Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Застосування мікросерверної архітектури для побудови відмовостійкої хмарної системи»

Студента 2м курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ Журбенка Максима Олеговича
підпис студента

Науковий керівник
Старший викладач кафедри програмної інженерії та кібербезпеки Десятко Альона Миколаївна
_____
підпис керівника

Гарант освітньої програми
доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки Криворучко Олена Володимирівна
_____
підпис гаранта

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Журбенку Максиму Олеговичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Затосування мікросерверної архітектури для побудування відмовостійкої хмарної системи»

Затверджена наказом ректора від "13" грудня 2019 р. № 4304

2. Строк здачі студентом закінченої проекту 01 грудня 2020

3. Цільова установка та вихідні дані до проекту

Мета проекту усунення ризиків, пов'язаних з уразливістю хмарного серидовища в наслідок відмови роботи однієї або більше внутрішніх систем.

Об'єкт дослідження процес розробки хмарної системи

Предмет дослідження розробка хмарної системи.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ХМАРНИХ ТЕХНОЛОГІЙ

1.1. Опис хмарних систем

1.2. Обгрунтування вибору Microsoft Azure

1.3. Середовище розробки Visual Studio

1.4. Висновок до розділу 1

РОЗДІЛ 2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ ХМАРНОЇ СИСТЕМИ

2.1. Мова програмування C#

2.2. Архітектура додатку

2.3. Підходи побудови відмовостійкої системи

2.3. Висновок до розділу 2

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ХМАРНОЇ СИСТЕМИ

3.1. Мікросерверна архітектура

3.2. Моделювання хмарної системи

3.3. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Аналіз предметної області застосування хмарних технологій*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Вибір програмних засобів та проектування хмарної системи*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Реалізація хмарної системи*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*8.12.2020-9.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Криворучко О.В

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О.В.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Журбенко М.О.

(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист Десятко А.М

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _____

(прізвище, ініціали)

може бути допущений до захисту екзаменаційній комісії.

Гарант освітньої програми Криворучко О. В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці хмарної системи, що має більшу відмовостійкість завдяки мікросерверній архітектурі.

В результаті порівняльного аналізу аналогічних рішень визначено вагомі недоліки подібних хмарних систем, що були прийняті до уваги та усунені у нашій розробці.

Розробка була виконана у серидовищі Visual Studio. Обрана мова програмування в обох випадках – C#.

Готову хмарну систуему було успішно протестовано відповідно до функціональних вимог.

Ключові слова: Visual Studio, хмарна система, C#, Microsoft Azure.

ABSTRACT

For example, prior to the progress of the robot, the robotized system is assigned, so there is more flexibility in the microserver architecture.

As a result of a detailed analysis of analogous solutions, there are some shortcomings of similar systems that will be taken to the point of respect for our devices.

The development of the Bull Viconan at the Visual Studio Seridovisch. The MOV program was copied in both versions - C #.

I am ready for the system, successfully protested to the point of being functional.

Key words: Visual Studio, xmarna system, C #, Microsoft Azure.

ЗМІСТ

					КНТЕУ 121 06м-09.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*19.10.20*	Застосування мікросерверної архітектури для побудування відмовостійкої хмарної системи	*Стадія*	*Арку*	*Арку*
Керівник		Десятко А. М.		*19.10.20*		*Зміст*	*2*	*36*
Гарант		Криворучко О.В.		*19.10.20*		*Факультет інформаційних технологій 2м курс, 6м група*		
Розробив		Журбенко М.О.		*19.10.20*	*Зміст*			

ВСТУП

Актуальність. Сьогодні важко посперечатися з тим, що технології, засновані на хмарні обчислення, неймовірно затребувані і активно розвиваються. Під технологією хмарних обчислень (cloud computing) розуміється технологія, яка дозволяє об'єднувати ІТресурси різних апаратних платформ в єдине ціле і надавати користувачеві доступ до них через мережу Інтернет. Хмарні сервіси надають користувачам через мережу Інтернет доступ до своїх ресурсів за допомогою безкоштовних або умовно безкоштовних хмарних додатків, програмні та апаратні вимоги яких не припускають наявності у клієнтів високопродуктивних комп'ютерів.

В даний час виділяють три моделі обслуговування хмарних обчислень:

1. Програмне забезпечення (ПЗ) як послуга (SaaS, Software as a Servise). Споживачеві надаються програмні засоби - додатки провайдера, що виконуються на хмарної інфраструктурі.

2. Платформа як послуга (PaaS, Platform as a Service). Споживачеві надаються кошти для розгортання на хмарної інфраструктурі створюваних споживачем або придбаних додатків, що розробляються з використанням підтримуваних провайдером інструментів і мов програмування.

3. Інфраструктура як послуга (IaaS, Infrastructure as a Service). Споживачеві надаються кошти обробки даних, зберігання, мереж та інших базових обчислювальних ресурсів, на яких можна розгортати і виконувати

довільний програмне забезпечення, включаючи операційні системи і додатки.

Мета дослідження: усунення ризиків, пов'язаних з уразливістю хмарного серидовища в наслідок відмови роботи однієї або більше внутрішніх систем.

Об'єкт дослідження: процес розробки хмарної системи.

*Предмет дослідження***:** розробка хмарної системи на базі Microsoft Azure.

У відповідності з метою дослідження поставлені наступні *завдання*:

1) визначити слабкі місця хмарних систем;

2) дослідити вимоги користувачів хмарних систем та сформувати функціональний блок технічного завдання на розробку системи;

3) проаналізувати та обґрунтувати вибір програмного забезпечення та технологій, що використовуватимуться у процесі розробки;

4) провести тестування готового рішення на реальних пристроях та розробити інструкцію користувача.

Методи дослідження, що були використані у роботі: аналіз, абстрагування, порівняння.

Наукова новизна дослідження полягає в удосконаленій концепції хмарної системи з вирішенням проблем відмовостійкості.

