

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЇВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка багаторівневого веб-застосунку на базі Azure

**Текстова частина до курсової роботи
за спеціальністю „Комп’ютерні науки”**

Керівник курсової роботи
кандидат техн. наук
Черкасов Д.І.

_____ (підпис)
“ ____ ” _____ 2020 р.

Виконав студент Благов В. С.
“ ____ ” _____ 2020 р.

Київ 2021

Зміст

Вступ	5
1. Огляд існуючих рішень	7
2. Структурна розробка власного рішення.....	11
2.1 Структурна схема.....	11
2.2 Перелік компонентів, їх характеристики, опис	12
2.3 Опис функціонування системи	14
2.4 Функціональне призначення.....	14
3. Детальна розробка компонента.....	15
3.1 Розгортання додатку на платформі Azure	15
3.2 Розробка додатку	20
Висновки	24
Список джерел.....	25
Додатки	26

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,
доктор техн. наук, декан ФІ А. М. Глибовець

(підпис)
“ ____ ” _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Благову Владиславу

4-го курсу факультету інформатики

ТЕМА: Розробка багаторівневого веб-застосунку на базі Azure

Вихідні дані

- Багаторівневий застосунок на базі Azure

Зміст ТЧ до курсової роботи:

1. Вступ
2. Огляд існуючих рішень
3. Структурна розробка власного рішення
4. Детальна розробка компонента
5. Висновки
6. Список джерел
7. Додатки (за необхідністю)

Дата видачі “ ____ ” _____ 201_ р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка багаторівневого веб-застосунку на базі Azure

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2020 р.	
2.	Огляд літератури за темою роботи	листопад - грудень 2020 р.	
3.	Оволодіння навичками розробки вільних від блокувань структур даних	грудень - лютий 2020 р.	
4.	Створення практичної частини роботи	лютий - березень 2021 р.	
5.	Написання пояснювальної роботи	березень 2021 р.	
6.	Створення слайдів для доповіді та написання доповіді	березень 2021 р.	
7.	Надання роботи керівнику для перевірки	березень - квітень 2021 р.	
8.	Корегування роботи за результатами перевірки керівником	квітень 2021 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів	квітень 2021 р.	
10.	Захист курсової роботи	12 квітня 2021 р.	

Студент _____ Благон В.С. _____

Керівник _____ Черкасов Д. І. _____

“ _____ ” _____ р.

Вступ

Що таке хмарні платформи? Простіше кажучи, хмарні платформи надають користувачам різноманітні обчислювальні послуги, включаючи сервери, сховища, бази даних, програмне забезпечення та аналітику. Тим самим вони пропонують вже деякі готові потреби для бізнесу, гнучкі ресурси та економічне та легке масштабування застосунків. Коли ви користуєтесь послугами цих платформ, при правильному налаштуванні платите лише за ту кількість ресурсів, скільки ви використовуєте. Тут відразу можна побачити велику економічну перевагу у порівнянні з розміщенням та обслуговуванням власних серверів. Ефективне керування інфраструктурою, а також масштабування у мірі зміни потреб вашого бізнесу додає значні переваги у економії ресурсів бізнесу. Ви завжди використовуєте хмарні платформи, навіть якщо цього не усвідомлюєте, наприклад, при надсиланні електронного листа, перегляду фільмів або серіалів у мережі, прослуховуванні музики, граючи в онлайн ігри або ж при зберіганні власних фотографій та інших документів. Під час розробки ПЗ такі платформи пропонують наступне:

1. Швидке та легке розгортання та масштабування програм – веб, мобільних або ж API сервісів.
2. Тестування власних додатків.
3. Зберігання, резервне копіювання та відновлення даних.
4. Аналізування даних за рахунок хмарних сервісів, такі як машинне навчання та штучний інтелект.
5. Використання інтелектуальних моделей, щоб допомогти залучити клієнтів та надати цінну інформацію стосовно збереження та обробки даних.

Дану роботу присвячено для дослідження таких специфікацій хмарної платформи:

