додатку та налаштування мережі.

Одна із основних причин простою є оновлення. Оновлення діляться на такі типи як: оновлення обладнання, оновлення операційної система, оновлення бази даних, тощо.

Ключовим фактором побудови рішення з високою доступністю є врахування всіх цих причин простоїв та розробка рішення, яке їх вирішує.

Справжні кластери додатків створюють більш високий рівень доступності на додаток до стандартних функцій Oracle. Відмовостійкі функції єдиного примірника, такі як швидке відновлення і онлайн-реорганізація, також застосовні до реальних кластерним додатків. Швидке відновлення роботи може значно скоротити середній час відмов з мінімальним впливом на продуктивність онлайн-додатків. Інтернет-реорганізації скорочують тривалість запланованих простоїв. Багато операцій можуть виконуватися в Інтернеті, поки користувачі оновлюють основні об'єкти. Реальні кластери додатків зберігають всі ці стандартні функції Oracle.

На додаток до всіх звичайних функцій відмовостійкий кластер використовує надмірність, що забезпечується кластеризацією, для забезпечення доступності при збоях n-мінус 1 в кластері з n вузлів. Іншими словами, всі користувачі мають доступ до всіх даних, якщо в кластері є один доступний вузол.

Щоб налаштувати реальні кластери додатків для забезпечення високої доступності, необхідно уважно розглянути апаратні і програмні компоненти кластера, як описано в наступному розділі.

Реальна кластерна прикладна середу повністю надлишкова, оскільки всі вузли мають доступ до всіх дисків кластера. Відмова одного вузла не впливає на здатність іншого вузла обробляти транзакції. До тих пір, поки в кластері є один виживає вузол, всі клієнти бази даних можуть обробляти всі транзакції, хоча це може привести до збільшення часу реакції через обмеження місткості на одному вузлі.

Кластерні системи часто не помічають надмірності відносин. Це тому, що середній час до відмови зазвичай становить кілька років. Отже, надлишкові кластерні відносини можуть не бути головним пріоритетом. Крім того, в залежності від рівня системи і рівня поліпшення надмірне з'єднання кластерів може бути нестерпним і не мати достатнього бізнес-обгрунтування.

Однак надмірне підключення кластера є важливим аспектом повністю надлишкового кластера. Без цього система дійсно не обходиться без єдиної точки відмови. Відносини кластера можуть не працювати з різних причин, і не всі з них беруться до уваги. Це також не може бути розглянуто, якщо метрики виробника надані. З'єднання може вийти з ладу через несправність пристрою, наприклад, через несправність генератора при підключенні комутатора або через людську помилку.

Реальні кластерні програми працюють на одному образі даних; всі вузли кластера мають доступ до одного і того ж набору файлів даних. Адміністраторам баз даних рекомендується використовувати апаратне віддзеркалення для підтримки надлишкових носіїв. В цьому відношенні реальні прикладні кластери нічим не відрізняються від разових. Надлишок диска залежить від базового апаратного і програмного віддзеркалювання даних, таких як надмірний масив незалежних дисків.

Реальні кластерні прикладні середовища мають повну надмірність вузлів; кожен вузол виконує свою власну копію операційної системи. Отже, ті ж міркування для надмірності вузлів застосовні до операційної системи. Cluster Manager - це розширення операційної системи. Оскільки програмне забезпечення менеджера кластера також встановлено на всіх вузлах кластера, забезпечується повне резервування.

У реальних кластерних додатках виконувані файли Oracle встановлюються на локальні диски кожного вузла, а екземпляр запускається на кожному вузлі кластера. Зверніть увагу, що якщо ви підтримуєте кластерну файлову систему, то буде встановлена ​​тільки одна копія власного додатка Oracle. Всі екземпляри мають рівний доступ до всіх даних і можуть обробляти будь-які транзакції. Таким чином, реальні кластери додатків забезпечують повне резервне копіювання програмного забезпечення бази даних.

Якщо вузол виходить з ладу, Oracle повинен змінити свій статус членства в кластері. Це відомо як реорганізація кластера, і зазвичай це відбувається швидко. Тривалість реорганізації кластера пропорційна кількості тих, що вижили вузлів кластера.

Global Cache і Global Enqueue Service надають програмні інтерфейси менеджера кластера для програмного забезпечення і відкривають членську карту кластера для випадків Oracle, коли вузли додаються або видаляються з кластера. Процес на кожному вузлі кластера зв'язується з Cluster Manager на відповідних вузлах і надає цю інформацію відповідним екземплярів Oracle.

Elmon також надає ще одну корисну функцію: постійна відправка повідомлень з сайту, де він працює, і часта запис на загальний диск. Відсутність цих функцій в будь-якому вузлі вказує на те, що ці вузли перестають бути членами кластеру. Цей збій призводить до зміни стану членства в кластері, і Elmon ініціює дії по відновленню, які включають повторне освоєння ресурсів Global Cache Service і Global Enqueue Service і відновлення екземпляра.

На цьому етапі середу кластера додатків знаходиться в систематичній паузі, і більшість клієнтських транзакцій призупиняється до тих пір, поки Oracle не відновить обробку.

Процес відновлення членського примірника гарантує, що всі члени кластера функціональні робить наступне:

* Забезпечити життєздатність зв'язку між усіма членами та можливість всіх членів реагувати.
* Забезпечує механізм видалення членів, які не є активними, арбітражує членство та усуває членів, які вирішили більше не належати до кластеру.
* Голосування за членство за допомогою контрольного файлу. Кожен учасник записує растровий файл у керуючий файл. Це частина запису про хід пункту пропуску.
* Видаляє членів на основі збою зв'язку, сприйме членів як закінчився термін дії, якщо вони не надають нормального періодичного серцебиття до керуючого файлу або якщо вони не відповідають на запит про їх статус.
* Встановлює голоси про членство та записує арбітром протокол результатів голосування.

Усі екземпляри читають протокол результатів голосування. Якщо члена немає на карті членства, припускає, що вузол закінчився. Надається відповідна діагностична інформація. Оскільки наразі налаштований, усі члени чекають нескінченно часу на повідомлення про закінчення терміну дії вузла. Примусового вилучення екземплярів немає.

### 1.2.4 Конфігурації відмовостійкого кластеру

У цьому розділі розглядаються конфігурації відмовостійкості реальних кластерів додатків:

* Конфігурація вузла за замовчуванням;
* Основні конфігурації високої доступності;
* Спільна конфігурація вузла високої доступності;

Конфігурація n-вузла є середовищем за замовчуванням. Всі вузли кластера беруть участь в обробці клієнтських транзакцій, і клієнтські сеанси можуть бути збалансовані під час з'єднання. Час відгуку оптимізується для існуючих ресурсів кластера, таких як процесор і пам'ять, шляхом розподілу навантаження між вузлами кластера для створення високодоступних середовищ [12].

У разі збою вузла екземпляр іншого іншого вузла виконує необхідні дії по відновленню. Клієнти бази даних в невдалому екземплярі можуть бути збалансовані по навантаженню між збереженими примірниками кластера n-мінус 1. Збільшення навантаження на кожен зі збережених примірників можна мінімізувати і підвищити доступність, зберігаючи час відгуку в допустимих межах. У цій конфігурації навантаження програми бази даних може бути розподілена по всіх вузлах і, отже, забезпечує оптимальне використання ресурсів комп'ютера кластера.

Ви можете легко налаштувати базову систему високої доступності для реальних групових застосунків в середовищах з двома вузлами. Первинний екземпляр на одному вузлі приймає підключення користувача, тоді як вторинний екземпляр на іншому вузлі приймає з'єднання, коли первинний вузол виходить з ладу або коли спеціально вибраний параметром. Ви можете налаштувати це вручну, керуючи маршрутизацією транзакцій до конкретних екземплярів. Проте, True Application Clusters надають первинну / вторинну функцію конфігурації, щоб зробити це автоматично.

Налаштуйте функцію первинного / вторинного примірника, встановивши для цього параметра значення 1. У середовищі з двома вузлами екземпляр, який першим монтує базу даних, приймає на себе основну роль примірника. Інший екземпляр бере на себе роль вторинної влади. Якщо основний екземпляр не працює, то додатковий примірник приймає на себе основну роль. Коли невдалий екземпляр повертається в активний стан, він приймає на себе роль вторинного примірника.

Вторинний екземпляр стає основним тільки після того, як Cluster Manager повідомляє його про збої основного примірника, але до початку процесів настройки Global Cache і Global Enqueue Service, а також відновлення кеша і транзакцій. Перенаправлення на збережену копію є прозорим; програмування додатків не потрібно. Все, що вам потрібно зробити, це внести невеликі зміни в конфігурацію рядків підключення клієнта.

У первинної / вторинної конфігурації обидва примірники працюють в той же час, що і в будь-якому середовищі реальних кластерів додатків з n-вузлами. Однак користувачі додатків бази даних підключаються тільки до призначеного первинного примірнику. Основний вузол вивчає всі ресурси Global Cache Service і Global Enqueue Service. Це зводить до мінімуму обмін даними між вузлами і забезпечує рівень продуктивності, який практично можна порівняти з традиційною одноузловой базою даних.

Вторинний екземпляр може використовуватися спеціально налаштованими клієнтами, відомими як адміністративні клієнти, для операцій пакетної звітності або завдань адміністрування бази даних. Це забезпечує деякий рівень використання для другого вузла. Це також може допомогти вивантажити ємність процесора з основного примірника і виправдати інвестиції в надлишкові вузли.

Функція первинної / вторинної конфігурації працює як на виділеному сервері, так і на загальному сервері. В сучасних конфігураціях високої доступності середовища виділених серверів не використовують реєстрацію «Прослуховувач всередині простору». Запити на підключення, зроблені слухачеві певного примірника, можуть бути пов'язані тільки зі службою цього примірника. Це поведінка аналогічно конфігурації n-вузла за замовчуванням в середовищах виділених серверів.

## 1.3. Оптимальне резервування

### 1.3.1 Завдання оптимального резервування комп’ютерних систем

Основуючись на статті [13], резервування є одним з простих і досить ефективних методів підвищення надійності. Однак при резервуванні виникає завдання не тільки забезпечити задані показники надійності, але добитися цього якомога більш економічно, з найменшими сумарними витратами на резервні елементи для системи в цілому, або при заданих ресурсних обмеженнях досягти максимально можливої ​​надійності. Завдання оптимального резервування частіше виникає в відмовостійких системах дозволяють користувачеві або функціональної програмі продовжувати роботу і тоді, коли в апаратних або програмних засобах системи виникають відмови. При проектуванні таких систем слід прагнути не тільки до досягнення необхідної їх надійності, але і до досягнення цієї надійності при мінімальних засобах, тобто до знаходження оптимального рішення. У відмовостійких комп'ютерних системах і машинах існує ряд параметрів, від яких залежить надійність системи. Так само як кількість резервних елементів, пристроїв або підсистем; параметри систем контролю і діагностики; характеристики системи програмного забезпечення; величини, що характеризують архітектуру, конфігурацію роботи системи та інші:

(1.1)

Надійність представляється у вигляді функціональної залежності від перерахованих параметрів. У якості подібних обмежують ресурсів можна розглянути вартість, масу, габаритні розміри, споживану потужність і т.п. Вибір виду обмежує ресурсу визначається конкретним типом системи і її призначенням. Часто виділяють одну найбільш важливу характеристику - вартість.

Зазвичай в задачах оптимального резервування передбачається, що вартість резерву для системи в цілому:

(1.2)

де mi - число резервних блоків в i-ой підсистемі КС; і крім того, сама вартість резерву i-й резервної групи визначається як:

(1.3)

де ci - вартість одного блоку в i-ой підсистемі КС. При наявності одного обмежує фактора (вартості) можливі постановки двох наступних завдань оптимального резервування.

1. Пряма задача. Роздільним резервуванням системи, що складається з m-резервних груп, домогтися того, щоб показник надійності був не менше заданого Rзад при мінімально можливої вартості резерву в цілому, тобто:

(1.4)

2. Зворотне завдання. Роздільним резервуванням системи, що складається з m-резервних груп, домогтися того, щоб при максимально можливому показнику надійності системи R вартість всього резерву не перевищила заданого значення ззаду, тобто:

(1.5)

якщо в якості показника надійності вибрати ВБР Рс, то:

(1.6)

де Сi - вартість одного блоку в i-й підсистемі комп'ютера або КС;

mi - число резервних блоків в i-й підсистемі комп'ютера або КС;

Cзад - задане значення вартості резервних блоків машини або КС;

Pc - ймовірність безвідмовної роботи КС за час Т.

Суть оптимізаційної задачі, що полягає в підвищенні надійності системи шляхом резервування при обмеженнях на сумарну вартість, можна пояснити на двох простих приватних випадках. Припустимо, що всі елементи системи равнонадежни і в кожної резервної групі є рівно по одному основному елементу. У цьому випадку пріоритет щодо резервування спочатку отримують ті групи, елементи яких характеризуються найменшою вартістю. Якщо ж елементи мають рівну вартість, то спочатку слід резервувати найменш надійні резервні групи. У більш складних випадках, коли резервні групи містять різну кількість елементів, а самі елементи в різних групах розрізняються і за показниками надійності, і за вартістю, для визначення оптимального складу резервних елементів в системі потрібно використовувати спеціальні алгоритми розв'язання оптимізаційних задач.

Експериментальні завдання (задачі знаходження екстремуму функції min або max) з обмеженнями можуть бути вирішені аналітично (з використанням методу невизначених множників Лагранжа) і за допомогою чисельних методів: методу перебору і градіентного методу. При вирішенні оптимізаційних задач можуть бути обмеження на масу, вартість, габарити і інші характеристики системи.

### 1.3.2 Оптимальний розподіл резервів чисельними методами

Завдання оптимального резервування може бути вирішена не тільки аналітично (методом невизначених множників Лагранжа), але і численними методами. Чисельні методи визначення оптимального резерву дозволяють знайти більш точне рішення, і особливо ефективні при малому числі резервних підсистем. До чисельного методу відноситься метод перебору, коли порівнюють між собою всі можливі варіанти структури. Потім вибирають з них той, який найкраще відповідає встановленим вимогам по надійності. Однак число варіантів виходить дуже великим, тому метод перебору може бути використаний тільки в найпростіших випадках.

Позначимо кількість конкуруючих варіантів Nb. Для структури складної КС типу Nb визначається твором:

(1.7)

де - загальне число підсистем;

- максимально можливе число паралельних підсистем i-го типу.

Наприклад, n = 10 і = 10, при , тоді Nb = 1010, це практично виключає можливість перебору.

Розглянемо можливість скорочення числа варіантів при переборі. Введемо поняття домінуючий послідовності і розглянемо графік варіантів технічний рішень в координатах: вартість С - ймовірність відмови Q

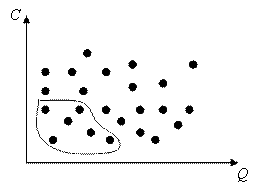


Рис. 1.3 Графічна ілюстрація домінуючою послідовності

Кожен варіант цьому графіку представляється точкою. З усіх варіантів заданої вартості, інтерес представляють тільки варіанти, що володіють мінімальною ймовірністю відмови Q. З усіх варіантів, при заданій ймовірності відмови Q, цікаві тільки варіанти, що відрізняються мінімальною вартістю С. Звідси випливає, що з усього безлічі варіантів інтерес представляють тільки ті, які знаходяться знизу і зліва.

Підмножина цих варіантів, перспективних з точки зору пошуку оптимального варіанта, називається домінуючою послідовністю. На рис. 1.3 домінуюча послідовність обведена. Потужність (кількість елементів) домінуючою послідовності зазвичай набагато менша потужності безлічі варіантів. Тому легко вибирати оптимальне рішення з варіантів, що входять в домінуючу послідовність. Чисельні методи дозволяють побудувати домінуючу послідовність поелементно, - до досягнення заданої надійності при мінімальній вартості або до досягнення мінімальної вартості при заданій надійності. При структурної оптимізації важливо знайти ефективний спосіб обмеження кількості розглянутих варіантів. Ефективним методом є градієнтний метод полягає в покроковому пошуку максимуму критерію типу Δlg(Q)/(ΔС), тобто відношення приросту логарифма ймовірності відмови системи до приросту вартості.

Метод найшвидшого метод дозволяє визначити частину елементів домінуючої послідовності, тобто деякі елементи можуть відмовитися пропущеними. Тому одержувані цим методом результати слід розглядати як наближені або проводити додатковий пошук.

Приклад оптимального розподілу резервів. Нехай є система, що складається з чотирьох підсистем, тобто n = 4. підсистеми характеризуються вартостями Ci і можливостями відмови за заданий час qi:

(1.8)

Потрібно побудувати систему, що володіє ймовірністю безвідмовної роботи Р ≥ 0,99 при мінімальній вартості С. Первісне стан системи, коли немає резервів, описуються вектором станів m = (m1 m2 m3 m4) = (1111). Порівняння варіантів (2111), (1211), (1121) і (1112) відбувається за критерієм Δlg (Q) / ΔС, що забезпечує максимум критерію. Використовуючи наведені вище формули знаходимо наближене рішення задачі. Величина оптимального розподілу резерву m = (5543).

## 1.4. Балансування навантаження

### 1.4.1 Необхідність балансувальника навантаження

Взаємодія кластерів забезпечується розподілом запитів від користувачів інтернет-ресурсів між серверами в режимі очікування. Якщо атаки роблять тимчасово неможливим доступ до ресурсів одного або декількох серверів, то навантаження буде розподілена між тими серверами, які залишаються в робочому стані. Основну роль в цьому розподілі грає балансувальник навантаження [13].

Балансувальник навантаження контролюють потік інформації між сервером і кінцевим пристроєм (ПК, ноутбук, планшет або смартфон). Не має значення, чи є це локальним сервером або сервером в центрі обробки даних або в загальнодоступній хмарі, він реалізований як фізичний сервер або віртуалізований. Балансувальник навантаження дозволяє серверам ефективно переміщати дані, запобігає їх перевантаженню і оптимізує використання ресурсів доставки додатків. Балансувальник також виконує поточні перевірки працездатності серверів, щоб якомога швидше видалити непрацюючі сервери з пулу, поки вони не будуть відновлені. Деякі балансувальники навантаження навіть запускають створення нових віртуалізованих серверів додатків, щоб впоратися зі зростаючим попитом.

### 1.4.2 Види балансувальників навантаження

Існує багато методів балансування навантаження з використанням різних алгоритмів для застосування у кожної конкретної ситуації [14]:

* Round Robin. Простий метод збалансування завантаження серверів. Кілька однакових серверів налаштовано для надання одних і тих же послуг. Усі налаштовані на використання одного і того ж доменного імені Інтернету, але кожен має унікальну IP-адресу. DNS-сервер містить список усіх унікальних IP-адрес, пов’язаних із доменним іменем Internet. Коли надходять запити на IP-адресу, пов’язану з іменем домену Інтернет, адреси повертаються послідовно обертовим чином.
* Weighted Round Robin. Зважений круглий Робін ґрунтується на простому методі балансування навантаження Round Robin. У зваженій версії кожному серверу в пулі надається статичне числове зважування. Сервери з більш високими рейтингами отримують більше запитів, надісланих їм.
* Least Connection. Ні Round Robin, ні зважений Round Robin не беруть до уваги поточне завантаження сервера при розподілі запитів. Метод найменшого з'єднання враховує поточне завантаження сервера. Поточний запит надходить на сервер, який обслуговує найменшу кількість активних сесій у поточний час.
* Weighted Least Connection. На основі методу найменшого з'єднання. Як і у методі зваженого кругообігу, кожному серверу присвоюється числове значення. Балансир навантаження використовує це при розподілі запитів на сервери. Якщо два сервери мають однакову кількість активних з'єднань, то новому запиту буде призначено сервер з більшою вагою.
* Agent-Based Adaptive Load Balancing. Кожен сервер у пулі має агент, який повідомляє про своє поточне навантаження балансиру навантаження. Ця інформація в реальному часі використовується при визначенні того, який сервер найкраще обробляти запит. Це використовується в поєднанні з іншими методами, такими як зважене найменше з'єднання.
* Chained Failover (Fixed Weighted). У цьому способі заздалегідь визначений порядок серверів налаштований у ланцюзі. Усі запити надсилаються на перший сервер ланцюга. Якщо він не може прийняти більше запитів, наступний сервер у ланцюжку надсилає всі запити, тоді третій сервер. І так далі.
* Weighted Response Time. Цей метод використовує інформацію відповіді з перевірки стану сервера, щоб визначити, який сервер реагує найшвидше в певний час. Наступний запит на доступ до сервера надсилається на цей сервер. Це гарантує, що будь-яким серверам, які знаходяться під великим навантаженням і які реагуватимуть повільніше, не надсилатимуться нові запити. Це дозволяє навантаженню з часом вирівняти наявний пул сервера.
* Source IP Hash. Цей алгоритм поєднує вихідні та цільові IP-адреси клієнта та сервера для створення унікального хеш-ключа. Ключ використовується для розподілу клієнта до певного сервера. Оскільки ключ можна відновити, якщо сеанс розбитий, запит клієнта спрямовується на той самий сервер, який він використовував раніше. Це корисно, якщо важливо, щоб клієнт підключився до сеансу, який все ще активний після відключення. Наприклад, для зберігання предметів у кошику для покупок між сеансами.

## 1.5. Кластеризація серверів

### 1.5.1 Необхідність застосування кластеризації серверів

Основним блоком кластера є окремий комп'ютер, також званий «вузлом». Кластери можуть збільшуватися в розмірах, додаючи більше машин. Кластер в цілому буде потужнішим, ніж швидше працюватимуть окремі комп'ютери, і тим швидше буде швидкість їх сполуки. Крім того, операційна система кластера повинна максимально ефективно використовувати доступне обладнання у відповідь на зміну умов [15]. Це стає більшою проблемою, якщо кластер складається з різних типів апаратного забезпечення ( "неоднорідний" кластер), якщо конфігурація кластера непередбачувано змінюється (машини, які підключаються і залишають кластер) і навантаження не може бути передбачена заздалегідь.

### 1.5.2 Види кластеризації

Відмовостійкий кластер **-** це кластер високої доступності, в якому при відмові одного сервера його функції переймають на себе інші машини в кластері. Таким чином, сервіси та додатки продовжують працювати без зупинки завдяки апаратної надмірності [15]. Щоб побудувати отказоустойчивую структуру, потрібно мінімум два фізичних сервера з системами зберігання даних. В рамках СГД дисковий простір розподіляється між усіма серверами, а для управління віртуальними машинами використовують спеціальне програмне забезпечення - гіпервізор. Віртуальні машини, запущені на серверах кластера, працюють за наступним принципом: якщо одна з них виходить з ладу, в роботу автоматично включається друга.