*Практичне значення дослідження***:** готове альтернативне рішення у вигляді хмарної системи менш вразливої до відмов.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ХМАРНОЇ СИСТЕМИ

1.1. Опис хмарних систем

Слово «хмара» (cloud) використовувалося в 1990-х роках для метафоричного позначення Інтернету: тоді Глобальна мережа представлялася чимось загадковим, невизначеним в своїх просторових межах, не відрізнятись від своїх внутрішніх елементів і швидко мінливих. Ідея «хмарних обчислень» була сформульована ще в 1960 році Джоном Мак-Карті, фахівцем з обчислювальної техніки. Він висловив припущення, що коли-небудь обчислення будуть організовані за принципом комунальних послуг, тобто будуть надаватися за окрему плату. У 1993 році термін «хмара» був вперше використаний в комерційних цілях для опису великих мереж, що задіюють технологію високошвидкісної одночасної передачі трафіку всіх видів (дані, голос і відео) в мережах з комутованими каналами. У них з'являлося проміжне віртуальне з'єднання між відправником і отримувачем, що спрощує процес передачі інформації. Розширення пропускної здатності Інтернету, в 90-ті роки не дозволило отримати значний стрибок у розвитку в хмарної технології, так як практично жодна компанія, ні технології того часу не були готові до цього. Однак сам факт прискорення Інтернету дав поштовх швидкому розвитку хмарних обчислень.

На початку XXI століття термін «хмарні обчислення» став вживатися стосовно створеному тоді напрямку SaaS (Software as a Service - «програмне

забезпечення як послуга»). Першопрохідцем в цьому відношенні став інтернет-магазин Amazon, який викрутився зі складної ситуації в період кризи доткомів шляхом перекладу своїх дата-центрів на рішення Open Source. 90% серверів компанії стали працювати на базі операційної системи Red Hat Linux, а апаратне забезпечення замінили на недорогі моделі серверів на основі чіпсетів від Intel і HR. У 2002 році були створені веб-сервіси Amazon. Вони представляли собою те, що через п'ять років стало називатися «хмарою», - набір сервісів, розташованих на віддалених серверах, до яких користувач може отримати доступ через веб-браузер з будь-якого пристрою, що має доступ в мережу Інтернет.

У 2007 році в подібний проект (Academic Cluster Computing Initiative), в якому брали участь Google і IBM, включилися кілька американських університетів. Для них ці компанії побудували дата-центри на 1600 серверів і оснастили їх відповідним програмним забезпеченням для управління і здійснення віддаленого доступу до обчислювальних ресурсів. Також в гонку за «хмарами» вступили Yahoo !, Microsoft і eBay, а 2008 рік комп'ютерна індустрія зустрічала вже під «хмарою»: аналітики навперебій розхвалювали нову стратегію оптимізації витрат за рахунок відмови від високопродуктивних комп'ютерів на користь інтернет-сервісів на зразок «Документи Google» .

З тих пір розвиток «хмар» проходило стрімко, багато компаній перейшли на них при першій нагоді, а незабаром з'явилися і сервіси, що надають послуги розподілених обчислень своїм клієнтам.

Розвиток і дослідження в цій області проходять і зараз - в 2008 корпорації HP, Intel, і Yahoo! створили спільну обчислювальну лабораторію Cloud Computing Test Bed, спрямовану на вдосконалення хмарних технологій

і прийомів роботи з ними. Національним інститутом стандартів і технологій США зафіксовані такі обов'язкові характеристики хмарних обчислень:

1. Самообслуговування на вимогу (англ. Self service on demand), споживач самостійно визначає і змінює обчислювальні потреби, такі як серверний час, швидкості доступу та обробки даних, обсяг збережених даних без взаємодії з представником постачальника послуг;

2. Універсальний доступ по мережі, послуги доступні споживачам через мережу передачі даних незалежно від використовуваного термінального пристрою;

3. Об'єднання ресурсів (англ. Resource pooling), постачальник послуг об'єднує ресурси для обслуговування великого числа споживачів в єдиний пул для динамічного перерозподілу потужностей між споживачами в умовах постійної зміни попиту на потужності; при цьому споживачі контролюють тільки основні параметри послуги (наприклад, обсяг даних, швидкість доступу), але фактичний розподіл ресурсів, що надаються споживачеві, здійснює постачальник (в деяких випадках споживачі все-таки можуть управляти деякими фізичними параметрами перерозподілу, наприклад, вказувати бажаний центр обробки даних з міркувань географічної близькості);

4. Еластичність, послуги можуть бути надані, розширені, звужені в будь-який момент часу, без додаткових витрат на взаємодію з постачальником, як правило, в автоматичному режимі;

5. Облік споживання, постачальник послуг автоматично обчислює спожиті ресурси на певному рівні абстракції (наприклад, обсяг збережених даних, пропускна здатність, кількість користувачів, кількість транзакцій), і на основі цих даних оцінює обсяг наданих споживачам послуг.

З точки зору постачальника, завдяки об'єднанню ресурсів і непостійного характеру споживання з боку споживачів, хмарні обчислення дозволяють економити на масштабах, використовуючи менші апаратні ресурси, ніж були потрібні б при виділених апаратних потужностях для кожного споживача, а за рахунок автоматизації процедур модифікації виділення ресурсів істотно знижуються витрати на абонентське обслуговування.

З точки зору споживача, ці характеристики дозволяють отримати послуги з високим рівнем доступності (англ. High availability) і низькими ризиками непрацездатності, забезпечити швидке масштабування обчислювальної системи завдяки еластичності без необхідності створення, обслуговування і модернізації власної апаратної інфраструктури.

Зручність і універсальність доступу забезпечується широкою доступністю послуг і підтримкою різного класу термінальних пристроїв (комп'ютерів, мобільних телефонів, інтернет-планшетів).

1.2. Обґрунтування вибору Microsoft Azure

Сьогодні великі компанії, такі як Amazon, Google і Microsoft, які мають великий досвід управління великими дата-центрами, пропонують клієнтам «орендувати» ємність даних. Це чудова новина для компаній, які хочуть зосередитися на своєму основному бізнес-додатку і не турбуватися про базову платформі. Хмарна платформа від Microsoft називається Windows Azure.