- Гнучка масштабованість в залежності від рівня навантаження

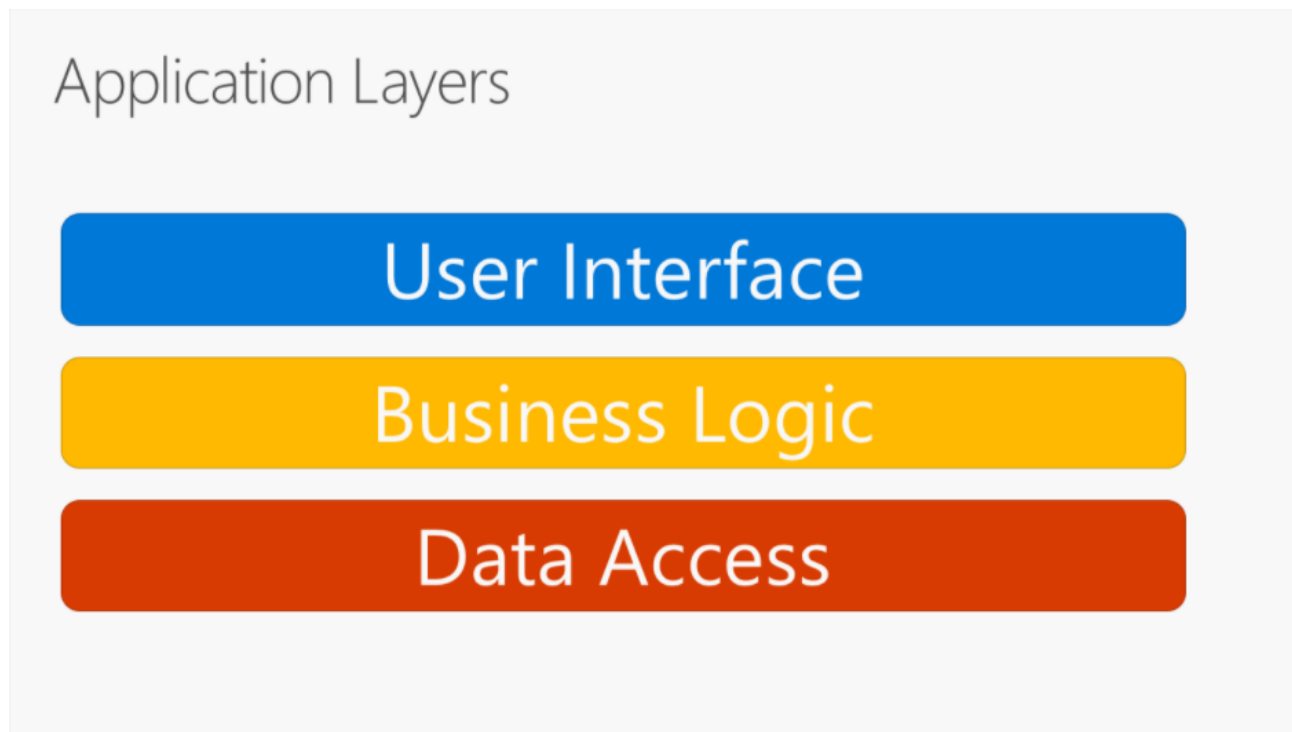
- Відмовостійкість за рахунок надлишковості і “самолікування”
- Прив’язка до мережевої хмарної інфраструктури: Geographical Regions, Availability Zones
- Використання хмарних функціональних компонентів: Load Balancers, Virtual Machine Instances

На сьогоднішній день усі компанії, від гігантів до початкових, використовують хмарні платформи для розміщення, розробки та тестування свого програмного забезпечення і знання їх специфіки та функціоналу дає спеціалісту велику перевагу на ринку праці, а тому й дана робота носить актуальний характер.