Застосування: всюди, де потрібна безперервна робота бізнес-сервісів і підтримка важливих баз даних (банк, біржа, цілодобове виробництво), електронні торгові майданчики.

Переваги: ​​надійність, висока доступність сервісів і додатків, скорочення втрат через простої в роботі.

В межах цього кластеру навантаження, яку створюють сервіси і додатки, рівномірно розподіляється між доступними машинами. Так виключається простий одного сервера, поки другий працює на межі можливостей. Для розподілу запитів в кластері використовується один або кілька вхідних обчислювальних вузлів, через які завдання перенаправляються з однієї машини на іншу.

Застосування: ЦОД, комплекси для ERP / CRM-систем, сервіси білінгу в телекомунікаційних системах.

Переваги: ​​масштабованість, можливість використання недорогих серверів стандартної архітектури.

Кластер побудований на основі декількох серверів, об'єднаних високошвидкісними лініями передач і спеціальним ПО. На виході утворюється єдина система для комплексних обчислень. Кожен сервер в такому кластері обробляє завдання, яка автоматично виділяється йому із загального обсягу роботи.

Застосування: аналітика, збір і обробка даних для Big Data, системи штучного інтелекту, нейронні мережі.

Переваги: ​​висока продуктивність в ресурсномістких завданнях, виконання паралельних обчислень.

## 1.6. Висновки до розділу

В даному розділі дипломної роботи були викладені основні теоретичні відомості стосовно захисту інформаційно-комунікаційної системи та проведений аналіз останніх досліджень і публікацій та які проблеми ще не були вирішеними.

Також, були розглянуті такі питання, як необхідність використання кластеризації серверів та для чого це потрібно, а також необхідність використання балансувальника навантаження для більш відмовостійкій поведінці розгорнутої інфраструктури.

Також, ми проаналізували методи кластеризації і було вирішено, що одного із реалізованих методів є недостатньо, тож щоб повисити рівень відмовостійкості, нам необхідно реалізувати як кластерну відмовостійку систему, так і балансувальник навантаження. За основу ми візьмемо сервер на базі Oracle Glassfish Open Source Server 4.0 і усі іспити будемо проводити саме на ньому.

Слід відмітити, що один із найважливіших моментів відмовостійкості є резервування, а отже ми також його реалізуємо у наступному розділі.

**РОЗДІЛ 2**

# РЕАЛІЗАЦІЯ ТА НАЛАШТУВАННЯ ВІДМОВОСТІЙКОГО КЛАСТЕРУ

У цьому розділі буде проведений аналіз по швидкодії операційних систем та як вони себе ведуть у різних ситуаціях та базі Oracle Glassfish Open Source Server 4.0. По результатам цього аналізу ми виберемо найоптимальнішу операційну систему, швидкодія якої буде найпродуктивнішою.

Наступним кроком, ми будемо аналізувати нашу серверну інфраструктуру та вирішувати проблеми, зв’язані із ними. Головна наша ціль – зробити найбільш ефективну відмовостійку кластерну систему з балансуванням навантаження та резервацією.

Завдяки експериментам із побудовою серверної інфраструктури буде знайдено найоптимальніше рішення. Будуть розглянуті різні варіанти по побудові і реалізації серверних систем та аналіз іх плюсів та мінусів.

Оскільки ми будемо використовувати виключно безкоштовні програми, які мають відкритий вихідній код, то необхідно буде створити усе це не маючи цього функціоналу у самому сервері Oracle Glassfish Open Source Server 4.0 окрім відмовостійкого кластеру.

Не менш важливим завданням є якомога зекономити та понизити кількість витрат на реалізацію відмовостійкого кластеру з балансуванням навантаження та резервним копіюванням необхідних файлів.

## 2.1. Конфігурація обладнання

Для перевірки та тестування відмовостійкого кластеру був створений тестовий полігон у складі декількох серверів, кофігурацію якого можна побачити на таблиці 2.1.

На таблиці 2.2 ми бачимо конфігурацію іншого серверу, який ми будемо використувувати для балансувальника навантаження на базі NGINX. Він має майже ідентичку апаратну конфігурацію окрім оперативної пам’яті, тут вона є меншою, так як немає необхідності у великому об’єму оперативної пам’яті для балансувальника навантаження.

Ubuntu Glassfish Server Configuration Таблиця 2.1

|  |  |
| --- | --- |
| **Апаратний пристрій** | **Модель пристрою** |
| Процесор: | AMD A8-3870K |
| Відеокарта: | Radeon HD 6670 |
| Оперативна пам'ять: | 4хKingston DDR3-1866 8192MB |
| Жорсткий диск: | HDD SATA II/ SATA III – 5400-7200 Об/хв |
| Дисковий привід: | DVD-RW |
| Доповнення: | Кулер 120 мм (5 шт. + 2 підсвідки) |
| Кулер проц 80мм |

Ubuntu Server (NGINX) Configuration Таблиця 2.2

|  |  |
| --- | --- |
| **Апаратний пристрій** | **Модель пристрою** |
| Процесор: | AMD A8-3870K |
| Відеокарта: | Radeon HD 6670 |
| Оперативна пам'ять: | 1хKingston DDR3-1866 4096MB |
| Жорсткий диск: | HDD SATA II/ SATA III – 5400-7200 Об/хв |
| Дисковий привід: | DVD-RW |
| Доповнення: | Кулер 120 мм (5 шт. + 2 підсвідки) |
| Кулер проц 80мм |

Наступною проблемою, яку треба було вирішити - це вибір операційної системи.

## 2.2. Вибір операційної системи

Windows - це серія операційних систем, комп'ютерна операційна система, розроблена Microsoft для персональних настільних пристроїв або комп'ютера. Кожна операційна система оснащена графічним інтерфейсом користувача з робочим столом, що дозволяє користувачеві переглядати всі файли, відео і т.д. Тож ОС Windows поставляється майже з усією компанією, яка виготовила ПК або ноутбуки [16]. Linux - це операційна система з відкритим кодом на основі UNIX, створена у дев’яносто першому році. Це програмне забезпечення, яке розміщується під усім іншим програмним забезпеченням на комп'ютері. Користувачі можуть змінювати існуючий код і створювати з нього дистрибутиви, оскільки це операційна система з відкритим кодом. Linux в основному використовується як сервер - оскільки більшість веб-сторінок через Інтернет генеруються з серверів Linux [17].

### 2.2.1 Ключові відмінності Windows та Linux

* Linux є операційною системою з відкритим кодом, тоді як ОС Windows є комерційною.
* Linux має доступ до вихідного коду та змінює код відповідно до потреби користувача, тоді як Windows не має доступу до вихідного коду.
* Linux працюватиме швидше, ніж останні видання Windows, навіть із сучасним середовищем робочого столу та особливостями операційної системи, тоді як Windows старіші.
* Дистрибутиви Linux не збирають дані користувачів, тоді як Windows збирає всі відомості про користувача, які призводять до проблем конфіденційності.
* Linux є більш надійним, ніж Windows, так як в Linux ми можемо вбити додаток, якщо вони повісили через x kill команду, тоді як у Windows нам потрібно спробувати кілька разів, щоб його вбити.
* Linux підтримує широкий спектр вільного програмного забезпечення, ніж Windows, але Windows має велику колекцію програмного забезпечення для відеоігор.
* В Linux програмне забезпечення майже безкоштовне, оскільки всі програми, утиліти, складні програми, такі як відкритий офіс, безкоштовні, але у Windows також є безліч безкоштовних програм та утиліт, але більшість програм є комерційними.
* Linux дуже захищений, тому що легко визначити помилки та виправити, тоді як Windows має велику базу користувачів і стає ціллю для розробників вірусів та шкідливих програм.
* Linux використовується корпоративними організаціями як сервери та операційна система з метою безпеки в Google, Facebook, Twitter і т.д.
* У цій ситуації Linux та Windows мають однаковий пріоритет перед технічною підтримкою та підтримкою драйверів.
* Одна із основних відмінностей полягяє у швидкості випуску нових версій операційних систем. У Windows наразі відмовилися від релізів нових версій операційних систем заради того, щоб реалізувати усе на 10-й версії та підтримувати дану операційну систему. Якщо брати у приклад Ubuntu, як один із найпоширеніших дистрибутивів операційної системи на ядрі Linux, то кожні декілька років випускається нова, більш вдосконалена версія операційної системи яка підримується близько 5-ти років.

Детальне порівняння Linux та Windows Таблиця 2.3

|  |  |  |
| --- | --- | --- |
| **Основа для порівняння** | **Linux** | **Windows** |
| Доступ до вихідного коду | У Linux користувач має доступ до вихідного коду ядра та змінює код відповідно до його потреби. У нього є свої переваги, такі як помилки в ОС виправлятимуться швидко, а недоліки, такі як розробники можуть скористатися будь-якою слабкістю в ОС, якщо вони знайдуть. | У Windows кожен користувач не матиме доступу до вихідного коду, до нього матимуть лише члени вибраної групи. |
| Різноманітність налаштувань | Linux має різні дистрибутиви, які налаштовуються на основі потреб користувачів. | У Windows дуже мало варіантів налаштування. |
| Ліцензування | У Linux з операційною системою, що має ліцензію GPL, користувачі можуть змінювати програмне забезпечення, можуть повторно використовувати будь-яку кількість систем і навіть вони можуть продавати модифіковану версію. | У Windows, що мають ліцензію Microsoft, користувачі не матимуть доступу до вихідного коду (не можуть змінювати програмне забезпечення) та на основі ряду ліцензій - ми можемо встановлювати лише на таку кількість комп’ютерів. |
| Командний рядок | У Linux командний рядок є дуже корисним інструментом для адміністрування та щоденних завдань, але для кінцевих користувачів це не має великої різниці. | У Windows ми маємо командний рядок, але не можемо використовувати як командний рядок Linux. Нам потрібно ввести cmd, тоді командний рядок відкриється. |
| Виконати рівень | Linux має вбудовану здатність зупинятися на різних рівнях запуску, при цьому ми можемо працювати за допомогою командного рядка та GUI, якщо у когось є проблеми. | Якщо у Windows виникає якась проблема, щоб її виправити, нам потрібно перезавантажити на рівні 3 запуску як адміністратор / root, щоб знайти та виправити проблему. |
| Зручність | Linux складний в установці, але має можливість легше виконувати складні завдання. | Windows надає користувачеві просту систему управління, але встановлення потребуватиме більше часу. |
| Підтримка | Linux має підтримку за допомогою величезної спільноти форумів / веб-сайтів користувачів та пошуку в Інтернеті. | У Windows є легкодоступна підтримка, онлайн-форуми / веб-сайти, і вона також має платну підтримку. |
| Оновлення | В Linux користувачі мають повний контроль над оновленнями, ми можемо встановлювати, коли нам потрібно, і це займе менше часу без перезавантаження. | У Windows оновлення надходять у незручні моменти, наприклад, ви даєте друк на принтер, але раптом з’явиться спливаюче оновлення, яке змушує користувачів засмучуватися та потребує більше часу на встановлення. |
| Безпека | Linux є більш безпечним, ніж Windows, де хакерам або розробникам вірусів буде важко пробити Linux. | Windows є основною ціллю для розробників вірусів та шкідливих програм, і вона є найбільш вразливою без антивірусного програмного забезпечення. |

### 2.2.2 Тестування GF серверу на кожній з ОС

Оскільки наша ціль – знаходження найоптимальнішого рішення на базі Glassfish Open Source Server 4.0, було прийнято рішення зробити 3 тести на різних платформах та оцінити швидкодію різних сценаріїв на них: тест на запуск системи, тест на відновлення вузла чи серверу та вимкнення серверу.

У тесті на запуск системи ми заміряємо скільки часу знадобиться серверу щоб запустити додаток у Glassfish Open Source Server 4.0.

У тесті на відновлення системи ми перевіряємо наскільки швидко додаток знову буде доступним після перезапуску ноди у Glassfish Open Source Server 4.0.

Останній тест – це тест на вимкнення серверу, у якому ми перевіряємо наскільки швидко нода буде вимкненна у Glassfish Open Source Server 4.0.

Усі ці тести були вибрані на основі частоти використання цих функцій і вони є основними, щоб визначити найоптимальніше рішення для роботи із сервером Glassfish Open Source Server 4.0.

Було прийнято рішення використати усі можливі операційні системи з різними процесорами, щоб результат був найбільш об’єктивним, а саме:

* Windows Server 2016
* Ubuntu 18.04
* macOS Mojave

Перший наш тест – це запуск Glassfish Open Source Server 4.0 серверу та аналіз швидкодії. Кожен тест виконувався 3 рази та брався середній результат.

Рис. 2.1 Аналіз швидкодії запуску GF сервера

Як результат, ми бачимо, що найбільш швидка система на базі Glassfish Open Source Server 4.0 має Ubuntu із 8-ми ядерним процесором i7, коли ж Windows Server майже на останніх позиціях.

Наступний тест – відновлення працездатності Glassfish Open Source Server 4.0. Аналогічно було зроблено 3 тести та був узятий середній результат.

Рис. 2.2 Аналіз швидкодії відновлення GF сервера

Як ми бачимо, результат майже такий самий – Ubuntu займає перші позиції, коли ж Windows Server є майже на останніх.

Переходимо до останнього тесту – вимкнення серверу Glassfish Open Source Server 4.0.

Рис. 2.3 Аналіз швидкодії вимкнення GF сервера

Як ми бачимо, результат нічим не відрізняється від минулих тестів, все також Ubuntu займає лідерські позиції, Windows Server, нажаль, не може похизуватися такими ж гарними результатами.

### 2.2.3 Висновок по вибору ОС

Як результат, найбільш оптимальне рішення буде розгорнути Glassfish Open Source Server 4.0 на базі Ubuntu 18.04. Щодо вибору апаратної частини, то найбільш розумним буде рішення по використанню 8-миядерного процесора від компанії AMD. Він коштує в рази менше у порівнянні із процесорами від компанії Intel, але швидкодію має майже однакову.

## 2.3. Реалізація та оптимізація серверної архітектури

Оскільки ми вже прийняли рішення по вибору апаратної частини та операційної системи для нашого Glassfish Open Source Server 4.0, наступним кроком буде знаходження найбільш оптимального рішення з точки зору налаштування та оптимізації роботи нашого серверу та його роботи із іншими вузлами (клієнтами). У цьому розділі ми розглянемо різні варіанти побудові серверної архітектури на базі Glassfish Open Source Server 4.0.

### 2.3.1 Односерверна архітетура

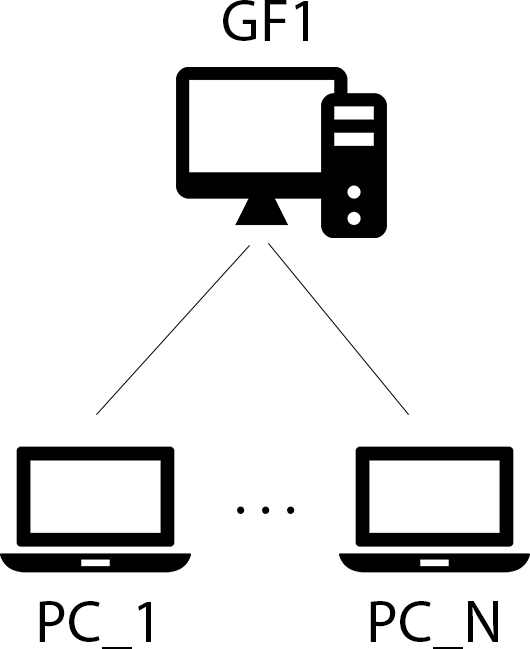


Рис. 2.4 Архітектура системи з одним сервером та клієнтами

Найбільш простим рішенням є використання усього одного серверу для усіх клієнтів. На перших погляд, це рішення має багато плюсів, таких як:

* Швидкість реалізації;
* Легка підтримка;
* Одна база;
* Низька вартість реалізації;

Це рішення дійсно має право на існування для маленьких корпорацій, де навантаження та клієнтів на сервер майже не буде, а його працездатність ні на що не впливатиме. Якщо ж компанія має достатньо велику кількість клієнтів, вона зіткнеться за наступними проблемами:

* При «падінні» одного серверу – вся система перестає працювати;
* Під навантаженням можуть бути проблеми, такі як підвисання, тощо;

### 2.3.2 Багатосерверна архітектура

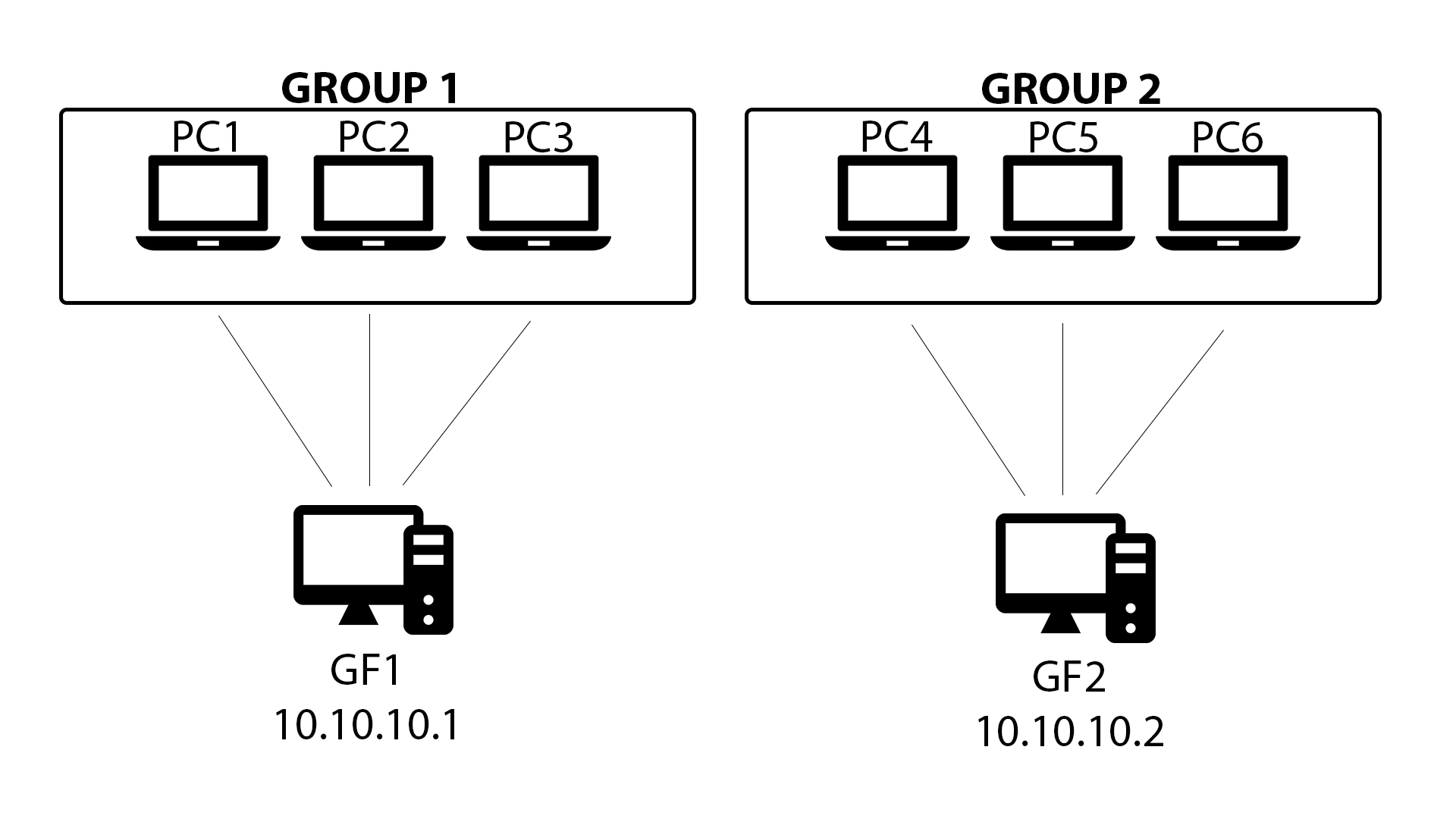


Рис. 2.5 Архітектура системи з декількома серверами та клієнтами

Для вирішення проблеми з навантаженням на сервери, в першу чергу приходить ідея – додати ще один сервер. Ми можемо вручну розподілити користувачів, поділивши їх на групи, та дати кожній групі свою адресу, тим самим ми закріпимо за кожної групою свій сервер. Тож, є лише один плюс цієї інфраструктури:

* Штучне/ручне розподілення навантаження;

Недоліків же набагато більше, ніж може здаватися:

* При «падінні» одного серверу – частина користувачів втрачає доступ до серверу;
* Своя база даних у кожного сервера;
* Ускладнене налаштування системи та складніша підтримка цієї системи;

### 2.3.3 Кластерна архітектура

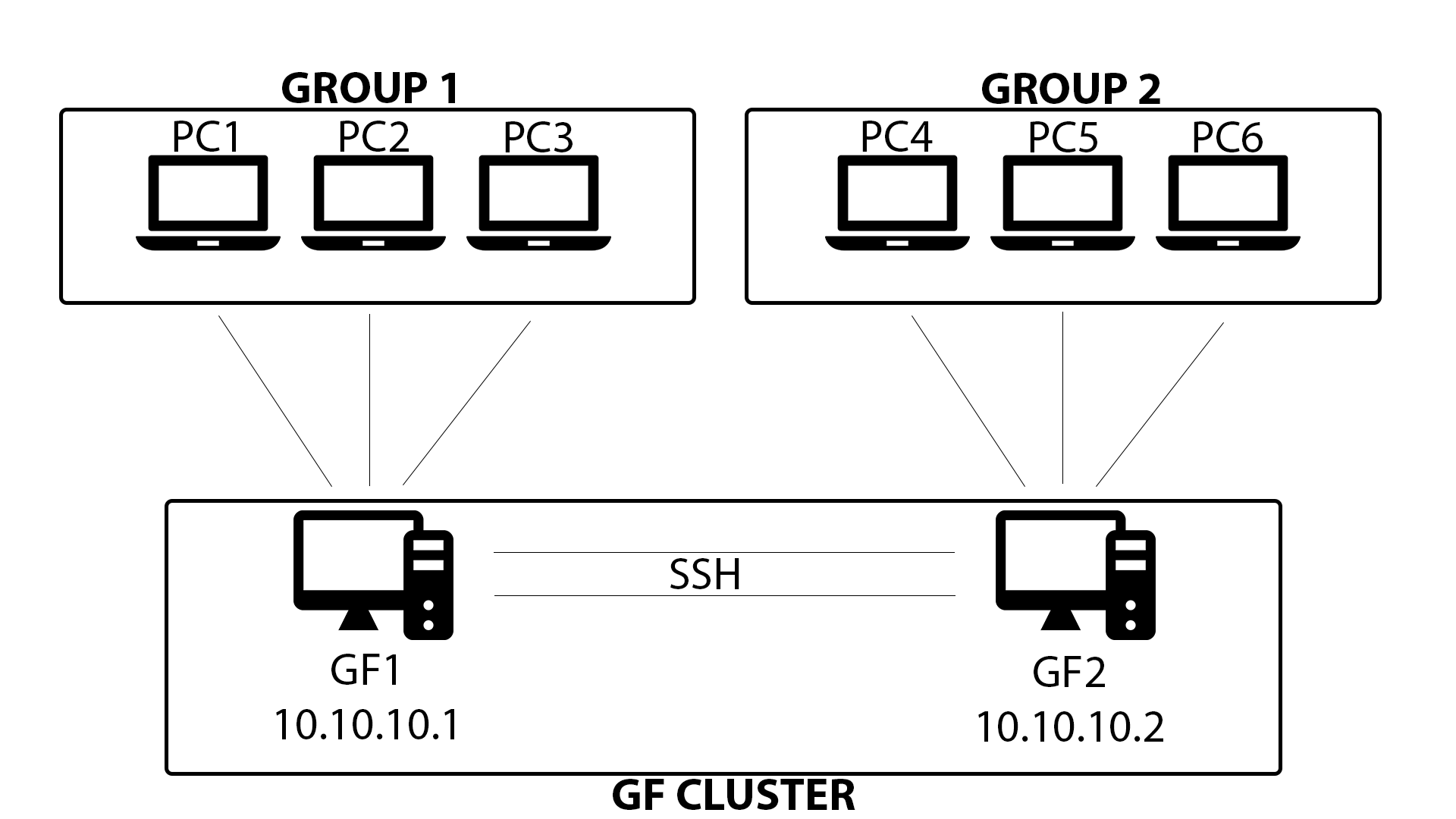


Рис. 2.6 Архітектура системи з кластером із серверів та клієнтами

Кластер GF – це іменований набір екземплярів сервера GlassFish, які обмінюються однаковими додатками, ресурсами та конфігураційною інформацією. Сервер GlassFish дозволяє вам управляти всіма примірниками в кластері як єдине ціле з одного хоста, незалежно від того, чи існують екземпляри одного і того ж хоста або різних хостів. Ви можете виконувати ті самі операції на кластері, які ви можете виконувати в некластеризованому екземплярі, наприклад, розгортаючи програми та створюючи ресурси.