Microsoft Azure, найкраща хмарна платформа Microsoft, сприяла значному зростанню доходів Microsoft з моменту її запуску в 2010 році,

який співпав з триваючим переходом компанії на хмарний ринок шляхом формування стратегічних партнерських відносин з лідерами хмарних обчислень, такими як Salesforce.com. Azure не тільки володіє потужними можливостями «платформа як послуга» (PaaS), але і в даний час є єдиною великою хмарної платформою, яка є лідером в області інфраструктури як послуги (IaaS), згідно з рейтингом Gartner. Нещодавно Forbes передбачив, що річний дохід Azure складе близько 2,3 мільярда доларів.

Azure стає популярною хмарної інфраструктурою для багатьох IT-фахівців. Ось кілька причин, за якими ці професіонали звертаються до Azure як до своєї хмарної платформі - і, можливо, чому вашій організації також слід подумати про це:

Економія коштів і швидка масштабованість:

З такою моделлю, як Windows Azure, компаніям потрібно платити тільки за ресурси, які використовують їх застосування. Наприклад, якщо компанії необхідно збільшити число користувачів або сховище даних, Microsoft може просто відрегулювати їх швидкість, роблячи її надзвичайно зручною і масштабованої. При такому підході «оплата за фактом» підприємства платять тільки за обсяг простору, який їм потрібен, замість того, щоб платити за порожнє сховище в максимальній сумі, яку вони ніколи не зможуть використовувати.

Надійність:

Крім економії коштів і швидкої масштабованості, ще однією перевагою Windows Azure і моделі IaaS / PaaS є надійність. Служби Windows Azure надаються з хмарних центрів обробки даних, які мають кілька вбудованих надлишкових. У разі збою одного сервера додатки компанії автоматично запускаються на іншому сервері в центрі обробки даних.

Прості поновлення:

Microsoft Azure підтримує безліч різних мов програмування, інструментів і середовищ, в тому числі програмне забезпечення та системи, розроблені для Microsoft і сторонніх виробників. Нові веб-додатки і оновлення можуть бути легко додані.

Тісно інтегрований з іншими інструментами Microsoft:

Для організацій, які покладаються на інструменти Microsoft, такі як SharePoint (яка нещодавно була визнана платформою № 1 для корпоративної спільної роботи), Office 365 і Outlook, доцільно інвестувати в хмарну платформу, яка легко інтегрується з продуктами Microsoft. Організації можуть також використовувати ті ж віртуальні машини в Azure, що і локальні, наприклад Windows і Linux, що ще більше спрощує операції. Багато галузеві експерти очікують, що Azure буде повільно, але вірно набувати популярності завдяки цій здатності пропонувати користувачам повністю цілісний і інтегрований пакет послуг.

IaaS і PaaS:

Azure може похвалитися привабливою комбінацією служб IaaS (керованих) і PaaS (некерованих). IaaS дозволяє компаніям передавати свою інфраструктуру хмарних обчислень на аутсорсинг і платити тільки за те, що вони використовують. PaaS дозволяє компаніям створювати свої власні веб-додатки та / або програмне забезпечення без необхідності купувати і підтримувати базову інфраструктуру. Це дозволяє організаціям налаштовувати своє хмарне програмне забезпечення, наприклад, Office 365, відповідно до їх точними специфікаціями і вимогами. Оскільки Azure є лідером галузі в обох цих категоріях, компанії можуть швидше і простіше створювати, розгортати і керувати програмами.

1.3. Середовище розробки Visual Studio

Microsoft Visual Studio - це програмна средапоразработке додатків для ОС Windows, як консольних, так і з графічним інтерфейсом.

У комплект входять наступні основні компоненти:

1. Visual Basic.NET - для розробки додатків на VisualBasic;
2. Visual C ++ - на традиційній мові C ++;
3. Visual C # - наязике C # (Microsoft);
4. Visual F # - на F # (Microsoft Developer Division).

Функціональна структура середовища включає в себе:

- редактор вихідного коду, який включає безліч додаткових функцій, як автодоповнення IntelliSense, рефракторінг коду і т. д
- відладчик коду;
- редактор форм, призначений для спрощеного конструювання графічних інтерфейсів;
- веб-редактор;
- дизайнер класів;
- дізайнерсхем баз даних.

Visual Studio також дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (Subversion і VisualSourceSafe), додавання нових наборів інструментів (для редагування і візуального проектування коду на предметно-

орієнтованих мовах програмування або інструментів для інших аспектів процесу розробки програмного забезпечення).

Переваги й недоліки:

Інтегроване середовище розробки (IntegratedDevelopmentEnvironment - IDE) Visual Studio пропонує ряд високорівневих функціональних можливостей, які виходять за рамки базового управління кодом.

Нижче перераховані основні переваги IDE-середовища Visual Studio.

Вбудований Web-сервер. Для обслуговування Web-додатки ASP.NET необхідний Web-сервер, який буде очікувати Web-запити і обробляти відповідні сторінки. Наявність в Visual Studio інтегрованого Web-сервера дозволяє запускати Web-сайт прямо з середовища проектування, а також підвищує безпеку, виключаючи ймовірність отримання доступу до тестовомуWeb-сайту з якого-небудь зовнішнього комп'ютера, оскільки тестовий сервер може приймати з'єднання лише з локального комп'ютера.

Підтримка безлічі мов при розробці. Visual Studio дозволяє писати код своєю рідною мовою чи будь-яких інших бажаних мовами, використовуючи весь час один і той же інтерфейс (IDE). Більш того, Visual Studio також ще дозволяє створювати Web-сторінки на різних мовах, але поміщати їх все в один і той же Web-додаток. Єдиним обмеженням є те, що в кожної Web-сторінці можна використовувати тільки якийсь один мову (очевидно, що в іншому випадку проблем при компіляції було б просто не уникнути).

Менше коду для написання. Для створення більшості додатків потрібно пристойну кількість стандартного стереотипного коду, і Web-сторінки ASP. NET тому не виключення. Наприклад, додавання Web-елемента управління, приєднання обробників подій і коригування форматування вимагає установки в розмітці сторінки ряду деталей. У Visual Studio такі деталі встановлюються автоматично.