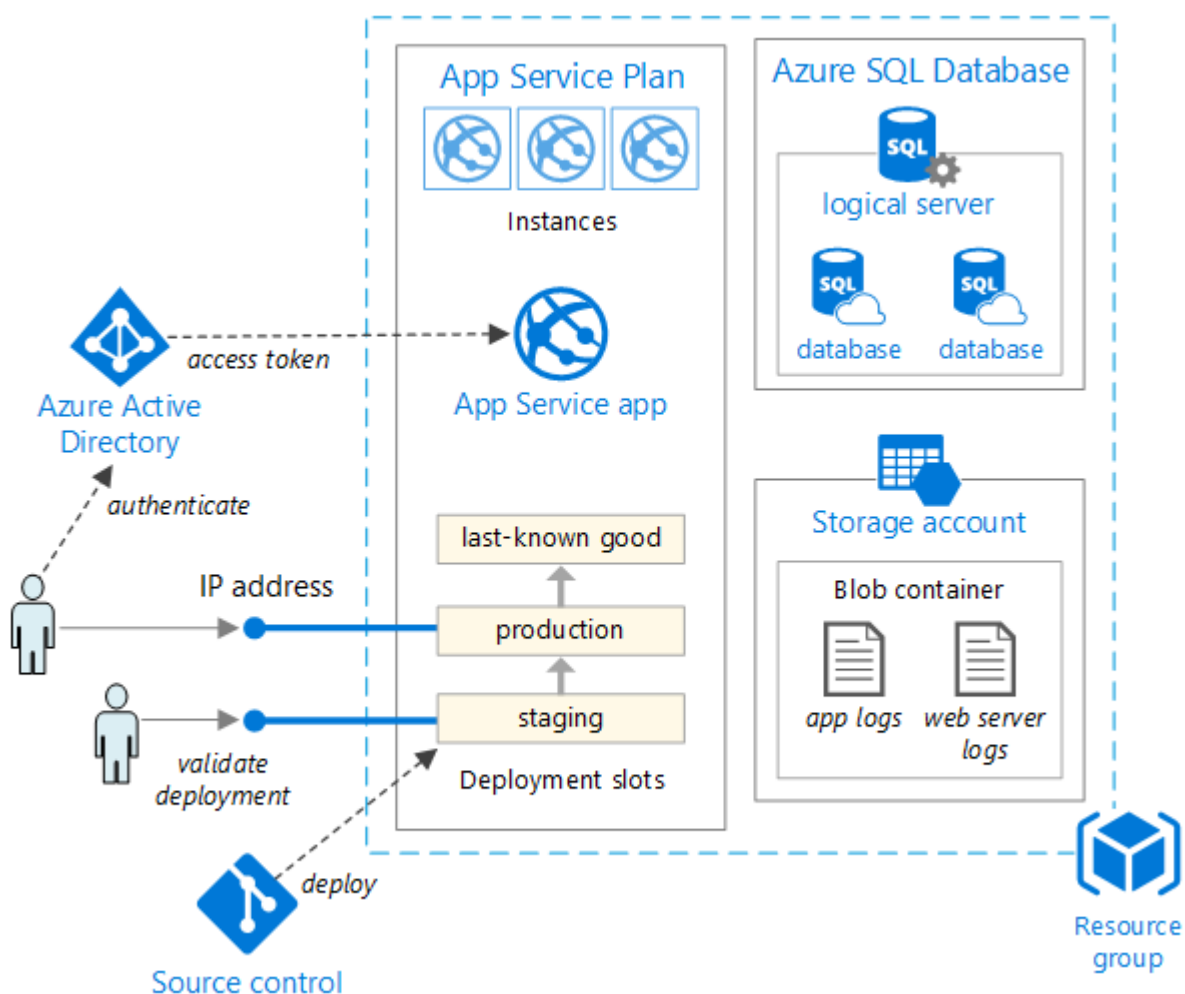
1. Огляд існуючих рішень

Порівняння існуючих рішень на платформі Azure

Традиційні додатки з N-шарової архітектурою

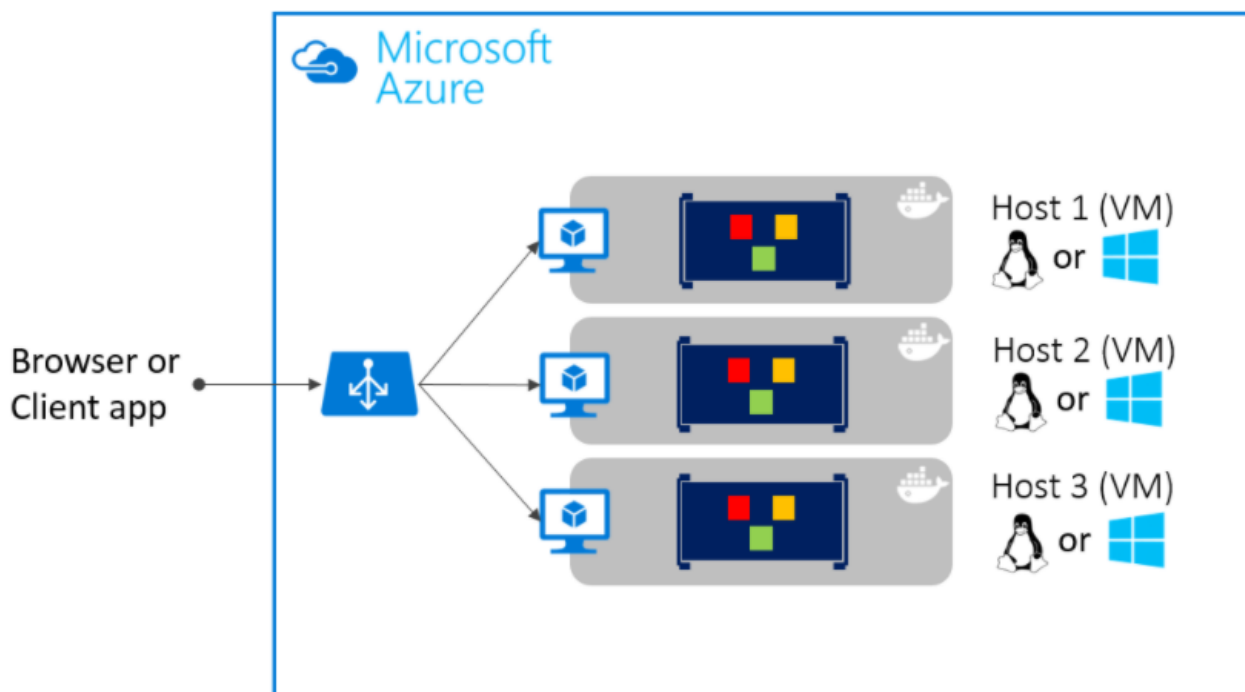


Як правило, у додатку визначаються шари призначеного для користувача інтерфейсу, бізнес-логіки і доступу до даних. У рамках такої архітектури користувачі виконують запити через шар призначеного для користувача інтерфейсу, який взаємодіє тільки з шаром бізнес-логіки. Шар бізнес-логіки, в свою чергу, може викликати шар доступу до даних для обробки запитів. Шар призначеного для користувача інтерфейсу не повинен виконувати запити безпосередньо до шару доступу до даних і будь-якими іншими способами безпосередньо взаємодіяти з функціями зберігання. Аналогічним чином, шар бізнес-логіки повинен взаємодіяти з функціями зберігання тільки через шар доступу до даних. Таким чином, для кожного шару чітко визначена своя обов'язок. Сама архітектура застосунку може виглядати наступним чином



Така архітектура підтримує вертикальне та горизонтальне масштабування, що дозволяє використовувати переваги хмарного масштабування за запитом. Під вертикальним масштабуванням розуміється збільшення числа ЦП, обсягу пам'яті, місця на диску і інших ресурсів на серверах, де розміщується додаток. Горизонтальне масштабування полягає в додаванні додаткових примірників таких фізичних серверів, віртуальних машин або контейнерів. Якщо додаток розміщується на декількох примірниках, для розподілу запитів між екземплярами програми використовується система балансування навантаження.