Кластер забезпечує високу доступність через захист відмов, масштабованість та врівноваження навантаження.

Захист від аварій. Якщо екземпляр або хост у кластері виходять з ладу, GlassFish Server виявляє збій і відновлює стан сеансу користувача. Якщо для кластера налаштовано балансир навантаження, балансир навантажень перенаправляє запити від невдалого екземпляра до інших екземплярів кластера. Оскільки однакові програми та ресурси є у всіх екземплярах кластеру, екземпляр може перейти на будь-який інший екземпляр кластеру. Щоб дозволити відновити стан сеансу користувача, кожен екземпляр кластера надсилає дані стану пам'яті іншому екземпляру. Оскільки дані стану оновлюються в будь-якому випадку, вони реплікуються.

Масштабованість. Якщо потрібна збільшена ємність, ви можете додавати екземпляри до кластеру без перебоїв у роботі. Коли екземпляр додається або видаляється, зміни обробляються автоматично.

Балансування навантаження. Якщо екземпляри кластера розподіляються між різними хостами, навантаження може бути розподілене між хостами для збільшення загальної пропускної здатності системи.

Служба управління групою (GMS) - це інфраструктурна складова, включена для екземплярів кластеру. Коли GMS увімкнено, якщо кластерний екземпляр виходить з ладу, кластер і сервер адміністрування домену (DAS) знають про помилку і можуть вжити заходів, коли виникає збій. Багато можливостей сервера GlassFish залежать від GMS. Наприклад, GMS використовується за допомогою реплікації сеансу в пам'яті, обслуговування транзакцій та сервісу таймера.

**Створення та налаштування кластеру у Glassfish на базі Ubuntu**

Після встановлення усіх необхідних компонентів для запуску Glassfish Open Source Server 4.0 та їх налаштування, приступаємо до налаштування та створення кластеру. Усі сервери Glassfish Open Source Server 4.0 у нас будуть на базі Ubuntu Server 18.04 [18].

Для початку, заходимо через браузер до панелі управління сервером Glassfish Open Source Server 4.0. Вводимо свої дані для входу, такі як логін і пароль, після чого авторизуємося у системі.

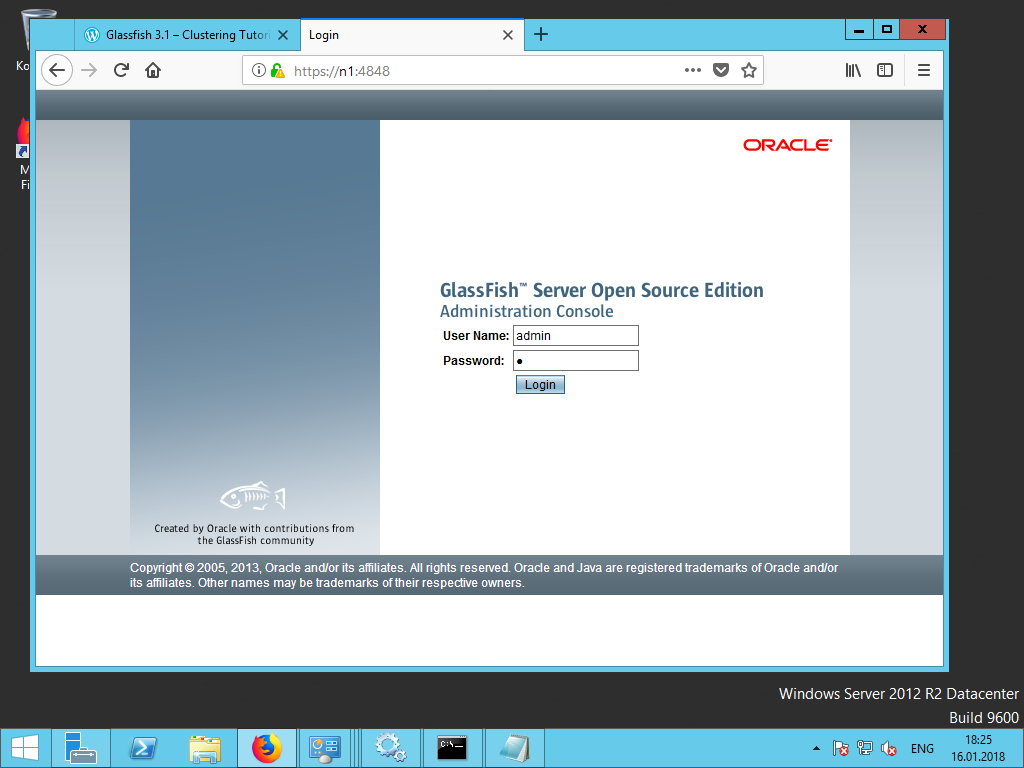


Рис. 2.7 Форма авторизації у Glassfish

Переходимо у вкладку «Clusters» та натискаємо кнопку «New..»

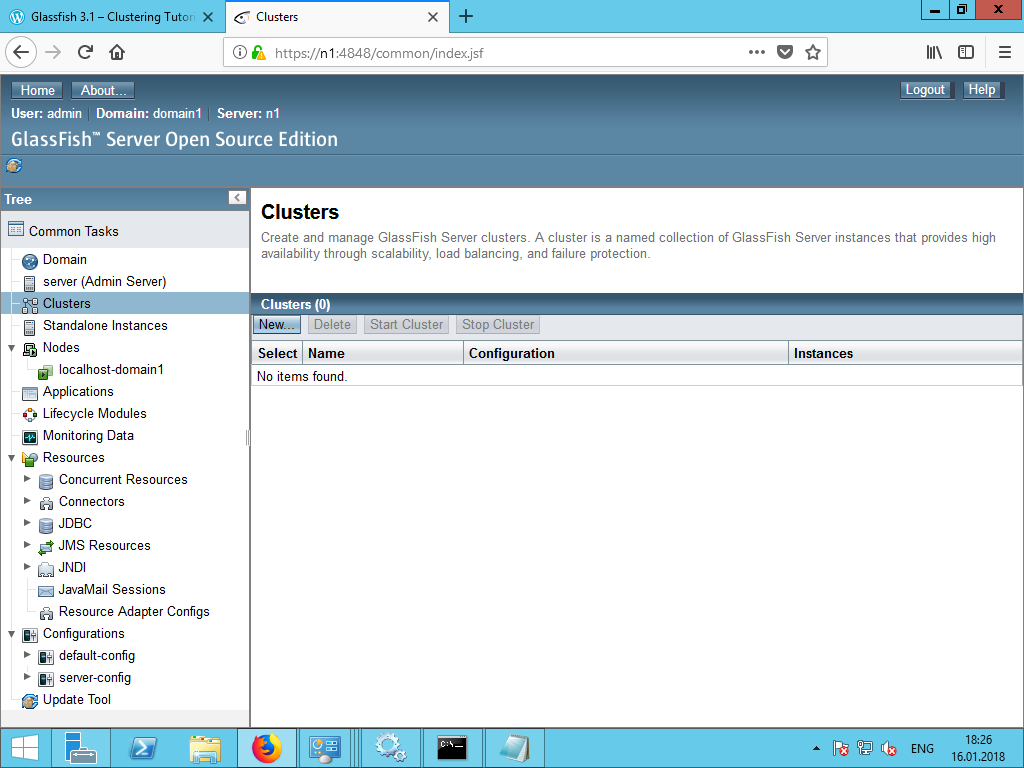


Рис. 2.8 Форма для управління та створення кластеру

У полі «Cluster Name» вводимо необхідне ім’я кластера (“c1”). Інші параметри не чіпаємо. Натискаємо «ОК».

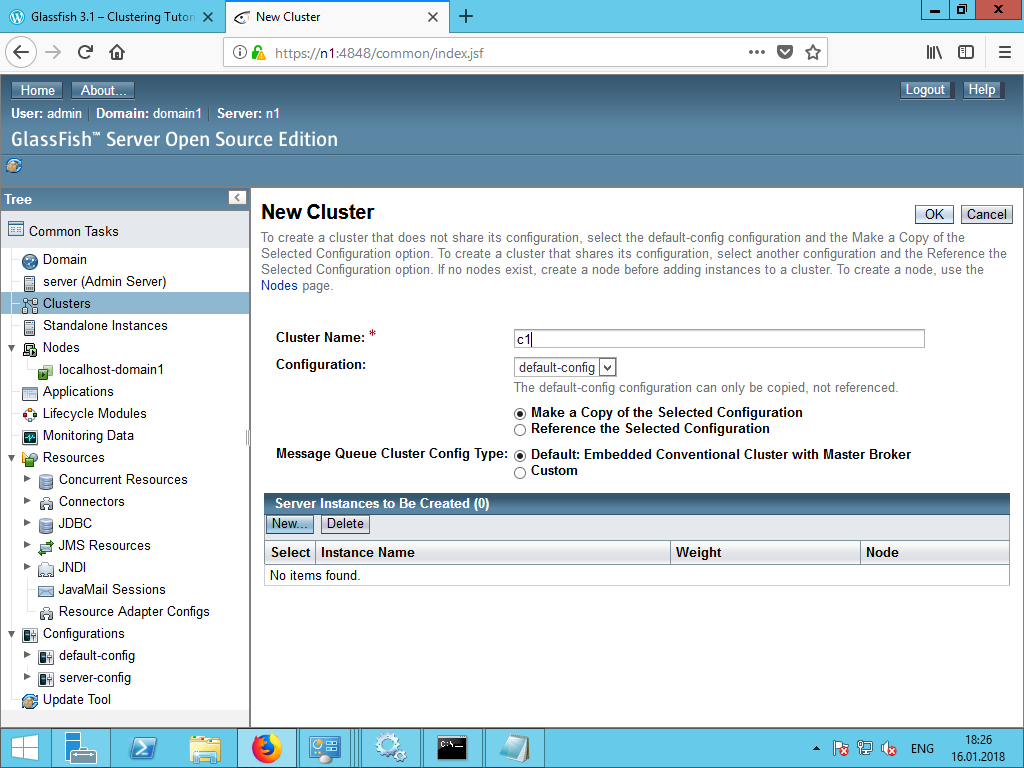


Рис. 2.9 Створення нового кластеру

Як ми бачимо, у нас з’явилось нове поле. Відкриємо параметри нашого кластера, натистувши на його ім’я (“c1”).

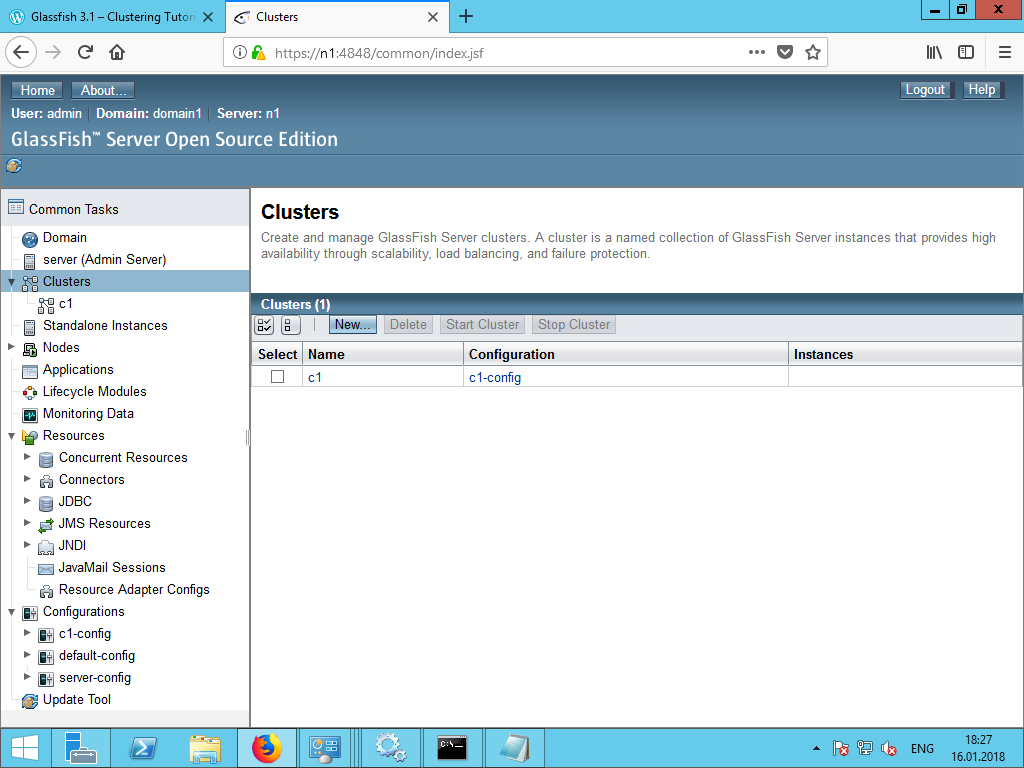


Рис. 2.10 Результат створення кластеру

Переходимо у вкладку «Instances».

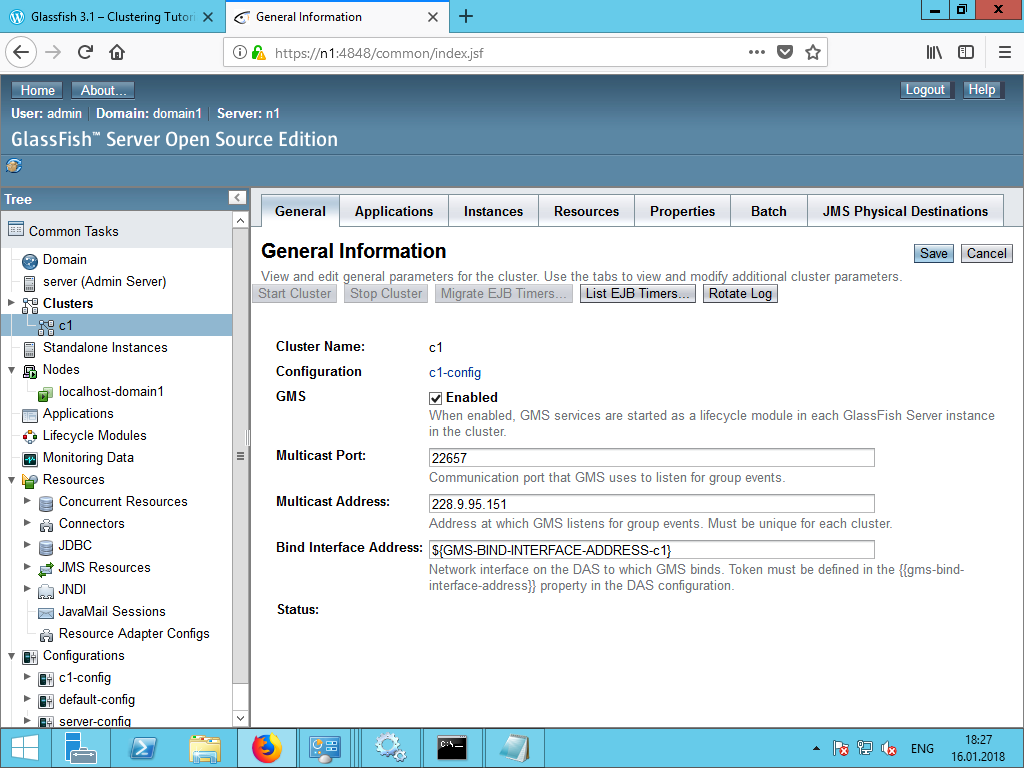


Рис. 2.11 Налаштування кластеру

Натискаємо кнопку «New…».

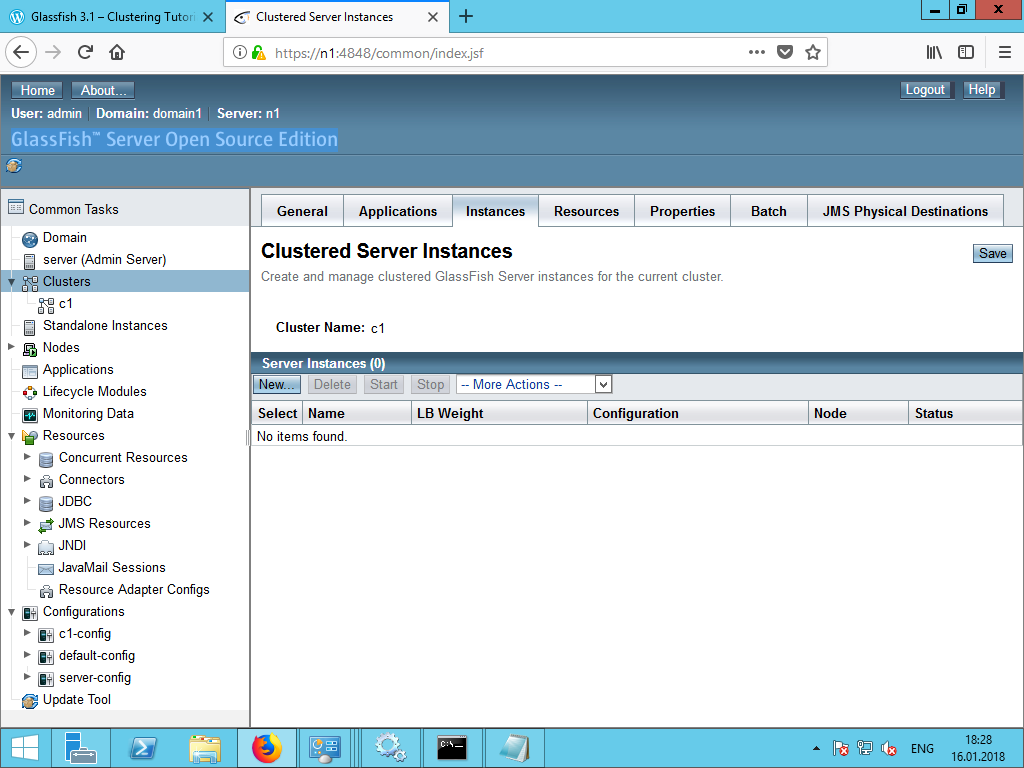


Рис. 2.12 Вкладка зі списком нод у кластері

Вводимо імя (“i1”). Інші параметри не чіпаємо. Натискаємо кнопку «ОК».

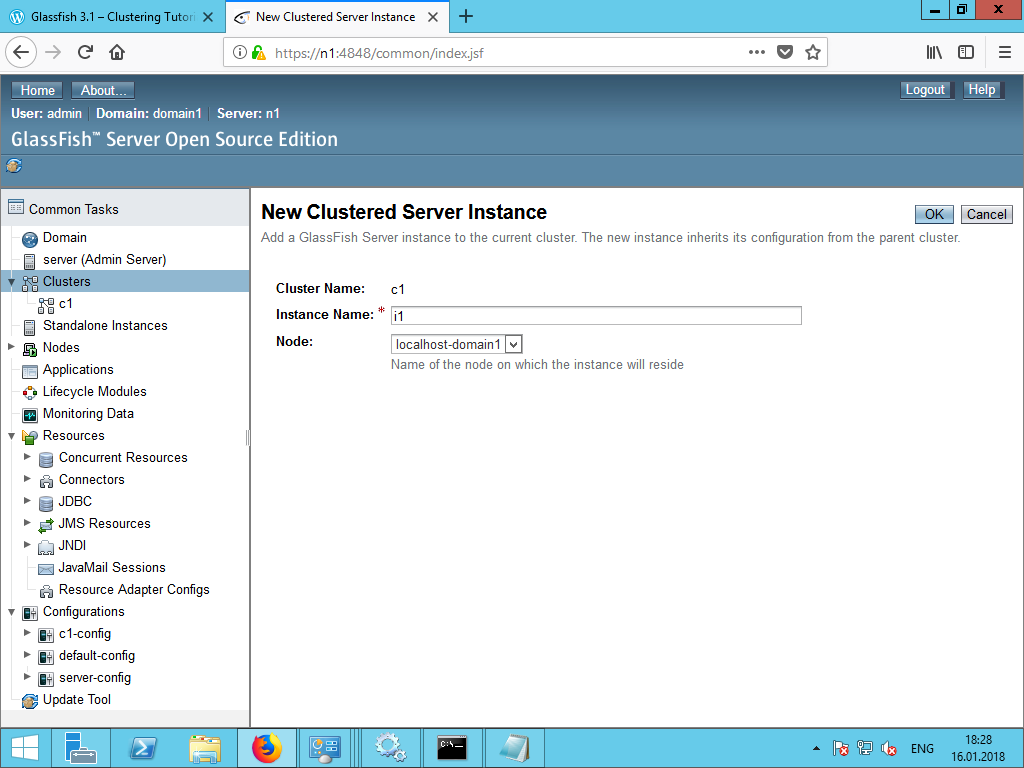


Рис. 2.13 Створення нової ноди у кластері

Як ми бачимо, з’явилось нове з’єднання. Натискаємо кнопку «Start Cluster».