Інтуїтивний стиль кодування. За замовчуванням Visual Studio форматує код у міру його введення, автоматично вставляючи необхідні відступи і застосовуючи колірне кодування для виділення елементів типу коментарів. Такі незначні відмінності роблять код більш зручним для читання і менш схильним до помилок. Застосовувані Visual Studio автоматично параметри форматування можна навіть налаштовувати, що дуже зручно у випадках, коли розробник вважає за краще інший стиль розміщення дужок (наприклад, стиль K & R, при якому відкриває дужка розміщується на тому самому рядку, що і оголошення, якому вона передує).

Більш висока швидкість розробки. Багато з функціональних можливостей Visual Studio спрямовані на те, щоб допомагати розробнику робити свою роботу якомога швидше. Зручні функції, на зразок функції IntelliSense (яка вміє перехоплювати помилки і пропонувати правильні варіанти), функції пошуку і заміни (яка дозволяє відшукувати ключові слова як в одному файлі, так і в усьому проекті) і функції автоматичного додавання і видалення коментарів (яка може тимчасово приховувати блоки коду), дозволяють розробнику працювати швидко і ефективно.

Можливості налагодження. Пропоновані в Visual Studio інструменти налагодження є найкращим засобом для відстеження загадкових помилок і діагностування дивної поведінки. Розробник може виконувати свій код по рядку за раз, встановлювати інтелектуальні точки переривання, при бажанні зберігаючи їх для використання в майбутньому, і в будь-який час переглядати поточну інформацію з пам'яті.

Visual Studio також має і безліч інших функцій: можливість управління проектом; вбудована функція управління вихідним кодом; можливість рефакторізації коду; потужна модель розширюваності. Більш того, в разі використання Visual Studio 2008 Team System розробник

отримує розширені можливості для модульного тестування, спільної роботи і управління версіями коду (що значно більше того, що пропонується в більш простих інструментах на кшталт Visual SourceSafe).

Як недолік можна відзначити неможливість відладчика (Microsoft Visual Studio Debugger) відстежувати в коді режиму ядра. Налагодження в Windows в режимі ядра в загальному випадку виконується при використанні WinDbg, KD або SoftICE.

1.4 Висновки до розділу 1

Отже, основне меню та інтерфейс Visual Studio є досить легким в опануванні для розробників, що вже раніше користувалися різноманітними IDE у процесі роботи із кодом, та, з іншого боку, є інтуїтивно-зрозумілим для новачків у використанні IDE.

РОЗДІЛ 2
ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ ХМАРНОЇ СИСТЕМИ

2.1. Мова програмування C#

C # - це мова з C-подібним синтаксисом. Тут він близький в цьому відношенні до C ++ і Java.

Будучи об'єктно-орієнтованою мовою, він багато перейняв у Java і C ++. Як і Java, C # спочатку призначався для веб-розробки, і приблизно 75% його синтаксичних можливостей такі ж, як у Java. C # також називають «очищеної версією Java». Ще 10% наш герой запозичив з C ++ і 5% - з Visual Basic. Решта 10% C # - це реалізація власних ідей розробників. Об'єктно-орієнтований підхід дозволяє будувати за допомогою C # великі, але в той же час гнучкі, масштабовані і розгортаються додатки.

C # вже давно підтримує багато корисних функцій:

- інкапсуляція,
- успадкування,
- поліморфізм,
- перевантаження операторів,
- статична типізація.

При цьому він все ще активно розвивається, і з кожною новою версією з'являється все більше цікавого - наприклад лямбда, динамічне зв'язування, асинхронні методи і т.д.

У порівнянні з іншими мовами C # досить молодий, але в той же час він вже пройшов великий шлях. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є версія C # 8.0, яка вийшла у вересні 2019 року разом з релізом .NET Core 3. Коли говорять C #, нерідко мають на увазі технології платформи .NET (Windows Forms, WPF, ASP.NET, Xamarin). І навпаки, коли говорять .NET, нерідко мають на увазі C #. Однак, хоча ці поняття пов'язані, ототожнювати їх невірно. Мова C # був створений спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше.

Колись Білл Гейтс сказав, що .NET - це найкраще, що зробила компанія Microsoft. У нього є вагомі підстави так вважати. Фреймворк .NET представляє потужну платформу для створення додатків. Ось його «кілер-фічі»:

Підтримка декількох мов. В основі .NET - загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому платформа підтримує кілька мов: поряд з C # це VB.NET, C ++, F #, а також різні діалекти інших мов, прив'язані до .NET, наприклад, Delphi. NET. Код на будь-якому з цих мов компілюється в збірку спільною мовою CIL (Common Intermediate Language) - свого роду асемблер платформи .NET. Тому можна зробити окремі модулі однієї програми на різних мовах.

Потужна бібліотека класів. .NET представляє єдину для всіх підтримуваних мов бібліотеку класів. Яке б додаток ми не збиралися писати на C # - текстовий редактор, чат або складний веб-сайт - так чи інакше ми задіємо бібліотеку класів .NET.

Різноманітність технологій. Загальномовне середовище виконання CLR і базова бібліотека класів - це основа для цілого стека технологій, які

розробники можуть задіяти при створенні різних додатків. Наприклад, для баз даних в цьому стеці є технологія ADO.NET і Entity Framework Core. Для графічних додатків з насиченим інтерфейсом - технології WPF і UWP. Для більш простих графічних додатків - Windows Forms. Для розробки мобільних додатків - Xamarin. Для створення веб-сайтів - ASP.NET і т.д.

.NET довгий час розвивався під назвою .NET Framework - переважно як платформа для Windows. Але з 2019 вона більше не розвивається - останньою версією цієї платформи стала .NET Framework 4.8.

У 2014 Microsoft почав випускати альтернативну платформу - .NET Core, яка повинна була увібрати в себе всі можливості застарілого .NET Framework і додати нову функціональність. Тому слід розрізняти .NET Framework, який призначений переважно для Windows, і багатоплатформовий .NET Core.