Монолітні додатки і контейнери



Монолітний додаток повністю замкнутий в контексті поведінки. Під час роботи воно може взаємодіяти з іншими службами або сховищами даних, проте основа його поведінки реалізується у власному процесі, а весь додаток зазвичай розгортається як один елемент. Для горизонтального масштабування такий додаток зазвичай цілком дублюється на декількох серверах або віртуальних машинах.

Монолітні додатки в Microsoft Azure можна розгорнути з використанням виділених віртуальних машин для кожного екземпляра. За допомогою масштабованих наборів віртуальних машин Azure можна легко масштабувати віртуальні машини. Служби додатків Azure також можуть виконувати монолітні додатки і легко масштабувати екземпляри, тому вам не доведеться керувати віртуальними машинами. Служби додатків Azure також можуть виконувати окремі екземпляри контейнерів Docker, спрощуючи розгортання. За допомогою Docker ви можете розгорнути одну віртуальну машину на вузлі Docker і виконувати на ній кілька примірників. Для управління масштабуванням можна використовувати систему балансування Azure як показано на рис. 5-14.

Розгортанням на різних вузлах можна керувати за допомогою традиційних методів розгортання. Вузлами Docker можна керувати за допомогою введення вручну команд виду `docker run` або автоматизовано, наприклад, за допомогою конвеєрів безперервної поставки (CD).

1) Порівняльний аналіз

Монолітний додаток

Стосовно монолітного підходу стає очевидним, коли додаток розростається і його необхідно масштабувати. Якщо масштабується додаток цілком - все вийде. Але в більшості випадків необхідно масштабувати всього кілька частин додатка, поки інші компоненти працюють нормально.

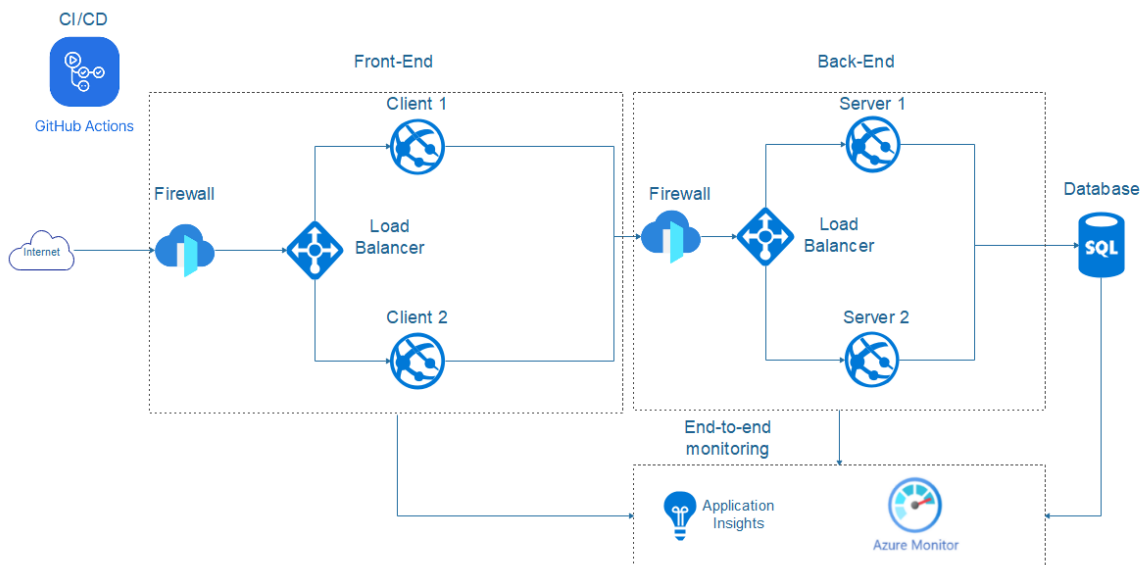
Монолітний підхід знайшов широке поширення і використовується багатьма організаціями при розробці архітектури. У багатьох випадках це дозволяє домогтися бажаних результатів, однак іноді організація зіштовхується з обмеженнями. У багатьох організаціях додатки будувалися за такою моделлю, оскільки кілька років тому за допомогою існуючих інструментів та інфраструктури надто складно було створювати архітектури, орієнтовані на служби (SOA), і проблем не виникало, поки програма не починала розростатися. Якщо ваша організація зіткнулася з обмеженнями монолітного підходу, наступним логічним кроком може стати розбиття додатку для більш ефективного використання контейнерів і мікрослужб.