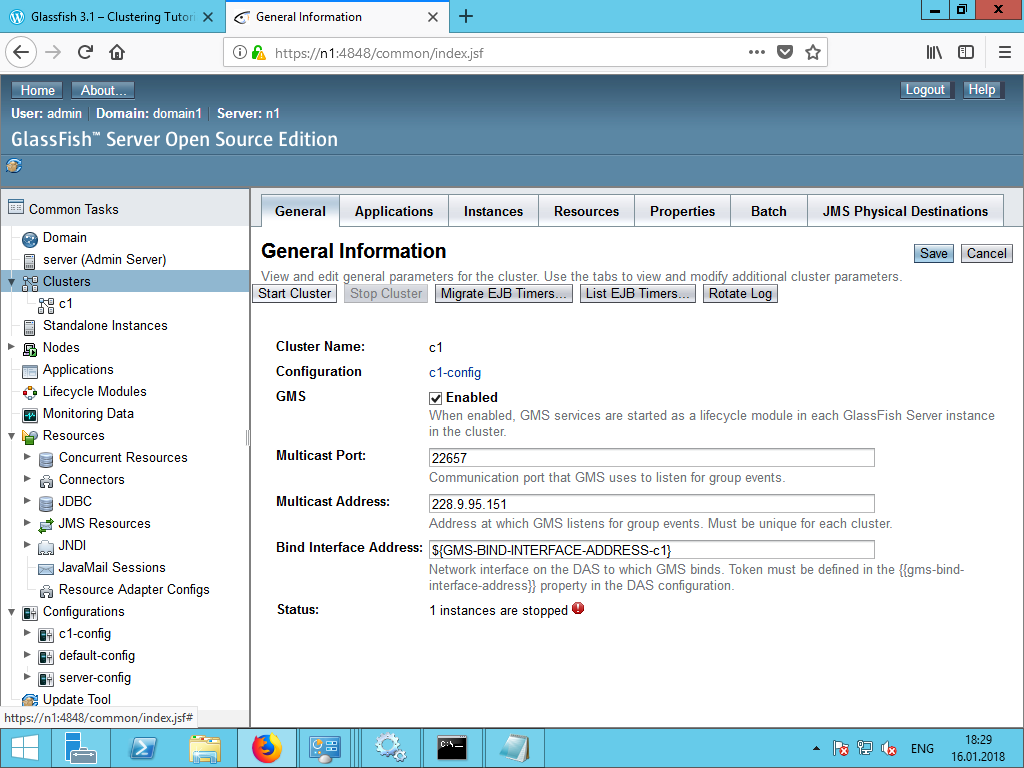


Рис. 2.14 Перевірка результатів після створення ноди

Після очікування, ми бачимо, що з’єднання встановлено.

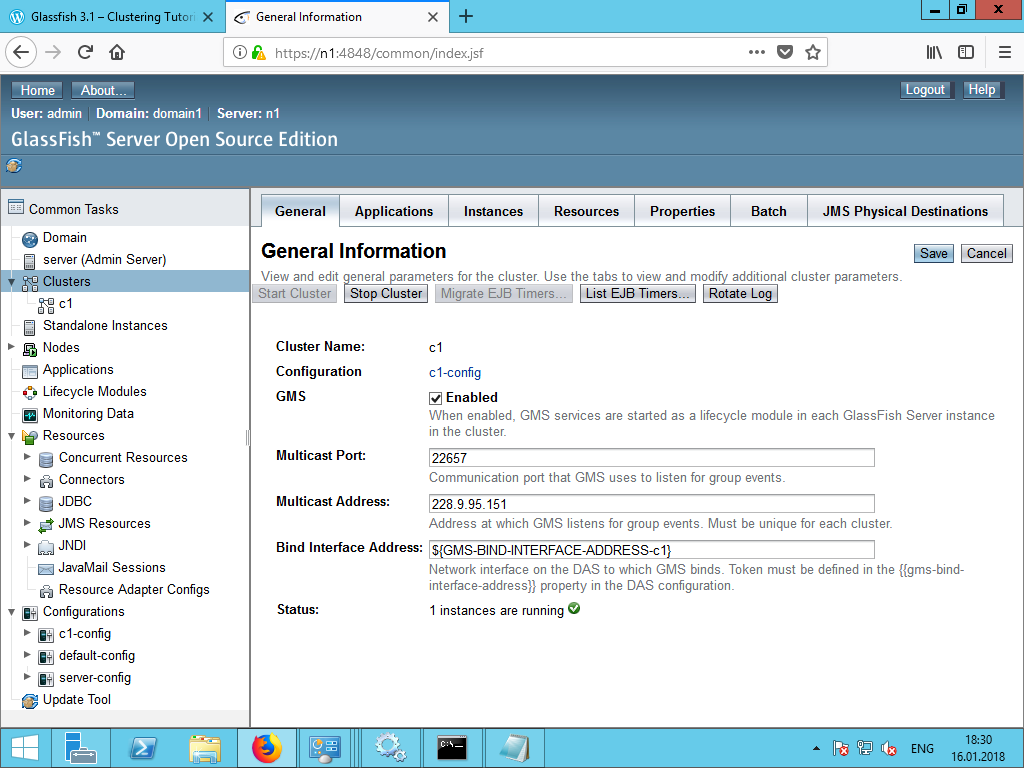


Рис. 2.15 Старт кластеру з однією нодою

На другому сервері (GF2) прописуємо команду asadmin --host n1 --port 4848 create-local-instance --cluster c1 i1. Якщо все було виконано правильно, то побачимо повідомлення: «Command create-local-instance executed successfully».

На першому сервері переходимо в панелі керування Glassfish Server (n1:4848) у вкладку «Nodes». Там ми маємо побачити, що у нас 2 ноди.

Далі, нам необхідно на другому сервері встановити SSH сервер. Це необхідно для встановлення зв’язку між серверами при кластеризації.

У метапакеті SSH міститься як клієнт, так і сервер, але при цьому, буде встановлений тільки сервер, так як клієнт вже є в Ubuntu за замовчуванням.

Для встановлення серверу в терміналі пропишемо наступну команду:

sudo apt-get install ssh

Після цього, запустимо наш сервер командою:

sudo service ssh start

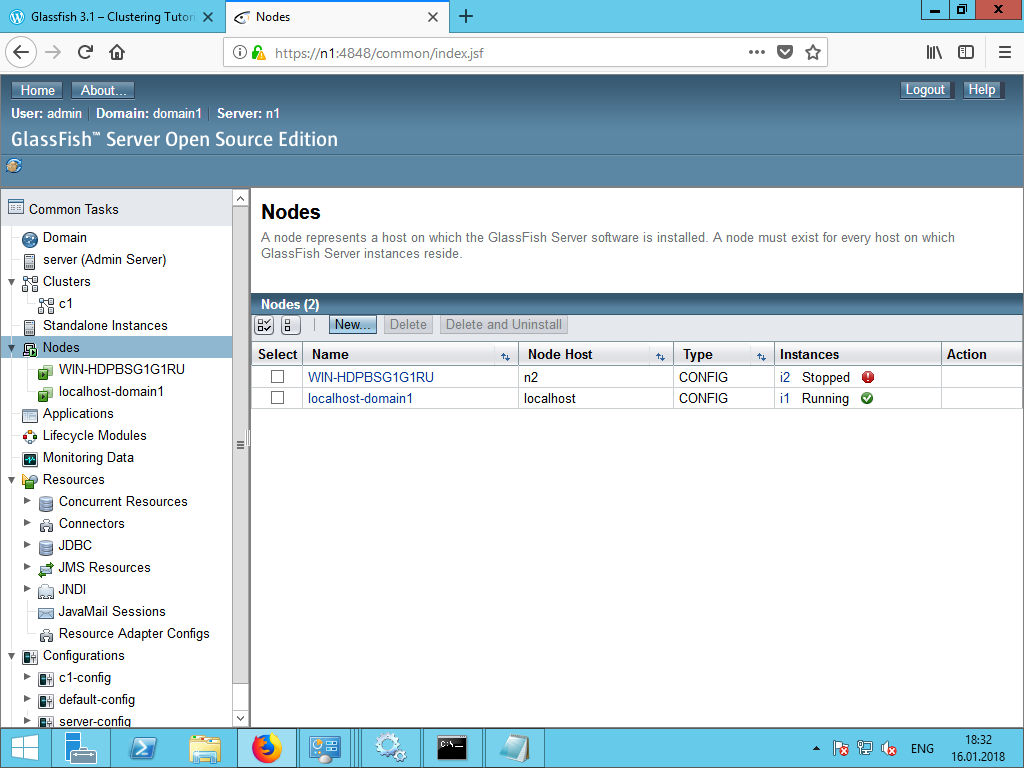


Рис. 2.16 Результат підключення 2-ї ноди до кластеру

З першого серверу відкриваєм панель налаштувань Glassfish Server (<http://n1:4848/>). Переходимо у вкладку «Nodes» та вибираємо ноду, яка має ім’я другого сервера (“WIN-…”).

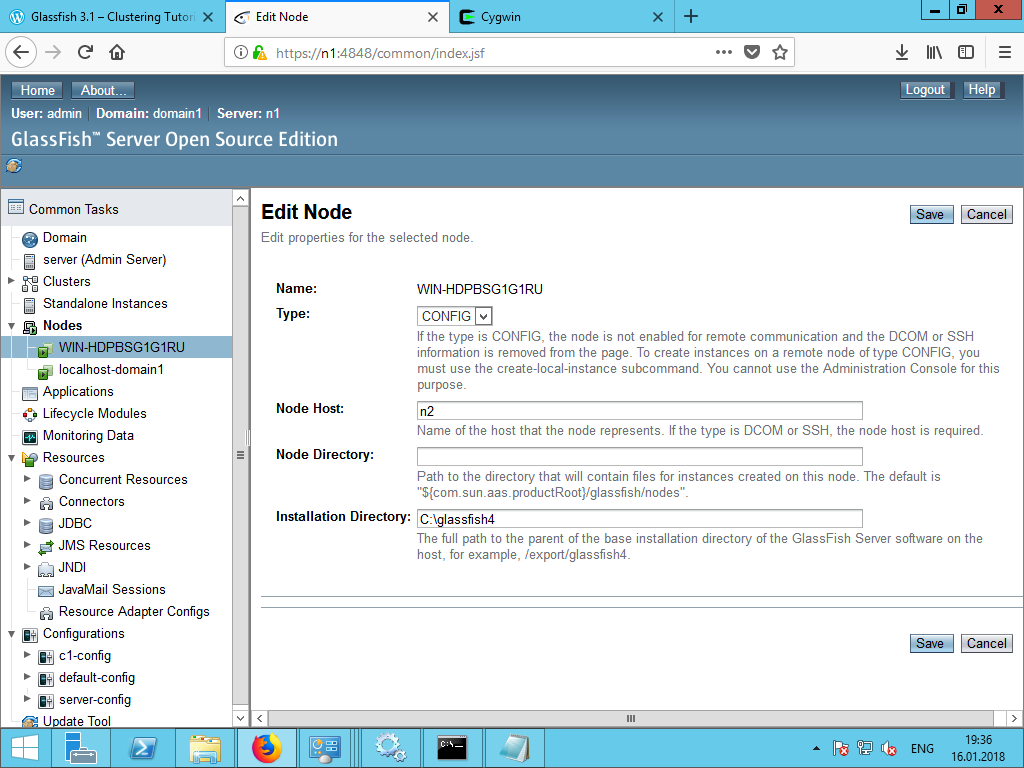


Рис. 2.17 Налаштування другої ноди

У вкладці «Type» вибираємо параметр «SSH».

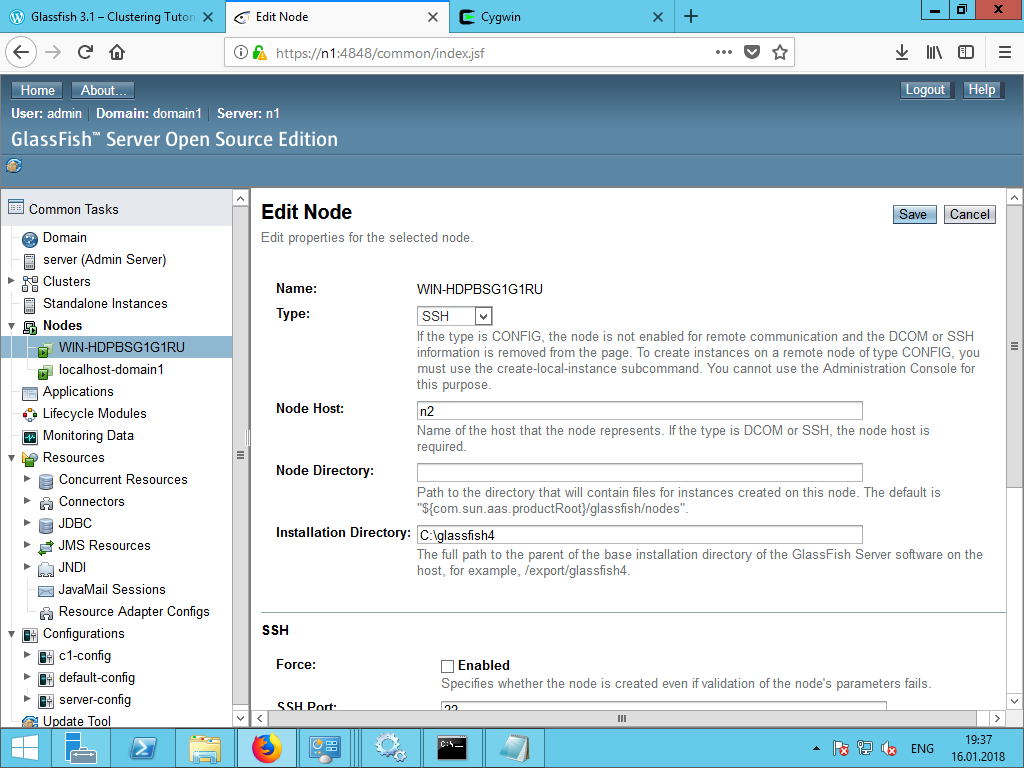


Рис. 2.18 Налаштування другої ноди

Тут же пролистуємо донизу, у полі «SSH User Name» вписуємо імя користувача для підключення до другого серверу по SSH (“sftpuser01”). Нижче змінюємо параметр «SSH User Authentication» на «Password». Вводимо пароль, що ми встановили на користувача (“sftpuser01”). Натискаємо кнопку «Save».

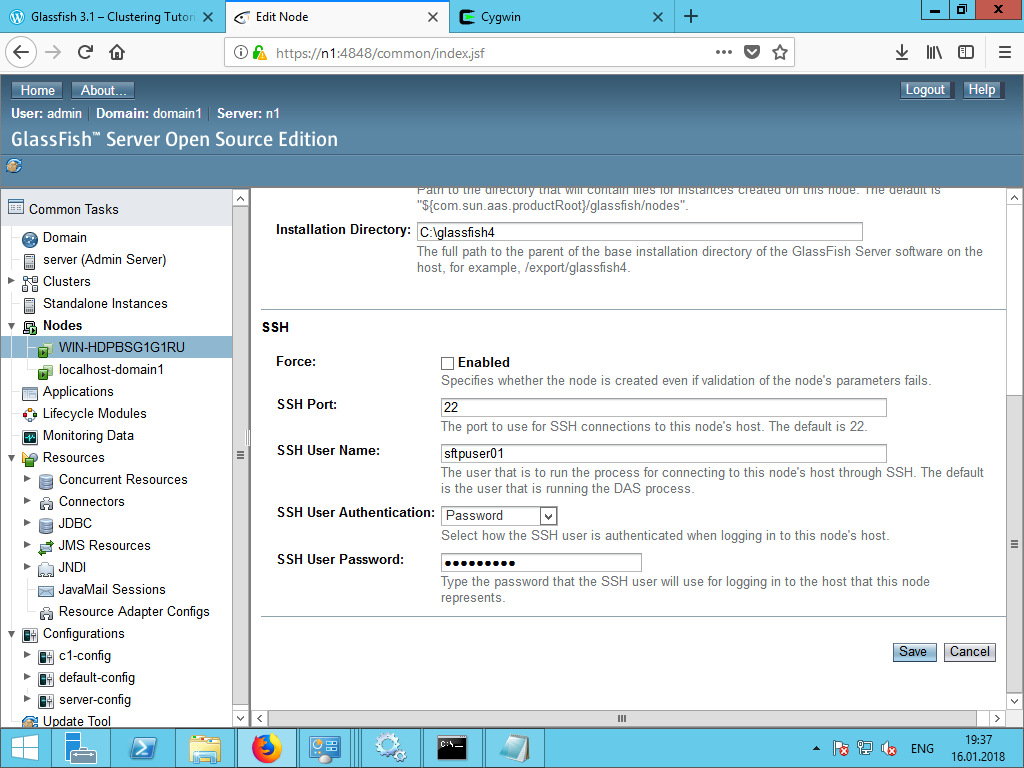


Рис. 2.19 Налаштування другої ноди

Якщо все було виконано правильно, висвітлиться напис «New values successfully saved».

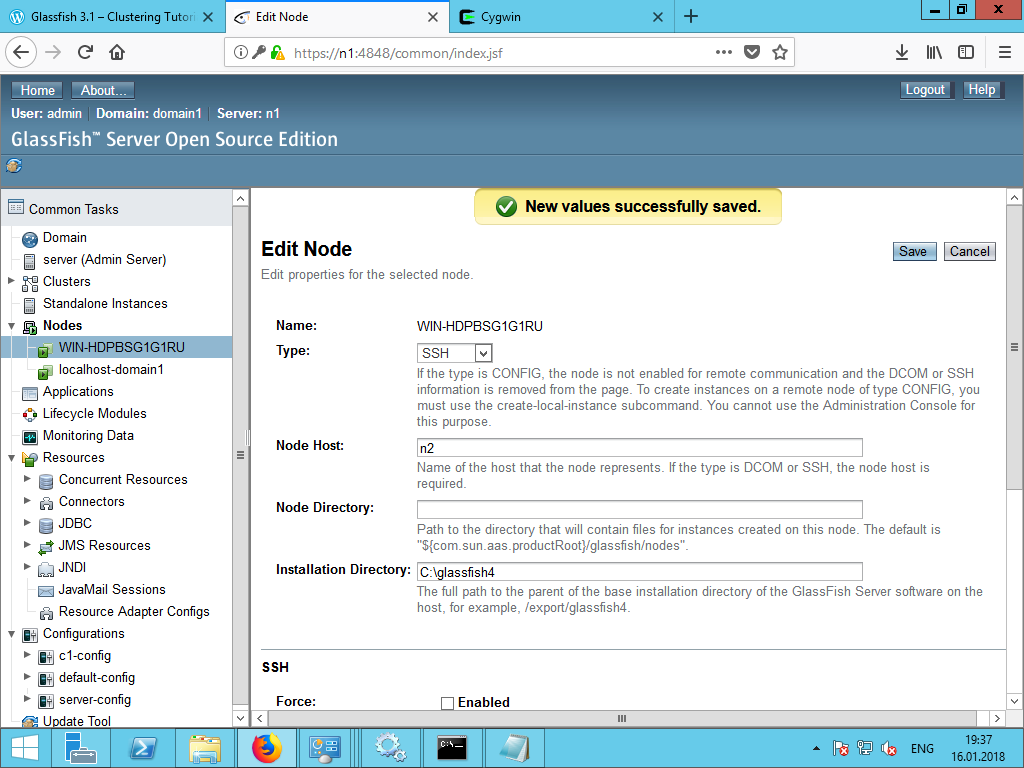


Рис. 2.20 Налаштування другої ноди

Переходимо у вкладку «Clusters», там вибираємо наш кластер (“c1”). Після чого переходимо у вкладку «Instances», де побачимо два з’єднання, одне із яких буде вимкнено.

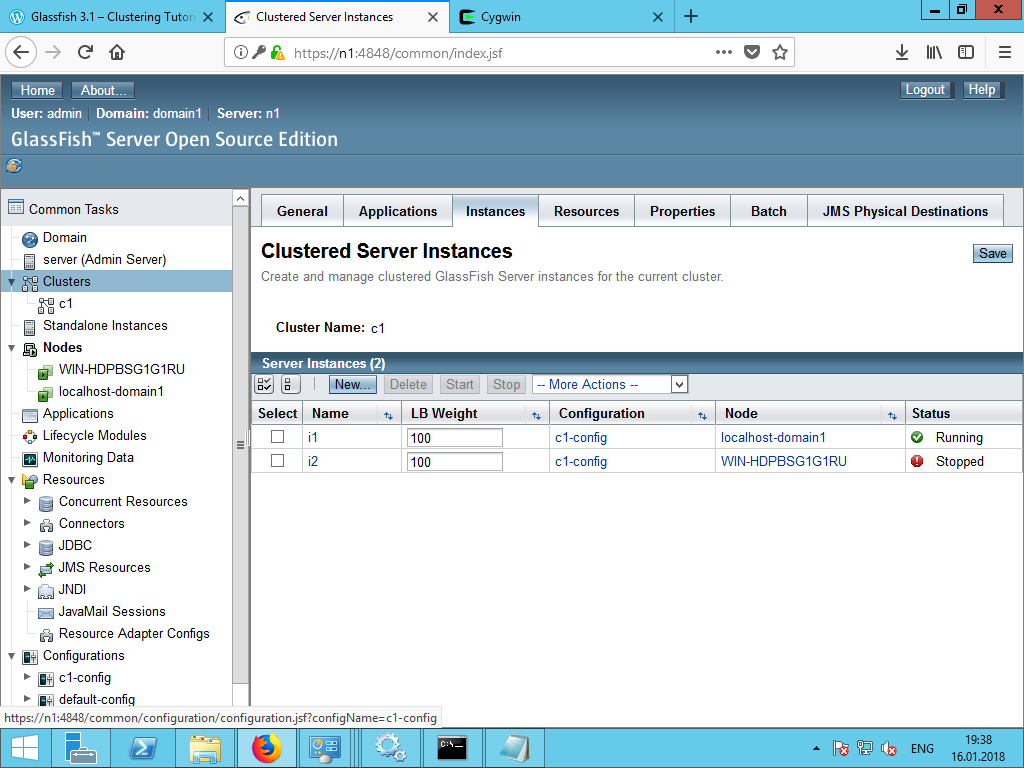


Рис. 2.21 Панель управління нодами у кластері

Вибираємо наше друге з’єднання, поставивши галочку біля його назви (“i2”). Натискаємо кнопку «Start». Чекаємо, поки з’єднання встановиться.