Переваги і недоліки мови C #:

У «шарпа» виділяють багато переваг:

- Підтримка переважної більшості продуктів Microsoft
- Безкоштовність ряду інструментів для невеликих компаній і деяких індивідуальних розробників - Visual Studio, хмара Azure, Windows Server, Parallels Desktop для Mac Pro і ін.
- Типи даних мають фіксований розмір (32-бітний int і 64-бітний long), що підвищує «мобільність» мови і спрощує програмування, так як ви завжди знаєте точно, з чим ви маєте справу.
- Автоматична «прибирання сміття» Це означає, що нам в більшості випадків не доведеться піклуватися про звільнення пам'яті.

Вищезазначена загальномовного середовища CLR сама викличе збирач сміття і очистить пам'ять.

- Велика кількість «синтаксичного« цукру »- спеціальних конструкцій, розроблених для розуміння і написання коду. Вони не мають значення при компіляції.
- Низький поріг входження. Синтаксис C # має багато схожого з іншими мовами програмування, завдяки чому полегшується перехід для програмістів. Мова C # часто визнають найбільш зрозумілим і придатним для новачків.
- За допомогою Xamarin на C # можна писати програми і додатки для таких операційних систем, як iOS, Android, MacOS і Linux;
- Сьогодні в будь-якому регіоні Росії є чимало вакантних місць на посаду C # програміст.

Але є у C # і деякі недоліки:

- Пріоритетна орієнтованість на платформу Windows;
- Мова безкоштовний тільки для невеликих фірм, індивідуальних програмістів, стартапів і учнів. Великої компанії покупка ліцензійної версії цієї мови обійдеться в кругленьку суму.

2.4. Архітектура додатку

При розгляді хмарної архітектури потрібно обов'язково мати на увазі модель хмарних обчислень. Розглянемо архітектуру IaaS.

Хмара створюється з декількох фізичних вузлів, з'єднаних швидкими каналами передачі даних з метою єдиного управління та передачі великих

обсягів інформації. Слово «кілька» можна сприймати буквально - фактичних з 3-5 вузлів можна побудувати невелику хмару. Як воно буде працювати, вже друге питання. У реальності в дата-центрі хмарного провайдера - сотні і навіть тисячі вузлів.

За допомогою спеціального ПО віртуалізації і формування хмарної інфраструктуру користувачі можуть отримувати доступ до ресурсів хмари, при цьому не замислюючись ресурси якого саме фізичного вузла їм виділені. По суті, користувачеві все одно ресурси якого вузла він використовував і де виконується його завдання, де зберігаються його дані - важливо, щоб завдання було виконано, а дані залишилися в цілості й схоронності.

В хмарі створюються віртуальні машини, на яких запущені гостьові ОС і різні встановлені користувачем програми. У «виртуалке» може виконуватися будь-яка, по суті, «операційка» - Windows Server, Linux, FreeBSD і ін.

На одному апаратному вузлі можна запустити кілька десятків віртуальних машин, які здаються в оренду.

Архітектура хмари моделі IaaS виглядає так:

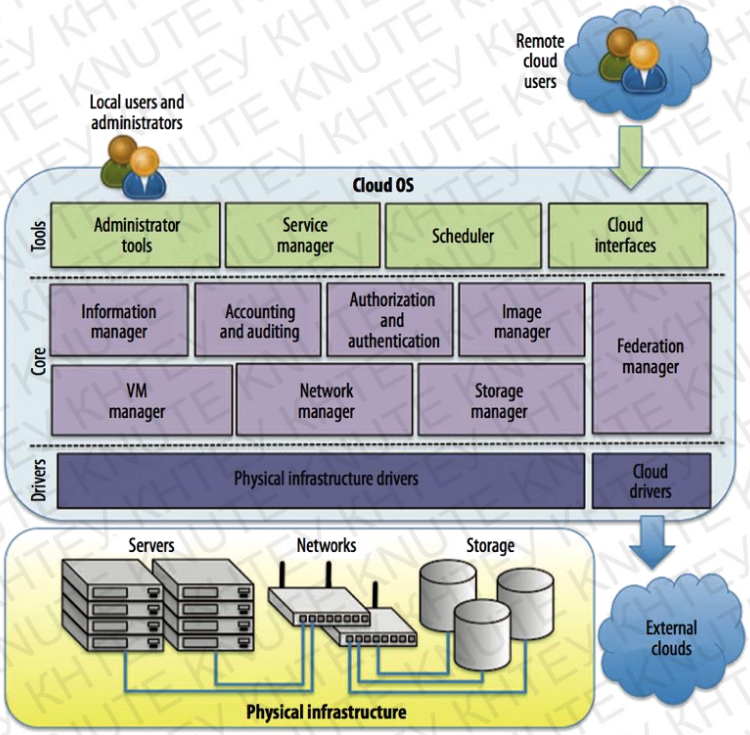

Рис. 2.1 Архітектура IaaS

Більшості користувачів знати повну архітектуру і не потрібно, але все-таки розглянемо основні моменти. Почнемо знизу вгору. Є якась фізична інфраструктура, що складається з серверів (Servers), мережевого обладнання (Networks) і пристроїв зберігання (Storage). На цій фізичної інфраструктурі виконується хмарна операційна система.

Як завжди, найнижчий рівень будь-якої операційної системи - це драйвери для взаємодії з залізом. Такі драйвери є і у хмарної ОС - драйвери фізичної інфраструктури (Physical infrastructure drivers). Є і драйвери хмари (cloud drivers) - вони потрібні для з'єднання з іншими, зовнішніми хмарами (external clouds).

Ядро хмарної ОС - всякі диспетчери. Є диспетчер віртуальних машин (VM manager), диспетчер мережі (Network manager), диспетчер сховища

(Storage manager) і т. Д. Кожен з диспетчерів відповідає за свою частину операційної системи.

На вершині системи - різні засоби управління - кошти адміністратора (administrator tools), диспетчер служб (service manager), планувальник (scheduler), інтерфейси хмари (cloud interfaces). Саме через інтерфейси хмари до нього підключаються користувачі хмари. Інтерфейси можуть бути самими різними, але останнім часом найбільш часто використовується веб-інтерфейс.

В основі застарілих IT, хостингу, керованого постачальника послуг (MSP) та хмар є загальноприйнятими будівельними елементами, які включають технології мереж, обробки та зберігання.