Багатошаровий додаток

Одним з недоліків традиційного багатошарового підходу є те, що обробка залежностей під час компіляції здійснюється зверху вниз. Це означає, що шар призначеного для користувача інтерфейсу залежить від шару бізнес-логіки, який, в свою чергу, залежить від шару доступу до даних. Це означає, що шар бізнес-логіки, який зазвичай містить ключові функції програми, залежить від

деталей реалізації доступу до даних (і часто від наявності самої бази даних). Тестування бізнес-логіки в такій архітектурі часто утруднено і вимагає наявності тестової бази даних. Для вирішення цієї проблеми може застосовуватися принцип інверсії залежностей. [1]

2. Структурна розробка власного рішення



2.1 Структурна схема

- Інтерфейс користувача -- Front-end
- Client-1 та Client-2 сервіси, що відповідають за оптимальний та швидкий показ контенту на сторінці користувача.
- Серверна частина -- Back-end
- Server-1 та Server-2 сервіси, що відповідають за усю бізнес логіку додатку, отримують та оброблюють запити зі сторони клієнта.
- Балансувальники навантаження -- load balancers
- База даних SQL
- Захисні фільтри – firewalls (Azure Front Door)
- Компоненти моніторингу

- GitHub Actions

2.2 Перелік компонентів, їх характеристики, опис

1) Інтерфейс користувача

Інтерфейс користувача буде написаний окремим додатком на мові Javascript з підключенням фреймворку React

2) Серверна частина

Серверна частина буде написана окремим додатком на мові C# з підключенням реляційної бази даних SQL Azure. База даних SQL використовує свою базу коду спільно з ядром СУБД Microsoft SQL Server.

3) Load balancers

У рамках віртуальної мережі Azure Load Balancer перенаправляє клієнтів, зв'язавши з кластером маршрутизаційну приватну IP-адресу.

4) Firewalls

Брандмауер потрібен, щоб управляти доступом до додатків та ресурсів і реєструвати відомості про нього в журналах. Брандмауер Azure підтримує фільтрацію вхідного і вихідного трафіку, а також внутрішніх периферійних підключень і гібридних підключень через шлюзи Azure VPN і ExpressRoute. [2]

5) Azure Monitor

Azure Monitor допомагає зібрати та проаналізувати дані телеметрії, що надходять до Azure і локального середовища, і приймати відповідні заходи. Azure Monitor за лічені секунди допомагає збільшити до максимуму продуктивність і доступність додатків, а також забезпечує завчасне виявлення проблем. [3]

6) GitHub Actions

GitHub Actions дозволяє безперервно створювати, тестувати та розгортати застосунки на будь-якій платформі та на будь-якій хмарі на таких мовах

як Node.js, Python, Java, PHP, Ruby, C/C++, .NET, Android и iOS. Він забезпечує швидкі конвеєри безперервних інтеграцій і поставки (CI / CD) для кожного проекту з відкритим кодом. [\[4\]](#)

2.3 Опис функціонування системи

- 1) Усі запити, зроблені на інтерфейсі користувача, відправляються на сервер
- 2) Балансувальник приймає вхідний трафік від користувача та відправляє запит на сервер, спираючись на завантаженість серверів.
- 3) За допомогою Баз даних SQL Azure виконується відправка телеметрії
- 4) Azure Monitor збирає та аналізує метрики інфраструктури і квоти
- 5) Розробники та адміністратори можуть продивлятися дані щодо працездатності системи, продуктивності та використання.

2.4 Функціональне призначення


Оформлення наукових джерел є однією з вимог, на яку користувачі витрачають багато часу. Після ґрунтовного аналізу застосунків для автоматизації оформлень наукових джерел (відповідно до вимог Вищої атестаційної комісії (ВАК) України) було прийнято рішення розробити повністю безкоштовний сайт для виконання цієї задачі з можливістю отримання історії попередніх оформлень. Сайт призначений для швидкого оформлення наукових джерел відповідно до вимог та проходження нормоконтролю при написанні Ваших публікацій, курсових, дипломних, дисертацій та інших наукових робіт.

3. Детальна розробка компонента

3.1 Розгортання додатку на платформі Azure

Спочатку була розгорнута база даних на платформі Azure з базовою конфігурацією – 10 DTUs (об'єм обчислювальних ресурсів), 250 GB

Basics • Networking Security Additional settings Tags Review + create

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#) 

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ ▼

Resource group * ⓘ ▼

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name *

✖ A database with the same name already exists on this server.

Server * ⓘ ▼

[Create new](#)

Want to use SQL elastic pool? * ⓘ ☐ Yes ☒ No

Compute + storage * ⓘ

Standard S0
10 DTUs, 250 GB storage
[Configure database](#)

Далі був розгорнутий веб-сервіс з операційною системою Windows зі стеком часу виконання .NET Core 2.1 (LTS). Даний сервіс буде відповідати за усю бізнес-логіку застосунку, котрий буде приймати та оброблювати усі запити з клієнтського сервісу.

Сведения о проекте

Выберите подписку для управления развернутыми ресурсами и затратами. Используйте группы ресурсов, такие как папки, для организации всех ресурсов и управления ими.

Подписка * ⓘ

Azure subscription 1

Группа ресурсов * ⓘ

VakSight

[Создать](#)

Сведения об экземпляре

Имя *

vaksight-api

.azurewebsites.net

✖ Имя приложения vaksight-api недоступно.

Опубликовать *



Код



Контейнер Docker

Стек времени выполнения *

.NET Core 2.1 (LTS)

Операционная система *



Linux



Windows

Регион *

West Europe

ℹ Не можете найти свой план Службы приложений? Попробуйте изменить регион.