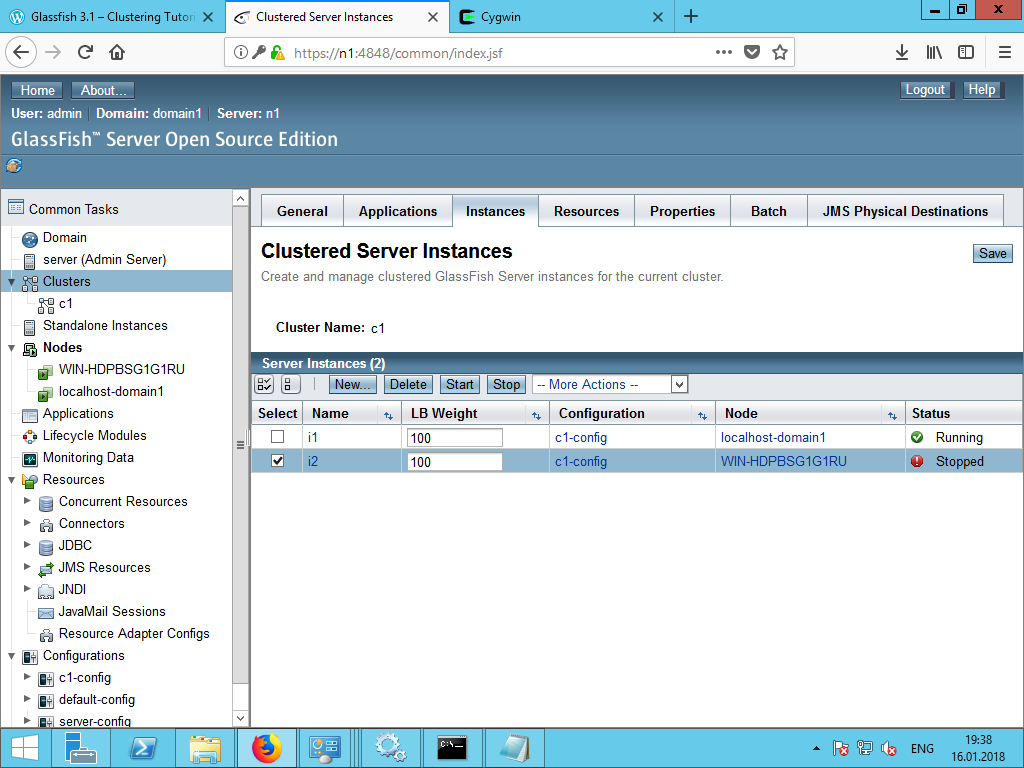


Рис. 2.22 Панель управління нодами у кластері

Якщо все було виконано правильно, з’єднання буде встановлено.

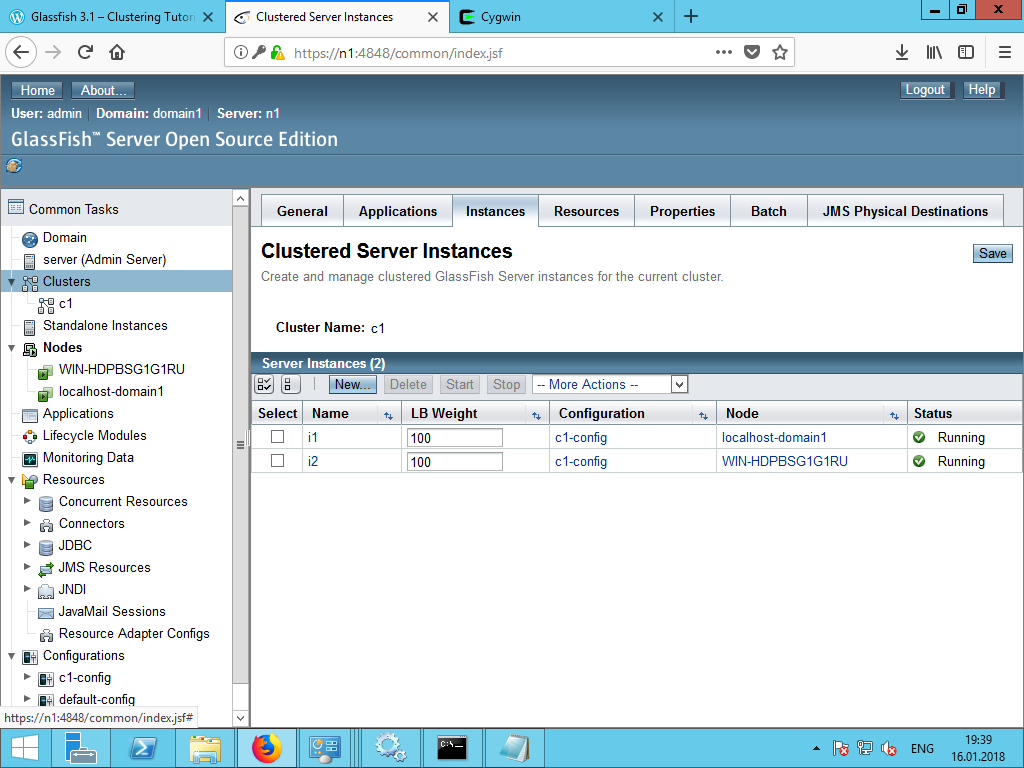


Рис. 2.23 Панель управління нодами у кластері

Завдяки створенню кластерної інфраструктури було вирішено питання, що стосувалося масштабування системи та її підтримки. У Glassfish Open Source Server 4.0 підтримка кластеру дуже проста, усе можна зробити через GUI у браузері. Тож, серед плюсів даної інфраструктури можна виділити:

* Штучне/ручне розподілення навантаження;
* Спрощене масштабування системи та підтримка;

Але, все ще є два дуже значних недоліки:

* При виході одного серверу із строю – часть користувачів залишається без доступу;
* Різна база даних у серверів;

### 2.3.4 Кластерна архітектура з використанням БН

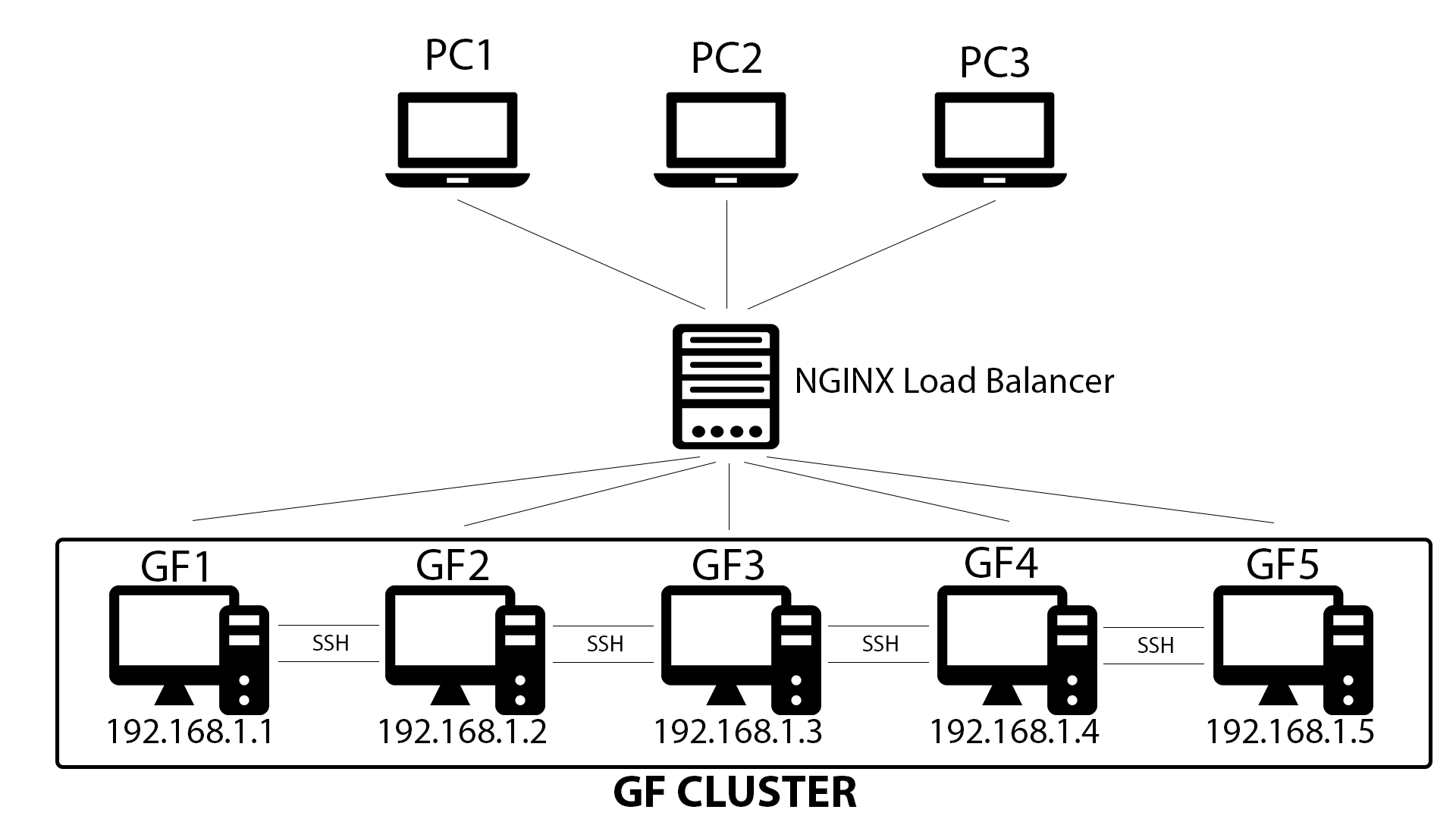


Рис. 2.24 Кластерна інфраструктура зі застосуванням балансувальника навантаження

Балансування навантаження стосується ефективного розподілу вхідного мережевого трафіку через групу резервних серверів, також відомих як ферма серверів або пул серверів.

Сучасні веб-сайти з високим трафіком повинні обслуговувати сотні тисяч (якщо не мільйони) одночасних запитів користувачів або клієнтів і швидко та надійно повертати правильні текстові, зображення, відео чи дані програми. Для економічного масштабування для задоволення цих великих обсягів сучасна найкраща обчислювальна практика зазвичай вимагає додавання більше серверів.

Балансир навантаження виступає в ролі "дорожнього поліцейського", що сидить перед вашими серверами і маршрутизує запити клієнтів на всіх серверах, здатних виконати ці запити таким чином, щоб максимально використовувати швидкість і потужність і гарантувати, що жоден сервер не буде перевантажений, що може погіршити продуктивність . Якщо один сервер знижується, балансир навантажень перенаправляє трафік на решта онлайн-серверів. Коли в групу серверів додається новий сервер, балансир завантаження автоматично починає надсилати на нього запити.

Таким чином, балансир навантаження виконує такі функції:

* Ефективно розподіляє запити клієнтів або завантаження мережі на декілька серверів;
* Забезпечує високу доступність та надійність, надсилаючи запити лише до серверів, які перебувають в мережі;
* Забезпечує гнучкість додавання або віднімання серверів, як диктує попит;

**Порівняння Apache і NGINX**

Apache і Nginx - два найпопулярніших веб-сервера з відкритим вихідним кодом в світі. Разом вони несуть відповідальність за обслуговування більше 50% інтернет-трафіку. Обидва рішення можуть обробляти різні робочі навантаження і працювати з іншим програмним забезпеченням для забезпечення повного веб-стека [19].

Хоча Apache і Nginx володіють багатьма якостями, їх не слід вважати повністю взаємозамінними. Кожен перемагає по-своєму, і важливо розуміти ситуації, коли вам може знадобитися переоцінити ваш веб-сервер, щоб вибрати з чого. У цій статті ми обговоримо, як кожен сервер зберігається в різних областях.

Перш ніж заглиблюватися в відмінності між Apache і Nginx, давайте коротко розглянемо передумови цих двох проектів і їх загальні характеристики [20].

Apache. HTTP-сервер Apache був створений Робертом МакКуалом в 1995 році і розроблявся Apache Software Foundation з 2000-х років. Оскільки веб-сервер HTTP є оригінальним базовим проектом і на сьогоднішній день є їх найбільш популярним програмним забезпеченням, його часто називають просто «Apache». Apache Web Server є найпопулярнішим сервером в Інтернеті з 1996 року. Завдяки цій популярності Apache має велику документацією і інтегрованою підтримкою інших програмних додатків. Адміністратори часто вибирають Apache за його гнучкість, міць і широку підтримку. Він поширюється через динамічно завантажується модульну систему і може обробляти велику кількість різних мов без підключення до окремого програмного забезпечення.

NGINX. У 2002 році Ігор Сисоєв почав працювати над Nginx у відповідь на проблему C10K, яка була завданням для веб-серверів, щоб почати обробку десяти тисяч одночасних з'єднань, як вимога сучасної мережі. Початковий публічний випуск був зроблений в 2004 році, і ця мета була досягнута завдяки використанню асинхронної архітектури, керованої подіями. Nginx набув популярності з моменту свого випуску завдяки низькому завантаженні ресурсів і здатності легко масштабуватися на мінімальному апаратному рівні. Nginx відмінно підходить для швидкої доставки статичного контенту і призначений для передачі динамічних запитів іншого програмного забезпечення, яке краще підходить для цих цілей. Nginx часто вибирається адміністраторами для ефективності ресурсів і чуйності. Прихильники вітають увагу Nginx до основних веб-серверів і функцій проксі.

Архітектура обробки з'єднань. Одне велике розходження між Apache і Nginx полягає в тому, що вони обробляють з'єднання і трафік. Це забезпечує, мабуть, найсуттєвіше відмінність в тому, як вони реагують на різні умови водіння.

Apache. Apache пропонує безліч багатопроцесорних модулів (Apache називає їх MPM), які визначають, як клієнти обробляють запити. По суті, це дозволяє адміністраторам легко змінювати архітектуру обробки з'єднань. Це:

* mpm\_prefork: Цей модуль обробки породжує процеси однією ниткою для обробки запиту. Кожна дитина може одночасно працювати з одним з'єднанням. Поки кількість запитів менша за кількість процесів, цей MPM дуже швидкий. Однак продуктивність знижується швидко після того, як запити перевищують кількість процесів, тому це не є хорошим вибором у багатьох сценаріях. Кожен процес має значний вплив на споживання оперативної пам’яті, тому цей MPM складно ефективно масштабувати. Це все ж може бути хорошим вибором, хоча він використовується разом із іншими компонентами, які не побудовані з урахуванням ниток. Наприклад, PHP не є безпечним для потоків, тому цей MPM рекомендується як єдиний безпечний спосіб роботи з mod\_php, модулем Apache для обробки цих файлів.
* mpm\_worker: Цей модуль породжує процеси, які можуть управляти кількома потоками. Кожна з цих потоків може обробляти єдине з'єднання. Нитки набагато ефективніші, ніж процеси, а це означає, що цей MPM масштабується краще, ніж MPM передформою. Оскільки потоків більше, ніж процесів, це також означає, що нові з'єднання можуть негайно зайняти вільну нитку, замість того, щоб чекати вільного процесу.
* mpm\_event: Цей модуль схожий на робочий модуль у більшості ситуацій, але оптимізований для обробки постійних з'єднань. Під час використання робочого MPM з'єднання буде утримувати потік незалежно від того, чи активно запит робиться до тих пір, поки з'єднання буде збережене. MPM подій MPM підтримує живі з'єднання, відкладаючи виділені спеціальні потоки для обробки збереження живих з'єднань та передачі активних запитів іншим потокам. Це утримує модуль від задирання неперервними запитами, що дозволяє швидше виконувати. Це було відзначено стабільним із випуском Apache 2.4.

Як бачите, Apache пропонує гнучку архітектуру для вибору різних алгоритмів підключення і запитів. Безпечний вибір - це, в основному, функція еволюції сервера і зростаюча потреба в одночасності при зміні ландшафту Інтернету.

NGINX. Nginx вийшов на сцену після Apache, з великим розумінням проблем одночасності, з якими стикаються сайти в масштабі. Використовуючи ці знання, Nginx з самого початку розроблявся для використання асинхронного неблокуючого алгоритму обробки з'єднань, керованого подіями. Nginx генерує робочі процеси, кожен з яких може обробляти тисячі з'єднань. Робочі процеси досягають цього, впроваджуючи механізм швидкого циклу, який постійно перевіряє і обробляє події. Відключення фактичної роботи від з'єднань дозволяє кожному співробітникові турбуватися про з'єднання тільки при запуску нового події. Кожне із з'єднань, оброблених співробітником, поміщається у цикл подій, де вони існують з іншими сполуками. Події обробляються асинхронно в циклі, що дозволяє вам керувати своєю роботою без блокування. Коли з'єднання розривається, воно видаляється з петлі. Цей стиль обробки з'єднань дозволяє Nginx неймовірно масштабуватися при обмежених ресурсах. Оскільки сервер є однопоточні і процеси не генеруються для опрацювання кожного нового з'єднання, використання пам'яті і ЦП, як правило, залишається відносно узгодженим навіть в періоди високого навантаження.

Статичний і динамічний контент. Що стосується реальних випадків використання, одне з найпоширеніших порівнянь між Apache і Nginx - це те, як кожен сервер обробляє статичні і динамічні запити контенту.

Apache. Сервери Apache можуть обробляти статичний контент, використовуючи звичайні файлові методи. Виконання цих операцій в основному є функцією методів MPM, описаних вище. Apache також може обробляти динамічний контент, вставляючи мовної процесор в кожний з його робочих примірників. Це дозволяє йому виконувати динамічний контент на самому веб-сервері, не покладаючись на зовнішні компоненти. Ви можете включити ці динамічні процесори з динамічними модулями. Можливість Apache для внутрішньої обробки динамічного контенту означає, що конфігурація динамічної обробки повинна бути простіше. Зв'язок не обов'язково повинна бути узгоджена з додатковим програмним забезпеченням, і модулі можна легко замінити, змінивши вимоги до контенту.

NGINX. Nginx не може самостійно обробляти динамічний контент. Щоб обробити PHP і інші запити на динамічний контент, Nginx повинен перейти на зовнішній процесор для виконання і дочекатися повернення вмісту. Результати можуть бути передані клієнтові. Для адміністраторів це означає, що зв'язок повинна бути встановлена ​​між Nginx і процесором через один з протоколів, які Nginx може говорити (http, FastCGI, SCGI, uWSGI, memcache). Це може ускладнити завдання, особливо при спробі передбачити кількість дозволених з'єднань, оскільки для кожного виклику процесора буде використовуватися додаткове з'єднання. Однак цей метод має деякі переваги. Оскільки динамічний інтерпретатор не вбудований в робочий процес, його накладні витрати будуть доступні тільки для динамічного вмісту. Статичний контент може бути представлений по прямій лінії і буде пов'язаний тільки з перекладачем в міру необхідності. Apache також може працювати таким чином, але це виключає переваги попереднього розділу.

Розподілена та централізована конфігурації. Для адміністраторів одне з найбільш очевидних відмінностей між двома програмами полягає в тому, чи дозволено конфігурація на рівні каталогів в каталогах вмісту.

Apache. Apache включає можливість вирішувати додаткову конфігурацію на основі каталогів, перевіряючи і інтерпретуючи директиви в прихованих файлах в самих каталогах контенту. Ці файли відомі як файли .htaccess. Оскільки ці файли знаходяться всередині самих каталогів вмісту, Apache перевіряє кожен компонент шляху до запрошенням файлу на наявність файлу .htaccess і застосовує знайдені в ньому директиви. Це ефективно враховує децентралізовану конфігурацію веб-сервера, яка часто використовується для реалізації перезапису URL-адрес, обмежень доступу, авторизації та аутентифікації, навіть кешування політик. Хоча всі наведені вище приклади можна налаштувати в основному файлі конфігурації Apache, файли .htaccess мають ряд важливих переваг. По-перше, оскільки вони інтерпретуються кожен раз, коли знаходять шлях запиту, вони реалізуються негайно, без перезапуску сервера. По-друге, він дозволяє неавторизованих користувачам управляти деякими аспектами їх власного веб-контенту, котрі дають їм контроль над усім файлом конфігурації. Це дає можливість певного веб-програмного забезпечення, наприклад системам управління контентом, легко налаштовувати свою середу, котрі дають доступ до файлу центральної конфігурації. Це також використовується провайдерами загального хостингу для підтримки контролю над основною конфігурацією, надаючи клієнтам контроль над їх конкретними каталогами.

NGINX. Nginx не бере до уваги файли .htaccess і не надає механізм оцінки конфігурації для каталогів поза основним файлу конфігурації. Він може бути менш гнучким, ніж модель Apache, але у нього є свої переваги. Найбільш помітне поліпшення в порівнянні з конфігурацією на рівні каталогів .htaccess - підвищення продуктивності. Для типової установки Apache, яка може дозволити .htaccess в будь-якому каталозі, сервер перевірить наявність цих файлів в кожному з батьківських каталогів, що приводить до необхідного файлу для кожного запиту. Якщо під час пошуку виявлений один або кілька файлів .htaccess, вони повинні бути прочитані і інтерпретовані. Запобігаючи скасування каталогів, Nginx може швидше обслуговувати запити, виконуючи пошук по одному каталогу і зчитуючи файл для кожного запиту (за умови, що файл знаходиться в звичайній структурі каталогів). Ще однією перевагою є безпека. Спільне використання доступу до конфігурації на рівні каталогів також розширює обов'язки щодо забезпечення безпеки для окремих користувачів, які можуть виявитися не в змозі впоратися з цим завданням. Забезпечення контролю адміністратором над всім веб-сервером може запобігти деякі помилки безпеки, які можуть виникнути при доступі до інших сторін. Майте на увазі, що можна відключити інтерпретацію .htaccess в Apache, якщо ці проблеми резонують з вами.