Різні типи серверів, мереж та технологій зберігання відповідають різним вимогам до хмарних обчислень та хмарного сховища (наприклад, щільні стійкові та блейд-сервери з різною кількістю сокетів і ядер з різною швидкістю ГГц, потоки, обсяг пам'яті та можливості розширення вводу-виводу) . Параметри мережі включають швидкі 40GbE та 100GbE для зворотних та магістральних схем, а також більш поширені 10GbE та 1GbE для віртуальних приватних мереж (VPN) та оптимізацію пропускної здатності.

Параметри або рівні зберігання даних включають надшвидкі твердотільні накопичувачі, а також швидкі жорсткі диски середньої та великої ємності. Функції управління зберіганням даних включають захист даних - висока доступність (HA), резервне копіювання (BC) та відновлення після аварій (DR) - а також зменшення розміру (DFR) для оптимізації простору, таких як стиснення, дедуплікація та тонка підготовка, що дозволяє більше інформація, яка зберігатиметься довше за менших витрат.

Програмні засоби також дуже важливі при створенні служб і рішень і включають API, проміжне програмне забезпечення, бази даних, додатки, гіпервізори для створення віртуальних машин (VM) та інфраструктури віртуальних робочих столів (VDI), а також хмарні стекові програми, такі як OpenStack, та пов'язане з ними управління інструменти. Приклади віртуальних машин та гіпервізорів VDI включають Citrix / Xen, KVM, Microsoft Hyper-V, Oracle та VMware ESX / vSphere.

У всіх трьох випадках зберігання даних налаштовується на системи зберігання, пристрої зберігання та обчислювальні сервери.

Громадські хмари - це послуги, доступні безкоштовно та за окрему плату, що надають різні функції, такі як Amazon Web Services (AWS), Google Docs або програмне забезпечення для резервного копіювання даних Seagate® EVault®. Громадські хмари контролюються їх власниками, чиї клієнти вирішили користуватися їхніми послугами. З іншого боку, приватні хмари належать або експлуатуються та контролюються організаціями і подібні до застарілих послуг IT-послуг. Однак зауважте, що приватні хмари, побудовані з використанням загальнодоступних компонентів або служб, та існуючі за межами місця в різних місцях провайдерів хмар, називаються гібридними хмарами.

2.5. Підходи побудови відмовостійкої системи

Є два принципово різних підходи, хоча вони цілком можуть незалежно або Напівзалежна поєднуватися для системи. Один підхід - це DR (disaster recovery), коли повна копія всієї системи може бути піднята в іншому датацентрі. Даний метод виручає практично в будь-якій ситуації, проте, він може має дуже тривалий проміжок простою. Але тут теж треба

бути дуже акуратним, не тільки конфігурація системи, але і залізо в DR повинні збігатися з продакшеном.

Другий спосіб - це програмно реалізовувати відмовостійкість для кожного з компонент і їх взаємодії. Різних технік зробити систему більш стійкою дуже багато. Давайте розглянемо деякі з них.

Low level fault tolerant services:

Принцип тут такий - система повинна складатися з більш-менш незалежних систем кожна з яких повинна бути відмовостійкої. В ідеалі не вигадувати велосипед, а використовувати вже готові рішення. Це насправді найкращий варіант, коли вся система буде спиратися на так звані low level fault tolerant services.

Single point of failure:

Уникайте архітектури, в якій вся система валитися при падінні одного з компонент. Досягти це можна або використовуючи принцип надмірності, або робити компоненти по можливості незалежними, щоб при падінні одного з компонент, переставала працювати тільки частина функціональності, а решта частини системи продовжували працювати. Звичайно таке рішення не підходить для основного функціоналу системи, але якщо при проблемах в яких нибудь особливостях системи, що дають допоміжні функції, трапилася проблема, і це вимкнуло всю систему цілком, то, напевно, це не дуже добре.

Надлишковість(Redundancy):

Коли в системі присутній надмірна кількість необхідних компонент. І при падінні одного з таких надлишкових компонент, все повинно продовжувати працювати. При такому підході проектування можна виділити дві стратегії: active-active і active-passive.

Active-Active:

Це означає, що робота ведеться одночасно з двома ідентичними компонентами. Наприклад в системі, де клієнт отримує ціни котирувань, він може підписатися відразу в два різних компонента і отримувати ціни одночасно з двох місць. Якщо один з таких компонентів впаде, то клієнт взагалі не помітить, що в системі трапилася якась проблема, що є безсумнівним плюсом даного підходу. Як мінусів можна виділити, що подвоюється кількість трафіку і час на його обробку, а так само додаткова серверна інфраструктура, яка постійно знаходиться в робочому режимі і споживає ресурси. Ще даний підхід буває неможливо реалізувати через різні бізнес обмежень. Наприклад трейдінговая система не може посилати одночасно один і той же ордер в два незалежних коннектора до біржі, так як в разі успіх на біржі з'явитися відразу два ордери, а це абсолютно неприпустимо.

Active-Passive:

Дана стратегія, має тільки один постійно робочий компонент, в разі падіння якого, автоматично піднімається другий, відновлює стан і бере на себе всю роботу. Наприклад така функціональність є в TIBCO EMS. Якщо включається дана опція, то робочий екземпляр EMS постійно моніториться, а в разі відмови запускається другий примірник EMS, відправляє всі непрочитані повідомлення і починає приймати нові. Але тут дуже важливо розуміти, чим за це доведеться заплатити. В одному з проектів, де ми включали цю опцію, максимальна пропускна здатність даного компонента впала приблизно в два рази.

Так само важливо розуміти, що якщо не використовується готове рішення, то його acttive-passive реалізація швидше за все зажадає більше зусиль, ніж active-active solution, так як тут потрібен надійний спосіб перевірки, що активний компонент функціонує, і потрібно вміти відновлювати стан на момент падіння або постійно його синхронізувати.

Ще дане рішення завжди буде мати якусь затримку в роботі при падінні компонента. Зате пасивний компонент швидше за все буде споживати багато менше ресурсів і можете отримати до поломок рішення на більш слабкому залозі.