План службы приложений

Ценовая категория плана службы приложений определяет расположение, компоненты, стоимость и вычислительные ресурсы, связанные с приложением. [Дополнительные сведения](#)

План Windows (West Europe) * ⓘ

(Новое) ASP-VakSight-adc9

[Создать](#)

Номер SKU и размер *

Бесплатный F1

Общая инфраструктура, 1 ГБ памяти

Після вдалого розгортання сервісу необхідно налаштувати процедуру автоматизованого розгортання після кожного оновлення кодової бази. Для цього було використану службу GitHub Actions, оскільки наш код знаходиться на платформі GitHub. Якщо код знаходився би на платформі Bitbucket або GitLab, то можна використати Bitbucket Pipelines або ж GitLab Pipelines відповідно. Файл, котрий відповідає за процедуру автоматичного розгортання, знаходиться в додатку А.

Далі необхідно налаштувати масштабування сервісу. Існує два види масштабування сервісу – вертикальне та горизонтальне. У даній роботі виберемо горизонтальне. Створимо 2 екземпляри сервісу, оскільки у нашого сервісу повинно бути мінімум 2 екземпляри. Для автоматичного масштабування вкажемо максимальну кількість екземплярів 3. Якщо кількість екземплярів сервісу перевищує 1, то Azure самостійно створює Load Balancer, котрий буде розподіляти навантаження між усіма сервісами рівномірно. Для автоматичного масштабування необхідно задати правило, щоб система знала коли саме потрібно створювати нові екземпляри сервісу. Виберемо, що якщо навантаження на сервісі більше 70%, то треба створити новий екземпляр сервісу.

По умолчанию

Auto created scale condition

Предупреждение об удалении

Последнее или используемое по умолчанию правило повтора невозможно удалить. Вместо этого можно отключить функцию автомасштабирования.

Режим масштабирования

☒ Масштабировать на основе метрики

☐ Масштабировать до указанного числа экземпляров

Правила

Рекомендуется задать по меньшей мере одно правило горизонтального уменьшения масштаба. Чтобы создать правила, щелкните [Добавить правило](#).

Горизон...

Когда

ASP-VakSight-92b5

(Среднее) CpuPercentage > 70

Увеличить число на 1

+ Добавить правило

Ограничения экземпляров

Минимум ①

Максимальный ①

По умолчанию ①

2 ✓

3 ✓

2 ✓

Расписание

Данное условие масштабирования применяется, если ни одно другое условие масштабирования не выполняется.

Далі необхідно створити Firewall для балансування навантаження HTTP (s), захисту від атак DDoS і безпеки застосунку. Для цього необхідно лише підключити службу Azure Front Door до сервісу.



Служба Azure Front Door настроена для вашего веб-приложения.

[vaksight-api](#)

Далі про всяк випадок треба налаштувати резервне копіювання сервісу для швидкого відновлення у критичних ситуаціях. Під час створення резервного копіювання можна налаштувати період, коли повинні створюватися копії сервісу. У рамках цієї роботи виберемо створення резервних копій кожен тиждень.



Резервное копирование

Настройте создание восстанавливаемых архивных копий содержимого приложений, конфигурации и базы данных при архивации. [Дополнительные сведения](#)

Архивация настроена; запланированная архивация начата четверг, 8 апреля 2021 г., 17:09:06 GMT+3. Архивация будет производиться каждые 7 Дн.

Далі треба налаштувати перевірку працездатності сервісу. Health check підвищує доступність додатка, видаляючи непрацездатні екземпляри з підсистеми балансування навантаження. Якщо екземпляр залишається непрацездатним - він буде перезапущений. Обов'язково необхідно вказати період за який буде перевірятися.

Проверка работоспособности



Включить



Отключить

Путь *

Это путь, по которому мы будем проверять наличие неработоспособных экземпляров.

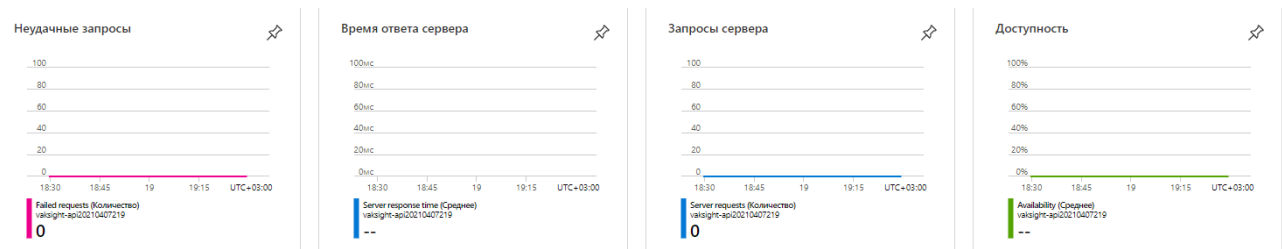
vaksight-api

Балансировка нагрузки

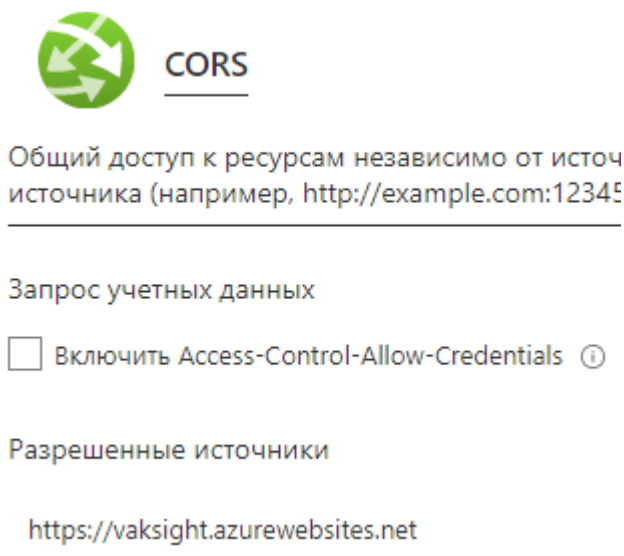
Настройте время, в течение которого неработоспособный экземпляр остается в подсистеме балансировки нагрузки перед удалением.