Інтерпретація на основі файлів і URI. Оскільки веб-сервер інтерпретує запити і зіставляє їх з реальними ресурсами в системі, це ще одна область, де два сервера розрізняються.

Apache. Apache надає можливість інтерпретувати запит як фізичний ресурс в файлової системі або як місце розташування URI, що може зажадати більш абстрактної оцінки. Зазвичай колишній Apache використовує блоки <Directory> або <Files>, в той час як він використовує блоки <Location> для отримання більш абстрактних ресурсів. Оскільки Apache спочатку розроблявся як веб-сервер, запити за замовчуванням зазвичай розглядаються як ресурси файлової системи. Він починається з отримання кореня документа і додавання частини запиту до номера хоста і порту, щоб спробувати знайти фактичний файл. По суті, ієрархія файлової системи представлена ​​в Інтернеті у вигляді дерева доступних документів. Apache надає ряд альтернатив, коли запит не відповідає базовій файлової системи. Наприклад, директива Alias ​​може використовуватися для відображення альтернативного місця розташування. Використання блоків <Location> - це метод роботи з самим URI замість файлової системи. Існують також регулярні вирази, які можна використовувати, щоб зробити настройку більш гнучкою у всій файлової системи. Хоча Apache здатний працювати як з базової файлової системою, так і з мережею, він сильно схильний до методів файлової системи. Це можна побачити в деяких дизайнерських рішеннях, включаючи використання файлів .htaccess для настройки каталогів. Самі документи Apache застерігають від використання блоків на основі URI для обмеження доступу, коли запит відображає базову файлову систему.

NGINX. Nginx був створений як веб-сервер, так і проксі-сервер. Завдяки архітектурі, необхідній для цих двох ролей, вона працює в першу чергу з URI, перекладаючи у файльну систему, коли це необхідно. Це можна побачити деякими способами побудови та інтерпретації файлів конфігурації Nginx. Nginx не забезпечує механізм визначення конфігурації для каталогу файлової системи, а замість цього аналізує сам URI. Наприклад, основними блоками конфігурації для Nginx є серверні та локаційні блоки. Блок сервера інтерпретує хост, який запитується, в той час як блоки розташування відповідають за відповідність частин URI, що надходять після хоста і порту. У цей момент запит інтерпретується як URI, а не як розташування у файловій системі. Для статичних файлів усі запити в кінцевому підсумку повинні бути віднесені до місця у файловій системі. Спочатку Nginx вибирає сервер і блоки розташування, які будуть обробляти запит, а потім поєднує корінь документа з URI, адаптуючи все необхідне відповідно до вказаної конфігурації. Це може здатися схожим, але аналіз запитів в основному як URI замість файлової системи дозволяє Nginx легше функціонувати як у ролі веб, пошти та проксі-сервера. Nginx налаштовується просто шляхом викладення способів реагування на різні схеми запитів. Nginx не перевіряє файлову систему, поки вона не готова подати запит, що пояснює, чому вона не реалізує форму файлів .htaccess.

Модулі. І Nginx, і Apache можуть бути розширені за рахунок модульних систем, але їх робота істотно відрізняється.

Apache. Модульна система Apache дозволяє динамічно завантажувати або вивантажувати модулі відповідно до потреб вашого сервера. Ядро Apache присутній завжди, а модулі можна включати або вимикати, додаючи або видаляючи додаткові функції і підключаючись до головного сервера. Apache використовує цю функціональність для різних завдань. Через зрілості платформи доступна велика бібліотека модулів. Їх можна використовувати для зміни деяких основних функцій сервера, таких як mod\_php, який вбудовує інтерпретатор PHP в кожного працівника. Однак модулі не обмежуються динамічної обробкою контенту. Інші функції включають в себе перезапис URL, аутентифікацію клієнта, посилення захисту сервера, ведення журналу, кешування, стиснення, проксі, обмеження швидкості і шифрування. Динамічні модулі можуть значно розширити базову функціональність без особливої ​​додаткової роботи.

NGINX. Nginx також реалізує модульну систему, але вона сильно відрізняється від Apache. Модулі Nginx не завантажуються динамічно, тому їх необхідно вибрати і скомпілювати в основне програмне забезпечення. Для багатьох користувачів це зробить Nginx набагато менш гнучким. Це особливо актуально для користувачів, яким незручно підтримувати власне скомпільований програмне забезпечення поза звичної системи упаковки. Хоча в дистрибутивні пакети зазвичай входять найбільш часто використовувані модулі, якщо вам потрібен нестандартний модуль, вам потрібно буде зібрати сервер самостійно з вихідного коду. Модулі Nginx як і раніше дуже корисні, і вони дозволяють вам диктувати, що ви хочете від вашого сервера, тільки включаючи функціональність, яку ви маєте намір використовувати. Деякі користувачі також можуть порахувати його більш безпечним, оскільки довільні компоненти не можуть бути підключені до сервера. Однак, якщо ви коли-небудь ставите свій сервер в положення, де це можливо, цілком ймовірно, що він вже скомпрометований. Модулі Nginx дозволяють використовувати багато з тих же можливостей, що і модулі Apache. Наприклад, модулі Nginx можуть забезпечувати підтримку, стиск, обмеження швидкості, ведення журналу, переписування, геолокацію, аутентифікацію, шифрування, потокову передачу і поштову розсилку.

Підтримка, сумісність, екосистема і документація. Головне, що потрібно мати на увазі, це те, що реальний процес підйому і запуску забезпечить доступну допомогу і підтримку серед іншого програмного забезпечення.

Apache. Оскільки Apache був популярним протягом тривалого часу, підтримка серверів досить поширена. Для основного сервера є велика бібліотека сторонньої і сторонньої документації і сценаріїв, заснованих на завданнях, пов'язаних з підключенням Apache до іншого програмному забезпеченню. Поряд з документацією багато інструментів і веб-проекти містять інструменти для завантаження себе в середовищі Apache. Це може бути включено в самі проекти або в пакети, підтримувані вашою командою по упаковці. В цілому, Apache отримає більше підтримки від сторонніх проектів просто через своєї частки ринку і тривалості часу, протягом якого він був доступний. Адміністратори також більш схильні мати досвід роботи з Apache не тільки через його поширеності, але й тому, багато людей починають з сценаріїв спільного хостингу, які майже повністю залежать від Apache через можливості розподіленого управління .htaccess.

NGINX. Nginx все більше підтримується, так як все більше і більше користувачів розглядають його за профілем продуктивності, але в деякі ключові областях його ще можна надолужити. Раніше було складно знайти вичерпну англомовну документацію для Nginx, тому що велика частина ранньої розробки і документації була російською мовою. По мірі зростання інтересу до проекту, документація була заповнена, і тепер на сайті Nginx і через сторонні організації є безліч адміністративних ресурсів. Що стосується сторонніх додатків, підтримка і документація стають все більш доступними, і в деяких випадках пакети оновлень вибирають між автоматичним настроюванням для Apache і Nginx. Навіть без підтримки конфігурація Nginx для запуску альтернативного програмного забезпечення зазвичай проста, якщо сам проект документує свої вимоги (дозволу, заголовки і т. д.).

Спільне використання Apache і Nginx. Розглядаючи переваги і обмеження як Apache, так і Nginx, ви можете краще зрозуміти, який сервер більше відповідає вашим потребам. Проте багато користувачів вважають, що ви можете використовувати потужність кожного сервера, використовуючи їх разом. Умовної конфігурацією цього партнерства є розміщення Nginx перед Apache в якості зворотного проксі-сервера. Це дозволить Nginx обробляти всі запити клієнтів. Цей параметр добре працює для багатьох людей, тому що він дозволяє Nginx функціонувати як сортувальна машина. Він опрацює всі запити, які він може, і перенаправляє ті запити, які він не в змозі обслуговувати. Скорочуючи запити, які запитує сервер Apache, ми можемо зменшити деякі блокування, що виникають, коли процес або потік Apache зайнятий. Ця конфігурація також дозволяє масштабувати, додаючи додаткові додаткові сервери. Nginx може бути налаштований для простого переходу до пулу серверів, що підвищує стійкість цієї конфігурації до збоїв і продуктивності.

Підсумкове порівняння Apache та NGINX Таблиця 2.4

|  |  |
| --- | --- |
| **Apache** | **NGINX** |
| Призначений бути веб-сервером | Призначений бути як веб-сервер, так і зворотний проксі-сервер |
| Не може обробляти кілька запитів одночасно з великим веб-трафіком | Може обробляти кілька запитів клієнтів одночасно з обмеженими ресурсами |
| Має багатопотоковий підхід до обробки запитів клієнтів | Має подієвий підхід для обслуговування запитів клієнтів |
| Модулі динамічно завантажуються або вивантажуються, що робить його більш гнучким | Модулі неможливо завантажувати динамічно. Повинно бути скомпільовано в рамках основного програмного забезпечення |
| Обробляє динамічний вміст у самому веб-сервері | Неможливо обробити динамічний вміст в рідних умовах |

Отже, виходячи з порівняння NGINX та Apache мій вибір пав на NGINX. Ми будемо використовувати звичайну версію NGINX оскільки вона є безкоштовною та покриває усі необхідні нам функції. У версії NGINX Plus є більш широкий функціонал, який можливо ми б і могли застосувати, але нам достатньо буде і звичайної версії. Що стосується який тип балансування потрібно використати, то усе залежить від умов та цілей. У нашому випадку, підтримка сессій чи довгих підключень до нод кластеру непотрібно, а отже ми будемо використовувати метод Round Robin.

Перш за все нам необхідно встановити сам балансувальник навантаження. Усе виконувати ми будемо на базі Ubuntu Server 18.04. Встановити NGINX ми можемо за допомогою команди:

sudo apt-get install nginx

На питання, чи хочемо ми продовжити, пишемо «У». Наступним кроком прописуємо таку команду:

nano /etc/nginx/sites-available/default

Це є файлом конфігурації NGINX. Ми будемо використовувати такі налаштування:

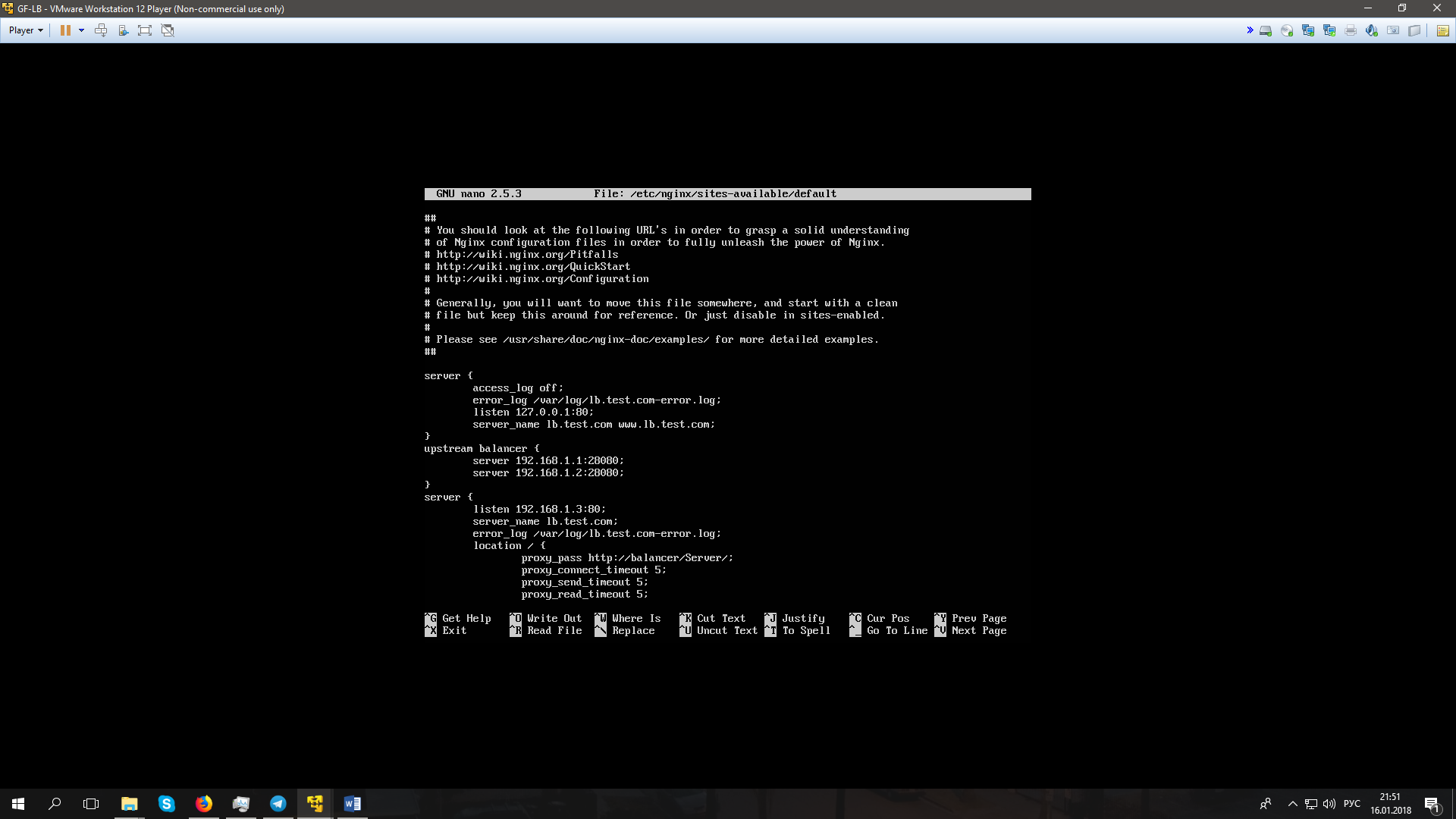


Рис. 2.25 Конфігурація балансувальника навантаження ч.1

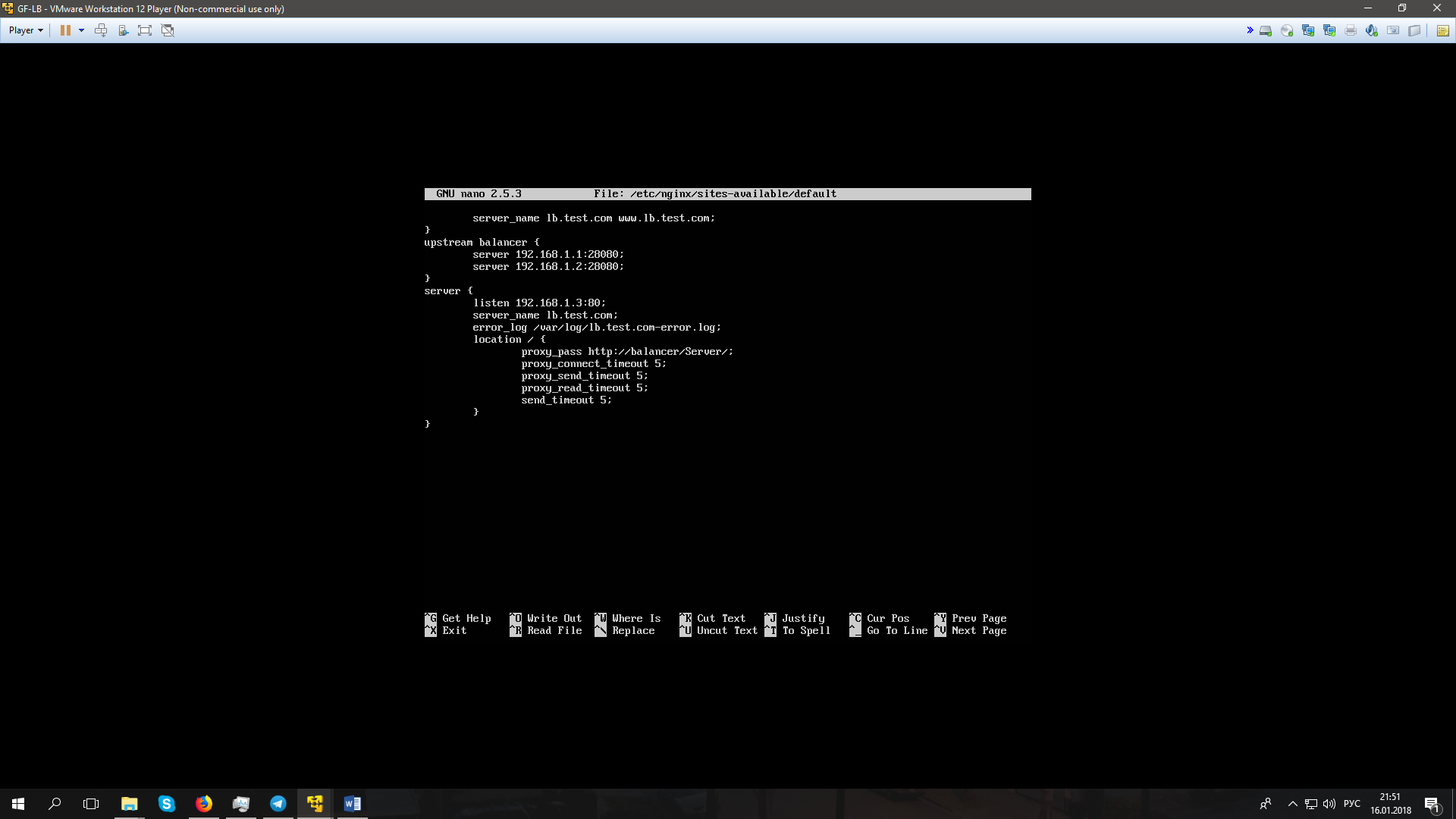


Рис. 2.26 Конфігурація балансувальника навантаження ч.2

Після налаштування та збереження усіх параметрів, перезавантажуємо службу командою sudo service nginx restart.

Перевіримо працездатність балансувальника навантаження. Для цього ми розгортаємо додаток у нашому Glassfish Open Source Server 4.0 та переходимо за адресою, яку ми виділили для нашого балансувальника. Бачимо такий результат:

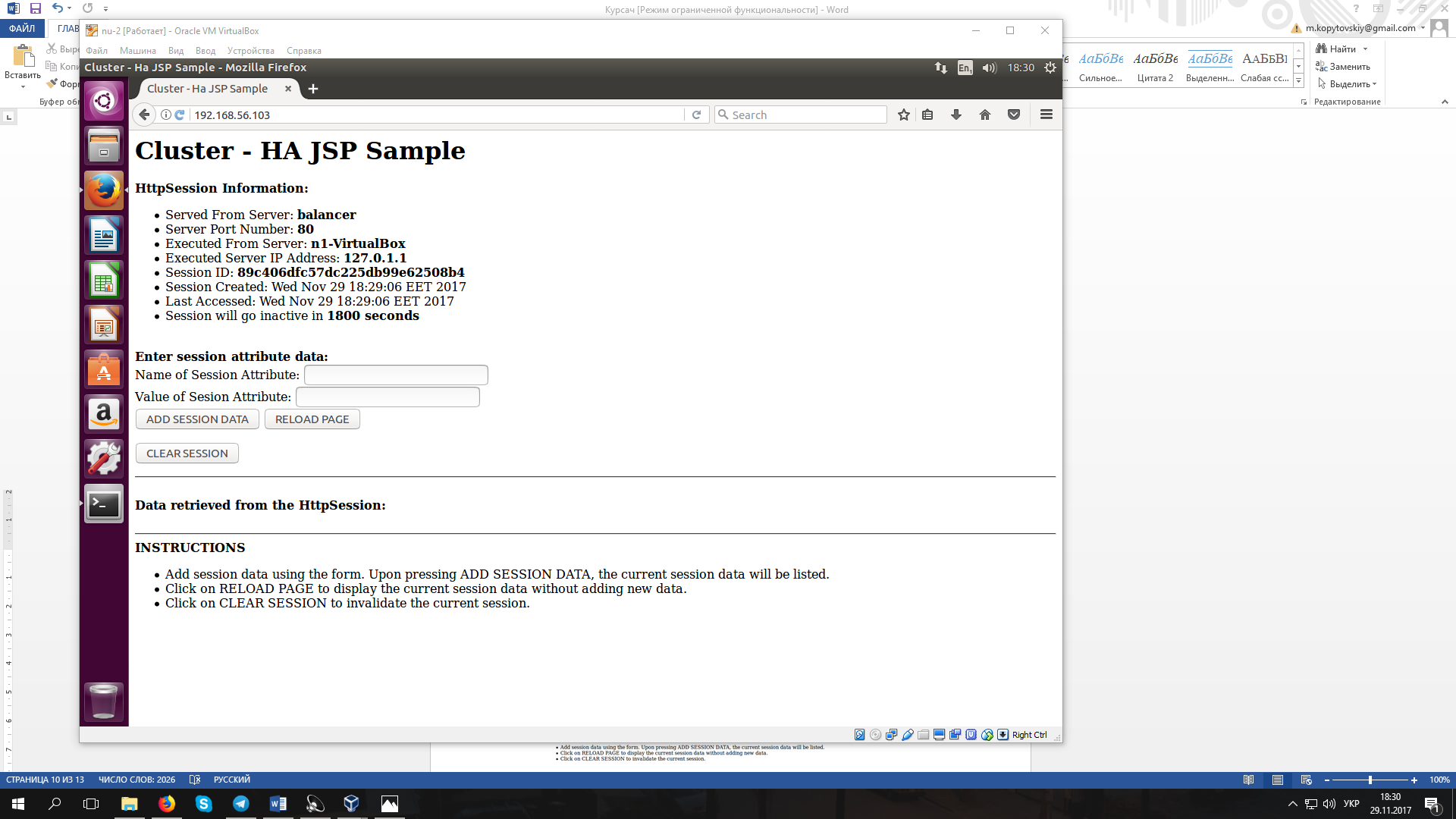


Рис. 2.27 Перенаправлення запиту на перший сервер

Результатом є те, що за адресою балансувальника навантаження, ми потрапили до першого вузла нашого сервера Glassfish Open Source Server 4.0. Оновимо сторінку у браузері, після чого дивимося на результат:

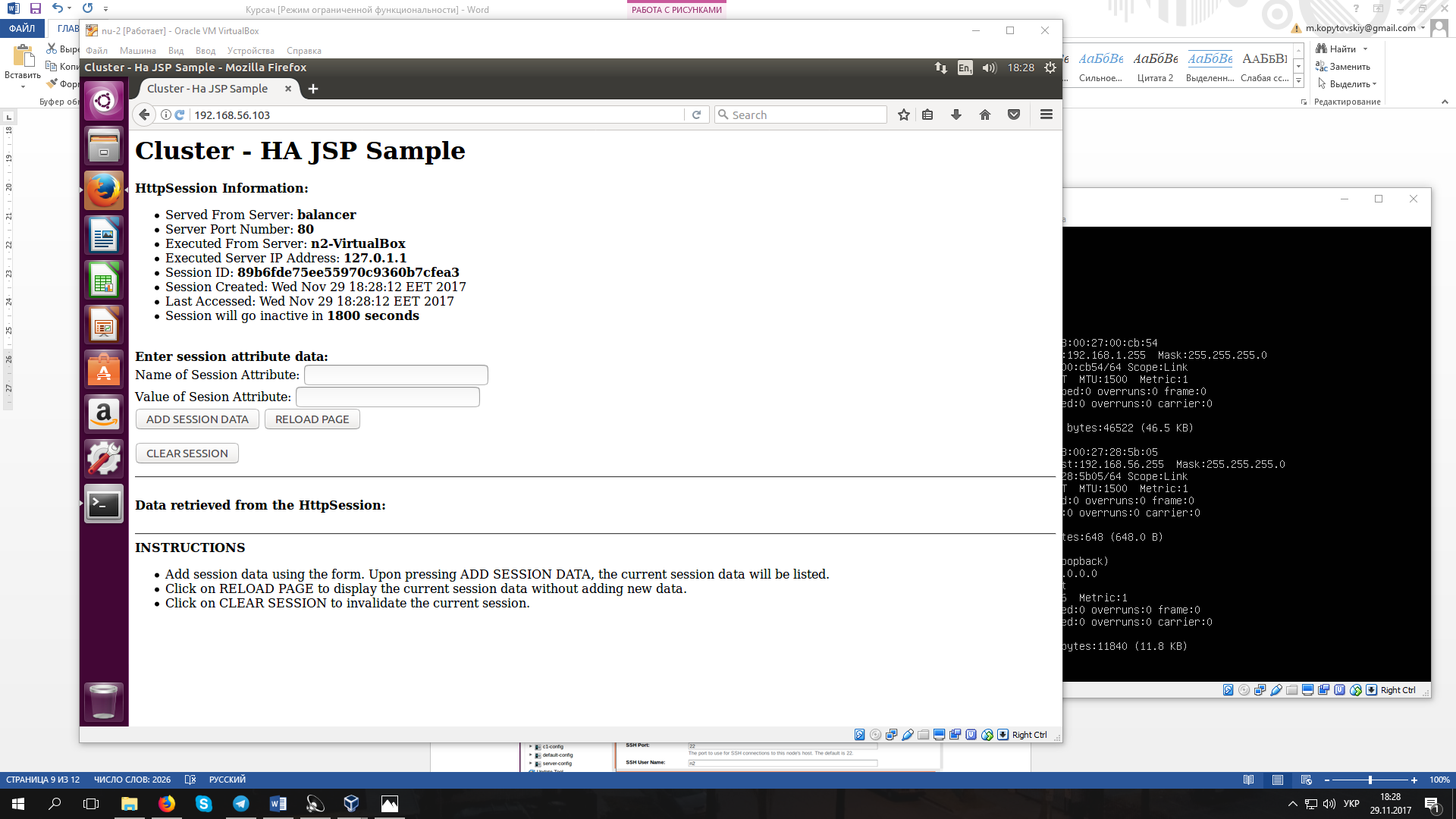


Рис. 2.28 Перенаправлення запиту на другий сервер

Так і є, на цей раз нас перенаправило до другої ноди нашого кластеру розгорнутого на Glassfish Open Source Server 4.0.

Що ж, одним застосуванням балансувальника навантаження ми вирішили майже усі наші проблеми, тож можна виділити такі переваги:

* Розумне розподілення навантаження (трафіку);
* Полегшене масштабування та підтримка системи;
* У разі виходу одного сервера зі строю – інший бере навантаження на себе;

Єдина проблема, яку ми не вирішили це:

* Різна база

### 2.3.5 Удосконалення кластерної архітектури на базі Glassfish

У ході цієї роботи ми майже знайшли найоптимальніше рішення для роботи з кластерною відмовостійкою системою з балансувальником навантаження. Щоб зробити нашу систему максимально позбавленною недоліків, було прийнято рішення розгорнути FTP сервер у нашій інфраструктурі та під’єднати кожну ноду з кластеру до неї, щоб усі файли, які зберігаються у Glassfish Open Source Server 4.0, зберігались на FTP сервері.

**Налаштування FTP серверу**

Оскільки ми будемо використовувати Ubuntu Server 18.04 як наш FTP сервер, то найбільш оптимальне рішення є використання VSFTPD [21].

VSFTPD - популярний тип FTP-сервера, є FTP-сервером за замовчуванням для багатьох операційних систем і доступний в стандартному репозиторії Ubuntu. Щоб встановити VSFTPD-сервер, використовуйте команду: apt-get install vsftpd. Після завершення процесу можна вважати, що ваш FTP-сервер встановлений.

Після установки необхідно налаштувати FTP-сервер під конкретні завдання за допомогою файлу, який знаходиться ось тут: /etc/vsftpd.conf. Змінимо деякі настройки наступним чином:

listen = YES

listen\_ipv6 = NO

anonymous\_enable = NO

local\_enable = YES

write\_enable = YES

chroot\_local\_user = YES

local\_umask = 022

force\_dot\_files = YES

Перш ніж запустити FTP-сервер, додайте рядок / usr / sbin / nologin в файл / etc / shell, щоб користувачі могли підключатися до FTP без надання їм shell-доступу: echo "/usr/sbin/nologin" >> /etc/shell

Запускаємо FTP сервер:

service vsftpd start

Внесимо зміни в файл /etc/hosts.allow, щоб заборонити доступ з усіх IP-адрес, крім тих, які планується використовувати для підключення до FTP-сервера:

vsftpd : 127.0.0.1 : allow

vsftpd : allowed\_IP\_adress : allow

vsftpd : ALL : deny

Отже, тепер ми можемо використовувати FTP сервер як сховища для нашого кластеру. Але, щоб підвисити безпеку нашого сховища, було прийнято рішення робити копії файлів з FTP серверу на хмарне сховище. Прочитавши багато літератури, було вирішено використати Dropbox для бекапу наших файлів. Є вже готове рішення, як зробити це автоматизовано. Рішення це називається Dropbox Uploader.

Dropbox Uploader - це набір скриптів BASH, який можна використовувати для завантаження, завантаження, видалення, онлайн-обміну файлами, синхронізації та резервного копіювання. Він написаний мовою BASH і потребує лише CURL [22].

Переваги Dropbox Uploader:

* Портативний: Він написаний в сценаріях BASH і потребує лише cURL (curl - це інструмент для передачі даних з сервера або на сервер, доступний для всіх операційних систем і встановлений за замовчуванням у багатьох дистрибутивах Linux).
* Безпечно: не потрібно вказувати своє ім’я користувача / пароль для цього сценарію, оскільки він використовує офіційний Dropbox API v2 для процесу автентифікації.

Особливості:

* Кросплатформенність
* Підтримка офіційного API Dropbox v2
* Пароль не потрібен
* Простий покроковий майстер налаштування
* Просте і чітке завантаження файлів
* Завантаження файлів та рекурсивного каталогу
* Розширення підстановки Shell (лише для завантаження)
* Видалити / перемістити / Перейменувати / Скопіювати / Список / Спільні файли
* Створіть посилання на спільний доступ
* Монітор змін

В першу чергу, що потрібно зробити – це встановити Dropbox Uploader. Зробити ми це можемо командою: curl "https://raw.githubusercontent.com/andreafabrizi/Dropbox-Uploader/master/dropbox\_uploader.sh" -o dropbox\_uploader.sh або командою git clone https://github.com/andreafabrizi/Dropbox-Uploader.git

Наступним кроком є налаштування дозвілу для цієї програми. Зробити можна завдяки команді $chmod +x dropbox\_uploader.sh. Після цього можемо запускати даний скрипт командою $./dropbox\_uploader.sh. Буде запит на введення логіну і паролю, після чого можемо приступати до запуску процесу резервного копіювання на хмарне сховище.

Щоб почати процес копіювання наших збережених файлів Glassfish Open Source Server 4.0 до хмарного сховища Dropbox, необхідно запустити скрипт наступною командою: ./dropbox\_uploader.sh upload /etc/gf\_cert /myfiles/gf\_cert.backup. Після цього запуститься процес копіювання файлів з директорії /etc/gf\_cert нашого FTP серверу до директорії /myfiles/gf\_cert.backup у хмарному сховищі Dropbox.

Тепер нам необхідно зробити так, щоб даних процес повторювався, наприклад, кожні пів години. Це ми можемо зробити, написавши невеличкий скрипт: /bin/bash -c "sleep 15 && while true; do dropbox\_uploader.sh upload /etc/gf\_cert /myfiles/gf\_cert.backup; sleep 1800; done"

Усе, зберігаємо скрипт і він буде виконуватись кожні 30 хвилин.

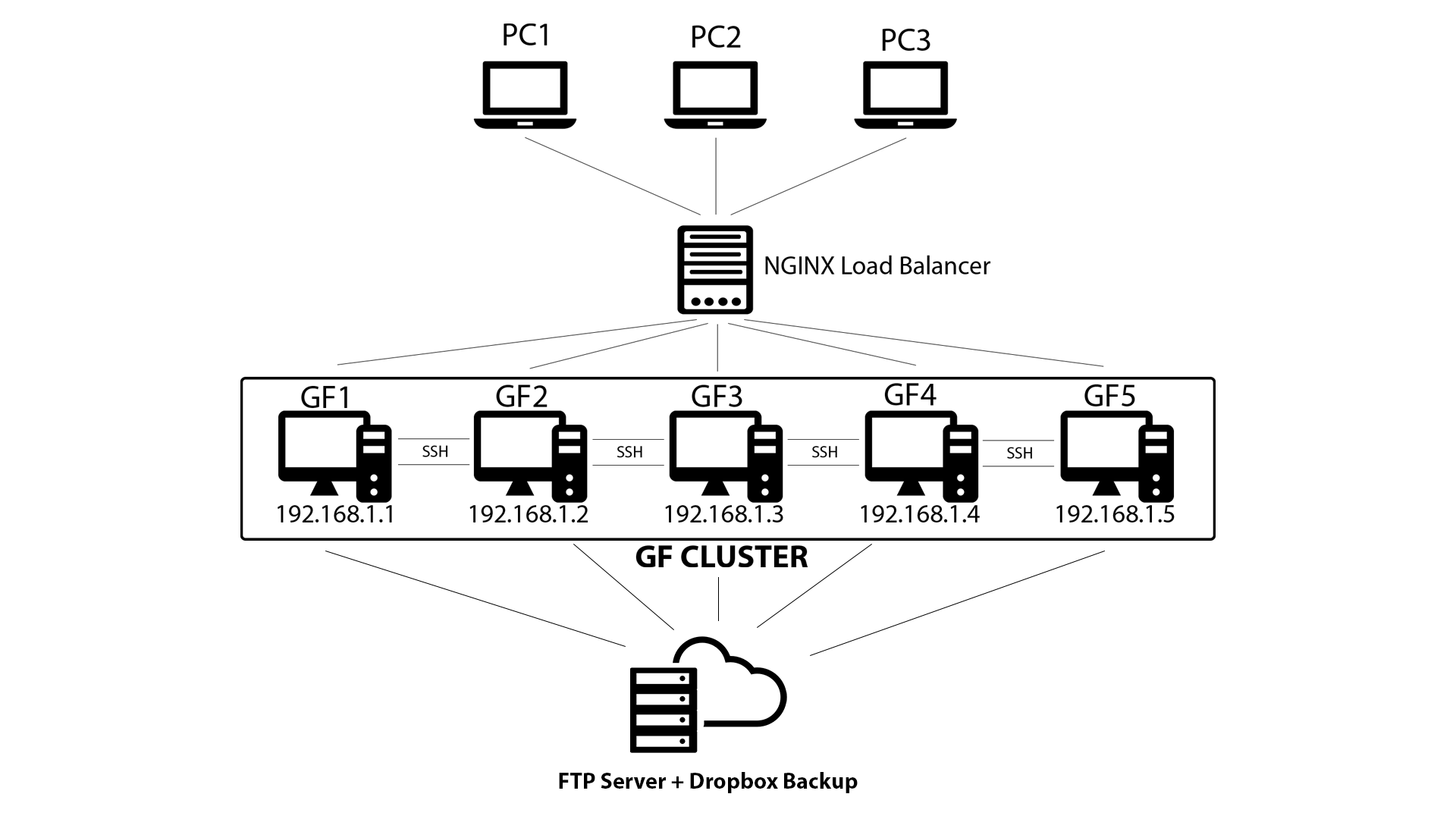


Рис. 2.29 Схема оновленої інфраструктури

**Розгортання кластеру у хмарному середовищі**

В ситуації, коли у серверного додатку багато користувачів і необхідне масштабування, встає питання – як зекономити, адже новий сервер коштує чимало грошей, а в додаток до цього є низка проблем, таких як постійне та безперебійне енергопостачання, заходи по безпеці і захисту конфіденційної інформації, тощо. Усі ці питання можуть вирішити хмарні середовища.

**Azure vs AWS**

Amazon Web Services (AWS) - це хмарна платформа від Amazon, що надає послуги в різних областях, таких як обчислення, зберігання, доставка та інші функції, які допомагають бізнесу рости і розвиватися. Ми можемо використовувати ці домени в якості сервісів, які можна використовувати для створення і розгортання додатків різних типів в хмарної платформі [23]. Ці сервіси призначені для роботи один з одним і забезпечують масштабовані та ефективні результати. Пропоновані сервіси AWS діляться на 3 типи: інфраструктура як послуга (IaaS), програмне забезпечення як послуга (SaaS) і платформа як послуга (PaaS). AWS був запущений в 2006 році і став кращою хмарної платформою серед доступних на даний момент хмарних платформ. Хмарні платформи пропонують безліч переваг, таких як управління накладними витратами, мінімізація витрат і багато іншого. Згодом AWS придбала ряд орієнтованих на технології організацій, які вони хотіли б об'єднати. Нещодавнє придбання AWS було зосереджена не на великих, усталених компаніях, а на організаціях, які могли б зміцнити і в цілому поліпшити пропозиції існуючих постачальників хмарних технологій. Ці придбання не додають AWS, але розширюють його основні послуги. Наприклад, AWS придбала TSO Logic, Sqrrl і CloudEndure. TSO Logic була хмарної компанією з міграції, яка надає аналітику, яка дозволяє клієнтам переглядати стан свого поточного центру обробки даних і моделювати хмарну міграцію. Sqrrl - це запуск безпеки, який збирає дані з таких точок, як шлюзи, сервери і маршрутизатори, і потім поміщає ці результати в панель безпеки. Cloud Endure - це компанія, яка займається міграцією навантаження на загальнодоступне хмара, аварійним відновленням і резервним копіюванням. Ці придбання не повинні істотно змінити AWS, а краще позиціонувати його. Наприклад, придбання CloudEndure має прискорити міграцію локальних робочих навантажень в хмару AWS.

Microsoft Azure - це хмарна платформа Microsoft, що надає послуги в різних областях, таких як обчислення, зберігання, бази даних, мережеві інструменти, інструменти для розробників і інші функції, які допомагають організаціям рости і розвивати свій бізнес. Служби Azure широко класифікуються як платформа як послуга (PaaS), програмне забезпечення як послуга (SaaS) і інфраструктура як послуга (IaaS), які можуть використовуватися розробниками і персоналом програмного забезпечення для створення, розгортання та управління службами і додатками. через хмару. Microsoft Azure був запущений в 2010 році і є одним з найбільших постачальників комерційних хмарних послуг. Він пропонує широкий спектр інтегрованих хмарних сервісів і функцій, таких як аналітика, обчислення, мережі, бази даних, сховище даних, мобільні і веб-додатки, які легко інтегруються з вашої середовищем для досягнення ефективності та масштабованості.

Ключові відмінності між Azure та AWS:

* Користувачі AWS EC2 можуть налаштувати власні VMS або попередньо налаштовані зображення, тоді як користувачам Azure потрібно вибрати віртуальний жорсткий диск, щоб створити VM, попередньо налаштований стороною, і потрібно вказати кількість ядер та необхідну пам'ять.
* AWS пропонує тимчасове сховище, яке буде призначено при запуску і знищенні екземпляра при його припиненні та S3 для зберігання об'єктів. Тоді як Azure пропонує тимчасове зберігання через блокове зберігання через сторінку Blobs для VM та Block Blobs для об'єктів.
* AWS пропонує віртуальну приватну хмару, щоб користувач міг створювати ізольовані мережі в межах хмари, тоді як Azure пропонує віртуальну мережу, за допомогою якої ми можемо створювати ізольовані мережі, підмережі, таблиці маршрутів, приватний діапазон IP-адрес, такий самий, як і в AWS.
* Azure відкрита для гібридних хмарних систем, тоді як AWS менш відкрита для приватних або сторонніх постачальників хмарних технологій.
* AWS дотримується оплати, коли беруть плату за годину, тоді як Azure також слідкує за платою і вони стягують за хвилину, що забезпечує більш точну модель ціноутворення, ніж AWS.
* AWS має більше функцій та конфігурацій, і він пропонує велику гнучкість, потужність та налаштування з підтримкою багатьох інтеграційних інструментів сторонніх виробників. Тоді як Azure буде простий у використанні, якщо ми знайомі з Windows, оскільки це платформа Windows, і легко інтегрувати локальні сервери Windows з хмарними екземплярами, щоб створити гібридне середовище.

Взвживши усі «за» та «проти», було прийнято рішення розгорнути дану систему на базі AWS, оскільки є безкоштовний пробний період у розмірі цілого року. Звичайно, далеко не всі функції будуть доступними за безкоштовний період, але для оцінки швидкодії системи буде більш ніж досить. Також, якщо буде йти питання про масштабування системи, то в AWS є дуже багато рішень.

AWS розділений на різні сервіси, і кожен може бути налаштований по-різному залежно від потреб користувача. Користувачі повинні мати можливість бачити параметри конфігурації та окремі серверні карти для послуги AWS.

Більше 100 послуг складають портфоліо Amazon Web Services, включаючи послуги з обчислень, баз даних, управління інфраструктурою, розробкою додатків та безпекою. Ці послуги за категоріями включають:

* Обчислювання
* Бази даних для зберігання
* Управління даними
* Міграція
* Гібридна хмара
* Мережа
* Інструменти розробки
* Управління
* Моніторинг
* Безпека
* Управління
* Управління великими даними
* Аналітика
* Штучний інтелект (ШІ)
* Мобільний розвиток
* Повідомлення та пуш нотифікації

Доступність. Amazon Web Services надає послуги з десятків центрів обробки даних, розподілених по областям доступності (AZ) в регіонах по всьому світу. AZ - це місце розташування, яке зазвичай містить кілька фізичних центрів обробки даних, в той час як регіон представляє собою набір AZ в географічній близькості, пов'язаних мережевими лініями з малою затримкою. Компанія вибирає одну або кілька зон присутності з різних причин, таким як актуальність і близькість до кінцевих клієнтам. Наприклад, клієнт AWS може розкручувати віртуальні машини (ВМ) і копіювати дані в різні бази даних, щоб отримати високонадійну інфраструктуру, стійку до збоїв окремих серверів або цілого центру обробки даних. Amazon Elastic Compute Cloud (EC2) - це сервіс, який надає віртуальні сервери - так звані екземпляри - для розрахунку потужності. EC2 пропонує десятки типів примірників з різними ємностями і розмірами, адаптованими до конкретних типів роботи і додатків, таким як завдання з використанням пам'яті і прискорені обчислення. AWS також пропонує інструмент автоматичного масштабування для динамічного масштабування ємності для підтримки працездатності і продуктивності примірника.

Місце зберігання. Amazon Simple Storage Service (S3) надає масштабується сховище для об'єктів для резервного копіювання, архівування та аналізу даних. ІТ-фахівець зберігає дані і файли в вигляді об'єктів S3 - розміром до 5 ГБ - в області S3 для їх організації. Підприємства можуть економити гроші за допомогою S3 через рівні зберігання для нечастого доступу або використовувати льодовик Amazon для тривалого зберігання в холодильнику. Amazon Elastic Block Store надає сховище томів для постійного зберігання для використання з екземплярами EC2, а файлова система Amazon Elastic пропонує хмарне кероване сховище файлів. Підприємства також можуть переносити дані в хмару з допомогою таких пристроїв зберігання, як AWS Snowball і Snowmobile або використовувати AWS Storage Gateway, щоб локальні додатки могли отримувати доступ до хмарним даними.

Бази даних, управління даними. Служба реляційних баз даних Amazon - яка включає варіанти для Oracle, різні версії SQL, MariaDB та власну високопродуктивну базу даних під назвою Amazon Aurora – вона забезпечує реляційну систему управління базами даних для користувачів AWS. AWS також пропонує керовані бази NoSQL через Amazon DynamoDB. Клієнт AWS може використовувати Amazon ElastiCache і DynamoDB Accelerator як кеш даних в режимі реального часу та додатків. Amazon Redshift пропонує сховище даних, що полегшує аналітикам даних виконання завдань бізнес-розвідки.

Міграція, гібридне хмара. AWS включає в себе безліч інструментів і сервісів, призначених для допомоги користувачам в перенесенні додатків, баз даних, серверів і даних в їх загальне хмара. Центр міграції AWS пропонує місце для моніторингу та управління хмарної міграцією. Опинившись в хмарі, EC2 Systems Manager допомагає ІТ-команді налаштувати локальні сервери і екземпляри AWS. Amazon також має партнерські відносини з декількома постачальниками технологій для спрощення розгортання гібридних хмар. VMware Cloud в AWS передає програмно-яка визначається технологію центру обробки даних з VMware в хмару AWS. Red Hat Enterprise Linux для Amazon EC2 - продукт іншого партнерства, розширює операційну систему Red Hat до хмари AWS.