Балансування навантаження (Load Balancing):

Зазвичай цей термін спливає при побудові високонавантажених систем. Однак для відмовостійких систем даний принцип теж можна застосувати. Ідея в тому, що є кілька ідентичних компонентів, між якими більш-менш рівномірно розподіляється вся навантаження. На відміну від вищеописаної active-active стратегії, тут кожну задачу виконує тільки один компонент. Даний механізм ідеально підходить для stateless компонент, інакше для відмовостійкості постійно доведеться синхронізувати стан. Наприклад в разі веб-серверів - робити реплікацію сесії. У даному рішенні дуже важливо мати як мінімум N + 1 redundancy, тобто якщо для пікових навантажень вам необхідно N працюють на всю котушку компонент, то в вашій системі має бути присутнім N + 1 таких компонент, так як інакше, якщо у вас вони впали з елементів, то на всі інші навантаження збільшитися і вся ваша система завалиться як доміно.

Найпоширеніший приклад load balancing - це, напевно, балансування навантаження веб-серверів. Тут бувають як програмні рішення (Nginx), так і спеціальні залізяки. У backend системах load-balancing часто використовують для реалізації балансування реалізують за допомогою JMS черги, вішаючи на НЕ неї кілька ідентичних слухачів. В даному випадку чергу гарантує, що повідомлення потрапить тільки в один з компонентів, і поки він його не обробить, наступне повідомлення він взяти не зможе. У такій же системі, якщо в якості черзі взяти topic, то можна реалізувати active-active систему.

Захисне програмування (Defencive coding):

Щоб додаток було максимально ВІДМОВОСТІЙКО, треба не тільки думати про це під час дизайну, а й у програмування. Потрібно завжди думати, а що якщо трапитися те-то і писати код обробляє всі такі непередбачені ситуації. Наведу кілька прикладів.

Infinite Loop. Слідкуйте за тим, щоб при виникнення помилки під час обробки повідомлення, не виходили в нескінченний цикл, який постійно намагається виконати дану операцію, особливо якщо повідомлення приходить вам ззовні, і ви не можете гарантувати, що воно коректно. Я вже ні один раз бачив, як з-за цієї помилки підвисає вся система. Що ж в цьому випадку робити? Наприклад, можна відправляти повідомлення про помилку в систему моніторингу (див. Нижче) і переходити до обробки наступного таска.

Reconnection. Обов'язково пишіть логіку реконнекта, особливо в системах з наявністю firewall.

Degradate gracefully. Throttling. Якщо ваш додаток отримує ззовні повідомлення, то обов'язково подумайте, що будете робити, якщо буде приходити повідомлень багато більше, ніж можна обробити. Наприклад, можете починати реджектіть запити або емулювати повільний коннекшен, сповільнюючи відсилання відповіді.

State Management. Спробуйте зробити так, щоб ваша система час від часу зберігала свій стан (checkpoint), щоб у разі великих проблем ви завжди могли відкотитися в останні консистентні стан. Наприклад, в FIX протоколі, поширеному в фінансовихдодатках, є поняття sequence. Кожне повідомлення має свій порядковий номер, сервер запам'ятовує номер останнього відісланого повідомлення, а клієнт запам'ятовує номер останнього прийнятого повідомлення. На рестарт вони обмінюються даними номерами, і якщо між ними є проміжок, то сервер висилає всі пропущені повідомлення.

InfrastructureError. Якщо в додатку сталася помилка, з якою не можете відновитися, наприклад OutOfMemoryError в java, то можна спробувати зупинити додаток і запустити його заново. Це можна автоматизувати за допомогою такої тулзи як Tanuki Java Service Wrapper, про даної функції якого докладно описано тут.

NullPointerException. Однак, не треба божеволіти, перевіряючи кожен вхідний параметр методу або конструктора на null. Краще прийміть за правило: ніколи не передавати в системі null між компонентами.

2.6. Висновки до розділу 2

Отже, підсумовуючи вище викладене, можна визначити, що вбудований в Visual Studio інструментарій Azure дозволяє полегшити процес розробки проекту. Що дозволяє прискорити розробку хмарних застосунків.

Також було розгялнуто декілка підходів в побудові відмовостійкої системи.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ХМАРНОЇ СИСТЕМИ

3.1. Мікросерверна архітектура

Безліч додатків, з якими ми регулярно перетинаємося (інтернет-банки, розважальні сервіси типу YouTube і так далі), часто створені з використанням безлічі технологій, які якось уживаються під одним дахом і не виглядають розрізнено.

Мікросервери - це архітектурний шаблон. Всі сервіси в цьому шаблоні:

- Маленькі
- Сервіс не повинен вимагати великої кількості людей для розробки. Одна команда може розробляти кілька сервісів.
- Сфокусовані
- Один сервіс - одна задача.
- Слабкозв'язаного
- Зміни в одному сервісі не впливають на інший.

Высокосогласованние

Компонент або клас створюються з урахуванням всіх методів вирішення бізнес-завдання.

Класичне монолітне додаток зазвичай має стандартну структуру Інтерфейс -> Бізнес-логіка -> Дані.

Один сервіс повинен вирішувати одне завдання і ці завдання визначаються частинами відповідальності додатки. Наприклад, можуть існувати різні сервіси для команд, що працюють в рамках одного проекту (припустимо, онлайн-магазину).

Зазвичай Мікросерверна архітектура застосовується як один з варіантів масштабування додатки. Всього таких варіантів може бути три:

Sharding («розбиття» або просто «шардінг») - дані і інструменти для доступу до них розміщуються на різних вузлах.

Mirroring (створення дзеркал) - дублювання всіх даних по безлічі однакових вузлів.

Власне, Мікросервери - функціональність розбита по бізнес-завдань, кожен сервіс може бути створений своїми засобами розробки.

Плюси і мінуси Мікросерверів

Мікросерверний підхід до проектування додатків - не найкращий вибір і не гірший, а один з. Вирішувати, чи будувати додаток з безлічі незв'язаних частин, потрібно з огляду на плюси і мінуси цього підходу.

Чіткий розподіл по модулях. Завжди буде зрозуміло, як працює та чи інша частина коду. Просто додавати нові функції.

Висока доступність. Якщо якась частина моноліту зламається - зламається все додаток. З Мікросерверамі інакше: сервіси можуть працювати не всі (не критичною, на кшталт авторизації), але додаток при цьому залишиться доступним.