10 minutes

Для збору даних о моніторингу сервісу був підключений сервіс Application Insights. Цей потужний інструмент дозволяє з легкістю продивлятися усю інформацію про функціонування сервісу.



Наостанок для майбутньої взаємодії клієнтського застосунку з сервером необхідно налаштувати CORS (Cross-origin resource sharing). Загальний доступ до ресурсів незалежно від джерела (CORS) дозволяє виконувати код JavaScript в браузері на зовнішньому вузлі для взаємодії з внутрішнім сервером.



Для тестування серверу можна перейти за посиланням - <https://vaksight-api.azurewebsites.net/swagger>

Для розгортання клієнтського сервісу також було обрано створити веб-сервіс на операційній системі Linux. Процес розгортання клієнтського сервісу ідентичний до процесу розгортання сервісу бізнес-логіки. Налаштування резервного

копіювання, фаєрволу, автомасштабування та моніторингу зовсім ідентичний до процесу налаштування сервісу бізнес-логіки. Для CI/CD також обраний GitHub Actions. Файл з конфігураціями процедури автоматизованого розгортання знаходиться в додатку Б. Посилання на клієнтський сервіс -

<https://vaksight.azurewebsites.net/>

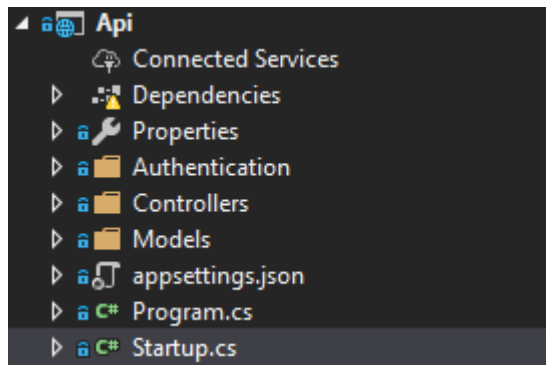
3.2 Розробка додатку

Клієнтська частина була написана за допомогою фреймворку React зі застосуванням компонентів. Компонент - це клас, який успадковується від класу `React.Component`. За допомогою функції «`render`» компонент повертає зображення, яке користувач отримує у себе у браузері. Також була реалізована навігація для переходу до окремих сторінок сайту.

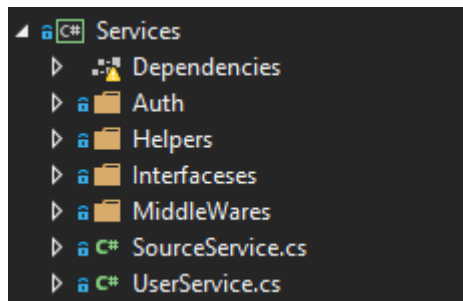
```
const App = () => (  
  <div>  
    <Router>  
      <Route exact path="/" component={Main}/>  
      <Route path="/history" component={History}/>  
      <Route path="/authMain" component={AuthMain}/>  
      <Route path="/signIn" component={SignIn}/>  
      <Route path="/signUp" component={SignUp}/>  
    </Router>  
    <Footer/>  
  </div>  
)
```

Серверна частина була написана на мові C# на основі фреймворку ASP.NET Core 2.2. Проект розбитий на 3 рівні:

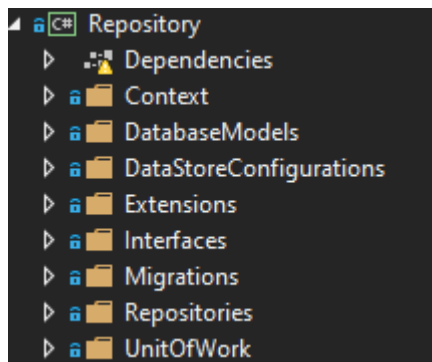
- API – 1 рівень, котрий відповідає за отримання, валідацію запитів та відправлення відповідей з відповідними статус-кодами.



- Service – 2 рівень, який відповідає за усю головну бізнес-логіку додатку (створення та форматування списків джерел, авторизація користувача)

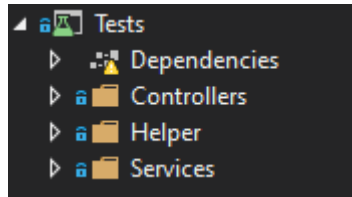


- Repository – 3 рівень, що відповідає за запити та вибірку з бази даних. Також на цьому рівні знаходяться міграції бази даних, оскільки для створення бази даних використовувався Code first підхід. За допомогою цих міграцій можна легко повернутися до минулого стану бази даних.