Мережа. Amazon Virtual Private Cloud (Amazon VPC) надає адміністратору контроль над віртуальною мережею, щоб використовувати розділ ізольованого хмари AWS. AWS автоматично надає нові ресурси VPC для додаткового захисту. Адміністратори можуть балансувати мережевий трафік за допомогою Elastic Load Balancing Service (ELB), яка включає в себе Application Load Balancer і Network Load Balancer. AWS також пропонує систему доменних імен під назвою Amazon Route 53, яка направляє кінцевих користувачів до додатків. ІТ-фахівець може встановити спеціальне з'єднання між локальним центром обробки даних і хмарою AWS через AWS Direct Connect.

Інструменти розробника. Розробник може використовувати інструменти командного рядка AWS і комплекти розробки програмного забезпечення (SDK) для розгортання додатків і служб і управління ними. Інтерфейс командного рядка AWS - це власний інтерфейс коду Amazon. Розробник також може використовувати AWS Tools for Powershell для управління хмарними службами з Windows і AWS Serverless Model Application для моделювання середовища AWS для тестування функцій Lambda. Пакети AWS SDK доступні на різних платформах і мовах програмування, включаючи Java, PHP, Python, Node.js, Ruby, C ++, Android і iOS. Amazon API Gateway дозволяє команді розробників створювати, управляти і контролювати свої власні API, які дозволяють додаткам отримувати доступ до даних або функцій зі служб резервного копіювання. API шлюзу обробляє тисячі викликів API одночасно. AWS також пропонує сервіс пакетної перекодування мультимедіа Amazon Elastic Transcoder і сервіс візуалізації робочих процесів для додатків мікросервісов, AWS Step Functions. Команда розробників може також створювати безперервну інтеграцію та постійні трубопроводи з послугами, такими як:

* AWS CodePipeline
* AWS CodeBuild
* AWS CodeDeploy
* AWS CodeStar

Розробник також може зберігати код в репозиторіях за допомогою AWS CodeCommit і оцінювати продуктивність додатків мікросервісов за допомогою AWS X-Ray.

Управління та моніторинг. Адміністратор може керувати і контролювати конфігурацію хмарних ресурсів, використовуючи AWS Config і AWS Config Rules. Ці інструменти разом з AWS Trusted Advisor можуть допомогти вашій ІТ-команді уникнути недоречних і непотрібних дорогих хмарних ресурсів. AWS пропонує кілька інструментів автоматизації в своєму портфоліо. Адміністратор може автоматизувати розробку інфраструктури за допомогою шаблонів AWS CloudFormation і використовувати AWS OpsWorks і Chef для автоматизації інфраструктури і призначених для користувача систем. Клієнт AWS може відстежувати стан ресурсів і додатків, використовуючи Amazon CloudWatch і панель моніторингу працездатності AWS, і використовувати AWS CloudTrail для зберігання активності призначеного для користувача інтерфейсу і прикладного програмування (API) для аудиту.

Керування захистом і. AWS надає повний спектр послуг хмарної безпеки, включаючи AWS Identity і Access Access, дозволяючи адміністратору ідентифікувати і управляти доступом користувачів до ресурсів. Адміністратор також може створити каталог користувачів за допомогою Amazon Cloud Directory або підключити хмарні ресурси до існуючої Microsoft Active Directory за допомогою служби каталогів AWS. Крім того, організація AWS дозволяє компанії створювати і управляти політиками для декількох облікових записів AWS. Хмарні провайдери також представляють інструменти, які автоматично оцінюють потенційні загрози національній безпеці. Інспектор Amazon аналізує середу AWS на наявність вразливостей, які можуть вплинути на безпеку і відповідність вимогам. Amazon Macie використовує технологію машинного навчання для захисту конфіденційних хмарних даних. AWS також включає в себе інструменти і сервіси, які використовують програмне забезпечення та апаратне шифрування, захищають від DDoS-атак, вимагають сертифікатів рівня захисту і транспортного рівня і фільтрації потенційно небезпечних веб-додатків. Сертифіковані спеціалісти AWS повинні мати можливість демонструвати і перевіряти технічні знання і навички хмарних обчислень в середовищах AWS. Консоль управління AWS - це графічний інтерфейс для AWS. Консоль управління може використовуватися для управління ресурсами в хмарних сховищах і облікових даних безпеки. Консоль AWS взаємодіє з усіма ресурсами AWS.

Управління великими даними і аналітика. AWS включає в себе безліч аналітичних і прикладних сервісів Big Data. Amazon Elastic MapReduce пропонує середу Hadoop для обробки великих обсягів даних, а Amazon Kinesis надає кілька інструментів для обробки і аналізу потокових даних. AWS Glue - це сервіс, який витягує, перетворює і завантажує завдання, а Amazon Elasticsearch дозволяє групі виконувати моніторинг додатків, аналіз журналів і інші завдання за допомогою інструменту Elasticsearch з відкритим вихідним кодом. Для запиту даних аналітик може використовувати Amazon Athena для S3, а потім візуалізувати дані за допомогою Amazon QuickSight.

Штучний інтелект. AWS пропонує широкий спектр платформ для розробки і доставки ШІ, а також пов'язаних додатків на основі ШІ. Інструментарій Amazon AI включає в себе Amazon Lex для технології голосового та текстового чату; Amazon Polly для перекладу тексту в мову; і Amazon Recovery для аналізу зображень та осіб. AWS також пропонує розробникам технологію для створення інтелектуальних додатків, заснованих на технології машинного навчання і складних алгоритмах. За допомогою AMS Deep Learning AWS (Amazon Machine Image) розробники можуть створювати і навчати призначені для користувача моделі штучного інтелекту за допомогою кластерів графічних процесорів або обчислювати оптимізовані екземпляри. AWS також включає в себе середовище розробки глибокого навчання для MXNet і TensorFlow. Для споживачів технології AWS засновані на голосових сервісах Alexa, і розробник може використовувати набір навичок Alexa для створення голосових додатків для пристроїв Echo.

Мобільна розробка. AWS Mobile Hub пропонує набір інструментів і сервісів для розробників мобільних додатків, в тому числі AWS Mobile SDK, який надає зразки коду та бібліотеки. Розробник мобільного застосування може також використовувати Amazon Cognito для управління доступом користувачів до мобільних додатків, а також Amazon Pinpoint для відправки push-повідомлень кінцевим користувачам додатки, а потім аналізувати ефективність цих повідомлень.

Відомості та push-повідомлення. Сервіси обміну повідомленнями AWS забезпечують базову зв'язок для користувачів і додатків. Amazon Simple Queue Service (SQS) - це керована чергу повідомлень, яка відправляє, зберігає і приймає повідомлення між компонентами розподілених додатків, щоб гарантувати, що частини програми працюють як задумано. Amazon Simple Notification Service (SNS) дозволяє компаніям відправляти паби на кінцеві точки, такі як кінцеві користувачі або служби. SNS включає функцію мобільного обміну повідомленнями, яка дозволяє відправляти повідомлення на мобільні пристрої. Amazon Simple Email надає платформу для ІТ-фахівців і маркетологів для відправки та отримання електронної пошти.

Моделі ціноутворення AWS і конкуренція. AWS пропонує модель оплати для своїх хмарних сервісів, або за годину, або за секунду. Також можливо зарезервувати певну кількість обчислювальної потужності за зниженою ціною для клієнтів, які платять повністю або підписуються на одно- або трирічні зобов'язання по використанню. Якщо потенційні клієнти не можуть собі цього дозволити, AWS Free Tier - ще один можливий спосіб використання сервісів AWS. Безкоштовний рівень AWS дозволяє користувачам отримати безкоштовний досвід роботи з сервісами AWS. Користувачі можуть використовувати до 60 продуктів і почати роботу на AWS з безкоштовним рівнем. Безкоштовний рівень пропонується в трьох різних варіантах: завжди безкоштовно, 12 місяців безкоштовно і пробна версія. AWS в основному конкурує з Microsoft Azure, Google і IBM на ринку публічної інфраструктури як послуга.

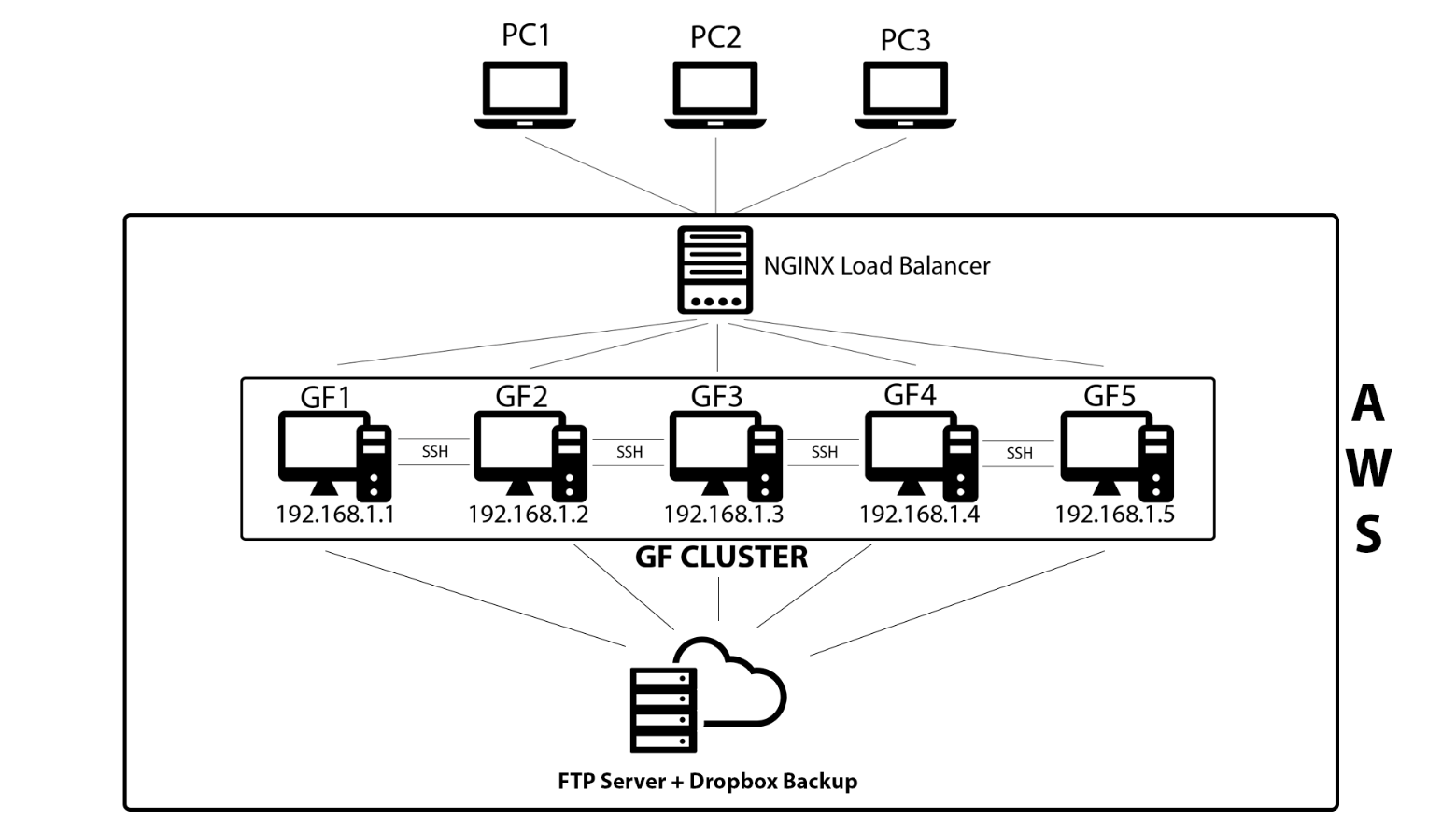


Рис. 2.30 Схема оновленої інфраструктури з використанням хмарного середовища

## 2.4. Висновки до розділу

В даному розділі дипломної роботи була реалізована відмовостійка система з кластеру із 5-ти вузлів на базі Glassfish Open Source Server 4.0 та балансуванням навантаження на базі NGINX. Були описані конфігурації кожного з серверу, що використовується в системі. Для кластеру це більш потужні сервери, для балансувальника навантаження трохи слабший. У ході роботи ми налаштували повністю систему, кластеризували її та додали балансувальника навантаження для більш стабільного з’єднання з серверами для клієнтів.

# ВИСНОВКИ

У ході роботи ми розглянули та вирішили по-етапно проблеми, зв’язані із інфраструктурою [24][25] та витратами на масштабування системи. Результатом вийшла готова інфраструктура, що знаходиться у хмарі, має резервне копіювання в іншу хмару, яка захищена і коштує мінімальну суму.

У статтях [24][25] було визначено, що наразі ефективність відмовостійкості потребує доопрацювання, тож у цій роботі ми знаходили один із варіантів підвищення завадостійкості кластерної архітектури.

У ході даної роботи були виконані наступні завдання:

* Був проведений аналіз стану даного питання за підсумками останніх досмліджень та публікацій;
* Виявлені та класіфіковані основні чинники, що впливають на відмовостійкість та доступність кластерних систем;
* Розглянута стуктура та основні складові кластерних систем;
* Проаналізувано балансування навантаження в умовах резервування;
* Побудувано та протестовано відмовостійкий кластер на основі додатку Oracle Glassfish Open Source Server 4.0;

Давайте протестуємо тепер швидкодію у порівнянні з нашими першими трьома тестами на різних платформах. У цьому тесті ми будемо порівнювати одну і ту саму платформу – Ubuntu Server 18.04. Оскільки у перших тестах ми перевіряли запуск, відновлення та вимкнення серверу Glassfish Open Source Server 4.0, то зараз ми будемо порівнювати ті результати із одним із вузлів кластеру.

Перший тест у нас є на запуск ноди кластеру:

Рис. 2.31 Різниця у часі при старті вузла

Як ми бачимо, різниця мінімальна, але вона є за рахунок використання SSD у хмарному сховищі.

Зробимо наступний тест із відновленням вже створеного вузла:

Рис. 2.32 Різниця у часі відновлення вузла

Як ми бачимо на рис. 2.32, різниця вже більш суттєва, майже 2 секунди. Перейдемо до останнього тесту із вимкненням вузла у кластері Glassfish Open Source Server 4.0:

Рис. 2.33 Різниця у часі відновлення вузла

Що ж, різниця хоч і не велика, але хмарне сховища має ряд переваг, як з точки зору безпеки, так і з точки зору маштабування.

Не можемо поговорити і про фінансову частину. Якщо інфраструктура планує дуже часто масштабуватися, інформація, що зберігається на серверах – конфіденційна, а клієнтів дуже багато, то розгорнувши серверну інфраструктуру на хмарі AWS можна зекономити більш ніж 50% від затрат. У випадку ж, коли у вас все є сервера, ви недовіряєте хмарним сховищам і хочете зберігати усю інформацію у себе, а швидкодії та потужності серверів досить для обробки клієнтської інформації, тоді необхідності до переходу до хмарного сховища відпадає, але слід відмітити, що «хмарна» інфраструктура вирішує дуже велику низку проблем за невеликі гроші.

Також, слід відзначити, що швидкість обробки сервером якоїсь інформації може різнитися в залежності від апаратної частини, якщо є виділені сервера, або від місцезнаходження та потужності хмарних серверів.

Головною метою даної роботи є підвищення ефективності завадостійкої кластерної системи на базі Oracle Glassfish Open Source Server 4.0.

Тож, підбиваючи підсумки, у ході даної роботи ми розглянули такі види серверної інфраструктури як:

* Один сервер та клієнти;
* Декілька серверів та клієнтів;
* Відмовостійкий кластер із серверів та клієнти;
* Відмовостійкий кластер із серверів з використанням балансувальника навантаження;
* Додаткові рішення: побудова FTP серверу з хмарним резервуванням та хмарна інфраструктура серверів.

Ми виділили плюси та мінуси кожної із цих архітектур і підсумовуючи усе вище розглянуте, можна прийти до висновку, що найбільш оптимальним рішення для малих організацій є один сервер, коли ж організація вже має деяку базу клієнтів, найбільш вдалим буде рішення це розгорнути відмовостійкий кластер із серверів з використанням балансувальника навантаження та спільним FTP сервером, що має резервне копіювання до хмарного сховища.

Тож кожен має вибрати під свої потреби та кошти спосіб, щоб предоставляти послуги клієнтам за найвигідніші кошти.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Protection of Information in Computer Systems*. /* Saltzer H., Schroeder D., Vol. 63, no. 09 (September). - 1975.  -  p. 1278-1308.
2. Information technology. Security techniques. Information security management systems. Overview and vocabulary. [Electronic resource] / ISO/IEC 27000:2018. – 2019. - – Access mode: World Wide Web. - URL: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&uact=8&ved=2ahUKEwj2wtLnpZzlAhUjAxAIHdyrDqsQFjADegQIBxAC&url=https%3A%2F%2Fwww.sis.se%2Fapi%2Fdocument%2Fpreview%2F80001198%2F&usg=AOvVaw1tWFIo1ZigiLm_eqAC3K4D> - Title from the screen.
3. The DDoS That Almost Broke the Internet. [Electronic resource] / Prince Matthew. - 2013. – Access mode: World Wide Web. - URL: <https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet> - Title from the screen.
4. Q2 2018 DDoS Trends report: 52 percent of attacks employed multiple attack types. [Electronic resource] / [Title’s author] – 2018. – Access mode: World Wide Web. - URL: <https://blog.verisign.com/security/ddos-protection/q2-2018-ddos-trends-report-52-percent-of-attacks-employed-multiple-attack-types> - Title from the screen.
5. Network attacks. [Electronic resource] / [Title’s author] – 2017. – Access mode: World Wide Web. - URL: <https://www.geeksforgeeks.org/basic-network-attacks-in-computer-network/> - Title from the screen.
6. Qijun Gu, Peng Liu. Denial of Service Attacks [Electronic resource] / University Park – 2014. – Access mode: World Wide Web. - URL: s2.ist.psu.edu/paper/DDoS-Chap-Gu-June-07.pdf - Title from the screen.
7. How to Defend Against DDoS Attacks: Six Steps. [Electronic resource] / [Title’s author] – 2018. – Access mode: World Wide Web. - URL: <https://www.a10networks.com/blog/how-defend-against-ddos-attacks-six-steps> - Title from the screen.
8. General information about cybersecurity and network security. [Electronic resource] / [Title’s author] – 2019. – Access mode: World Wide Web. - URL: <https://www.technewsworld.com/perl/section/cyber-security/> - Title from the screen.
9. Deep cybersecurity digest. [Electronic resource] / [Title’s author] – 2018. – Access mode: World Wide Web. - URL: <https://proglib.io/p/security-digest/> - Title from the screen.
10. Віртуалізація та відмовостійкість. [Електроний ресурс] / [Автори статті]. – 2016. – Режим доступу: World Wide Web. – URL: <http://integritysys.com.ua/solutions/ha-virtualization/> - стаття на головній сторінці.
11. Про сервери: просто і доступно. [Електроний ресурс] / [Автори статті]. – 2016. – Режим доступу: World Wide Web. – URL: <https://erc.ua/erc-reviews/19879/pro-serveri-prosto-i-dostupno/> - стаття на головній сторінці.
12. High availability (HA) [Electronic resource] / [Title’s author]. – 2019. – Access mode: World Wide Web. - URL: <https://searchdatacenter.techtarget.com/definition/high-availability> - Title from the screen.
13. Задачи оптимального резервирования [Электронный ресурс] / [Автор статьи]. – 2018. – Режим доступа: World Wide Web. – URL: <https://studizba.com/lectures/10-informatika-i-programmirovanie/350-nadezhnost-informacionnyh-sistem/4746-18-zadachi-optimalnogo-rezervirovaniya.html> - статья на главной странице.
14. Что такое баласировка нагрузки и как она работает. [Электронный ресурс] / [Автор статьи]. – 2016. – Режим доступа: World Wide Web. – URL: <https://www.servers.ru/knowledge/load-balancing/what-is-load-balancing-and-how-it-works> - статья на главной странице.
15. Кластер серверов [Электронный ресурс] / [Автор статьи]. – 2018. – Режим доступа: World Wide Web. – URL: <https://itelon.ru/solution/Cluster-servers/> - статья на главной странице.
16. Операционная система Windows [Электронный ресурс] / [Автор статьи]. – 2016. – Режим доступа: World Wide Web. – URL: <http://procomputer.su/operatsionnye-sistemy/66-os-windows> - статья на главной странице.
17. Операционная система Linux [Электронный ресурс] / [Автор статьи]. – 2016. – Режим доступа: World Wide Web. – URL: <https://www.linux.org/> - статья на главной странице.
18. Налаштування Oracle Glassfish Open Server під операційною системою Ubuntu [Електроний ресурс] / [Автори статті]. – 2014. – Режим доступу: World Wide Web. – URL: <https://community.vscale.io/hc/ru/community/posts/208345489-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%B8-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-GlassFish4-%D0%BD%D0%B0-Ubuntu-16-04> – стаття по налаштуванню GF під Ubuntu.
19. Apache Vs NGINX – Which Is The Best Web Server for You? [Electronic resource] / [Title’s author] – 2018. – Access mode: World Wide Web. - URL: <https://serverguy.com/comparison/apache-vs-nginx/> - Title from the screen.
20. Apache vs Nginx: Practical Considerations [Electronic resource] / [Title’s author] – 2019. – Access mode: World Wide Web. - URL: <https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations> - Title from the screen.
21. Как установить и настроить FTP-сервер на Ubuntu Linux [Электронный ресурс] / [Автор статьи.] – 2016. – Режим доступа: World Wide Web. – URL <https://www.servers.ru/knowledge/linux-administration/how-to-setup-ftp-server-on-ubuntu-16-04> - статья на главной странице.
22. Dropbox Uploader. [Electronic resource] / [Title’s author] – 2019. – Access mode: World Wide Web. - URL: <https://github.com/andreafabrizi/Dropbox-Uploader> - Title from the screen.
23. What is AWS? [Electronic resource] / [Title’s author] – 2019. – Access mode: World Wide Web. – URL: <https://www.guru99.com/what-is-aws.html> - Title from the screen.
24. Завадостійкий кластер для задач з екологічного менеджменту. [Електроний ресурс] / Копитовський М.В., Павлов В.Г., – 2019. – Режим доступу: World Wide Web. – URL: <http://www.ecoj.dea.kiev.ua/archives/2019/4/35.pdf> - стаття на головній сторінці.
25. Відмовостійкий кластер на базі серверів додатків Glassfish. [Текст] / Копитовський М.В., Павлов В.Г., – 2019. – №1(11) - Прийнята до друку та публікації.