Різноманітні технології. При розробці кожного сервісу ви вільні вибирати інструменти, які найкраще підійдуть для конкретної бізнес-логіки в цьому сервісі. Наприклад, вибрати оптимальну базу даних і зручні інструменти для роботи з нею. Мікросерверная архітектура також дозволяє

спробувати якусь нову технологію на окремому сервісі, що не переписуючи при цьому все додаток.

Відносна простота розгортання. Кожен сервіс піднімається самостійно, що робить процес розгортання і налагодження більш чистим.

Складність розробки. Якщо вам потрібно швидке рішення (прототип, невеликий додаток, стислі терміни), то Мікросервери вам не підійдуть. Швидкість розробки - висока плата за доступність і модульність.

Складність підтримки. Кожен Мікросервер потребує окремого обслуговуванні, тому потрібен постійний автоматичний моніторинг.о

3.2. Моделювання хмарної системи

Завдання, що відправляються користувачами через інтерфейс хмарної системи, поміщаються в глобальну чергу, підтримувану центральним хмарним диспетчером. Даний диспетчер, реалізуючи закладений в нього алгоритм планування, приймає рішення про призначення завдань на обчислювальні кластери, їх конкретні обчислювальні вузли (виділяє частину для multi-instance).

Кожен обчислювальний кластер не використовує локальний планувальник завдань, і його обчислювальні ресурси повністю виділені в розпорядження хмарного диспетчера. Таким чином, вузли всіх відчужених кластерів утворюють єдиний обчислювальний пул.

Система складається з диспетчера і обчислювальних кластерів CL1, CL2, ..., CLn. i-й кластер має обчислювальні вузли P1, i, P2, i, ..., Pm, і Джерело I генерує потік паралельних завдань, що відправляються користувачами в чергу Q диспетчера □. Канал S являє собою планувальник, який відповідно до закладеного в нього алгоритмом здійснює вилучення

завдань з черги Q і призначення їх на вільні обчислювальні ядра відповідних по конфігурації вузлів обчислювальних кластерів.

Кожен обчислювальний вузол характеризується кількістю процесорів і ядер, розміром пам'яті, розміром вільного сховища і ціною обчислень. Обчислювальний кластер характеризується сукупністю будь-яких обчислювальних вузлів.

Завдання описується параметрами тривалості очікування і виконання, часом приходу, запуску інтенсивності використання ресурсів і вимогами до ресурсів. Також у завдання є пріоритет і матриця зв'язності з зовнішніми ресурсами. Особливістю хмарних завдань є можливість міграції завдань між вузлами і ЦОД, а також нескінченне (в межах моделі) час виконання.

Передача даних між вузлами в хмарах заснована на принципі окремого мережевого сховища даних і виділених сервісах обміну інформацією. Різні екземпляри додатків для доступу до даних не використовують зв'язок один з одним. Такий зв'язок використовується тільки для синхронізації даних і для управління процесами. Потік передачі даних характеризується кількістю даних і матрицею зв'язності, а також статистичними законом розподілу часу між пакетами даних і параметрами цього закону.

Модель хмарної системи складається з моделей ресурсів (з диспетчером і чергами), вузлів, завдань і передачі даних. Сукупність моделей повинні описувати конкретну послідовність виконуваних дій (конкретний випадок використання хмари).

3.3. Висновок до розділу 3

Отже, хмарна система може бути використна для користувача навіть на основі мікросервісної архітекрути не затронучи дуже великі масштаби розробки, та не використовуючи багато людсських ресурсів.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Проведено аналіз слабких місць хмарних систем та вибрано архітектуру застосунку, яка є оптимальною для реалізації поставлених завдань.

2. Проведено аналіз відмовостійкості систем та вибрано найкращу для хмарної системи.

3. Проведений аналіз засобів розробки показав, що найзручнішим для реалізації поставлених завдань є VisualStudio з Azure.

4. В процесі виконання магістерської роботи розроблено хмарну систему для більш простого використання хмарних технологій.

5. Розроблена система повністю задовольняє поставлені функціональні вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Види хмар [Електронний ресурс] // Softline Company. - Режим доступу : http://www.softline.kiev.ua/ru/oblachnye-tekhnologii/vidy-oblakov.html.
2. Моделі сервісів хмарних вичислення. Ч. 2: Платформа як сервіс [Електронний ресурс] // IBM. Developer Works. - Режим доступу: http://www.ibm.com/developerworks/ru/library/cl-cloudservices2paas/.
3. Побудова приватних хмар [Електронний ресурс] // Parking.ru. - Режим доступу: http://www.parking.ru/private/private_cloud/ .
4. Рудковський О. Які ж насправді переваги надає «хмара» із десктопними додатками? [Електронний ресурс] / О.Рудковський. – Режим доступу: http://www.business.ua/opinions/khmarn_tekhnolog_dlya_b_znesu-268935/
5. Хмарні технології [Електронний ресурс] // Intecracy group. – Режим доступу : http://www.intecracy.ua/ua/poslugi/khmarni-tekhnologiji.html.
6. Що таке SaaS [Електронний ресурс] // Live Business. – Режим доступу: http://www.livebusiness.ru/faq/saas/ .
7. IaaS – Infrastructure as a Service. IT-інфраструктура в облаці [Електронний ресурс] // Rentacloud. – Режим доступу : http://www.rentacloud.su/iaaseurope .
8. Cloud Computing and Cloud Storage Architectures [Електронний ресурс] – Режим доступу : https://www.seagate.com/gb/en/tech-insights/cloud-compute-and-cloud-storage-architecture-master-ti/

					КНТЕУ 121 06м-09.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		*19.10.20*	Застосування мікросерверної архітектури для побудування відмовостійкої хмарної системи	*Стадія*	*Арку*	*Арку*
Керівник		Десятко А. М.		*19.10.20*		*СВД*	*36*	*36*
Гарант		Криворучко О.В.		*19.10.20*		*Факультет інформаційних технологій 2м курс, 6м група*		
Розробив		Журбенко М.О.		*19.10.20*	*Список використаних джерел*			