UnitOfWork - це патерн репозиторію, який використовується для інкапсулювання логіки роботи з базою даних. Цей патерн дозволяє спростити роботу з декількома репозиторіями за рахунок того, що всі репозиторії використовують один й той самий контекст даних.

- Тести проекту знаходяться в окремому проєкті. Проєкт призначений для тестування функціональності системи та використовує бібліотеку XUnit.



Оскільки проєкт створений за трьохрівневою архітектурою, то кожен рівень має можливість викликати функції та користуватися об'єктами нижчого рівня, але не вищого забезпечуючи інверсію залежностей.

Аутентифікація користувача зроблена за допомогою JWT-токенів. JWT-токен представляє собою веб-стандарт, який передає дані про користувача додатку у зашифрованому виді.

Усі конфігурації додатку зберігаються у файлі appsettings.json.

Entity Framework допомагає з роботою та налаштуванням бази даних.

Наприклад, конфігурація бази даних виглядає наступним чином

```
1 reference | 0 exceptions
public void Configure(EntityTypeBuilder<SourceRecord> builder)
{
    builder.ToTable("Sources");

    builder.HasKey(p => p.Id);

    builder.Property(p => p.Type).IsRequired();
    builder.Property(p => p.Content).IsRequired();

    builder.HasOne(p => p.User)
        .WithMany()
        .HasForeignKey(p => p.UserId)
        .OnDelete(DeleteBehavior.Cascade);
}
```

Для тестування застосунку була підключена бібліотеку Swagger, що надає інтерфейс для роботи з API сервісу.

Auth		▼
POST	/token	🔒
Protected		▼
GET	/api/Protected To access this method use Bearer Token Auth with the Jwt Token acquired from "/token" endpoint	🔒
Source		▼
POST	/api/source/electronic	🔒

Код налаштування виглядає наступним чином

```
// Enable middleware to serve generated Swagger as a JSON endpoint.
app.UseSwagger();

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
// specifying the Swagger JSON endpoint.
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
});
```

Весь код можна переглянути за наступними посиланнями:

<https://github.com/holyfair/site> - Front-End

<https://github.com/holyfair/site-api> - Back-End

Висновки

У даній роботі було розглянуто загальні типи веб-додатків та розроблена власна багаторівнева архітектура з долученням фаєрволів та балансувальників навантаження. Під час розробки для налаштування відмовостійкості було застосовано резервне копіювання сервісів, а також самолікування. Для кожного рівня були поставлені захисні фільтри firewalls, а також налаштовані компоненти моніторингу. Також для кожного рівня були налаштовані процедури автоматичного оновлення коду. Загалом під час розробки застосунку були використані та продемонстровані найбільш популярні компоненти для розробки веб-додатку від платформи Azure і можна сказати, що вона не дарма займає друге місце за популярністю серед інших хмарних платформ.

Список джерел

1. Загальні архітектури веб-додатків — Режим доступу
<https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>
2. Azure Firewalls – Режим доступу:
<https://azure.microsoft.com/ru-ru/services/azure-firewall/#overview>
3. Azure Monitor – Режим доступу:
<https://azure.microsoft.com/ru-ru/services/monitor/>
4. Azure Pipelines – Режим доступу:
<https://azure.microsoft.com/ru-ru/services/devops/pipelines/>

Додатки

А) Налаштування CI/CD бекенду

```

jobs:
  build:
    runs-on: windows-latest

    steps:
      - uses: actions/checkout@v2

      - name: Set up .NET Core
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: '2.2'

      - name: Build with dotnet
        run: dotnet build --configuration Release

      - name: dotnet publish
        run: dotnet publish -c Release -o ${env.DOTNET_ROOT}/myapp

      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v2
        with:
          name: .net-app
          path: ${env.DOTNET_ROOT}/myapp

  deploy:
    runs-on: windows-latest
    needs: build
    environment:
      name: 'production'
      url: ${ steps.deploy-to-webapp.outputs.webapp-url }

    steps:
      - name: Download artifact from build job
        uses: actions/download-artifact@v2
        with:
          name: .net-app

      - name: Deploy to Azure Web App
        id: deploy-to-webapp
        uses: azure/webapps-deploy@v2
        with:
          app-name: 'vaksight-api'
          slot-name: 'production'
          publish-profile: ${ secrets.AzureAppService_PublishProfile_4d14aadd68824bfc91e907b7b992722a }
          package: .

```

Б) Налаштування CI/CD фронтенду

```

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Set up Node.js version
        uses: actions/setup-node@v1
        with:
          node-version: '10.x'

      - name: npm install, build, and test
        run: |
          npm install
          npm run build --if-present

      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v2
        with:
          name: node-app
          path: build

  deploy:
    runs-on: ubuntu-latest
    needs: build
    environment:
      name: 'production'
      url: ${ steps.deploy-to-webapp.outputs.webapp-url }

    steps:
      - name: Download artifact from build job
        uses: actions/download-artifact@v2
        with:
          name: node-app

      - name: 'Deploy to Azure Web App'
        id: deploy-to-webapp
        uses: azure/webapps-deploy@v2
        with:
          app-name: 'vaksight'
          slot-name: 'production'
          publish-profile: ${ secrets.AzureAppService_PublishProfile_e46a1adde44e49d39df33757fa621d4b }
          package: .

```