Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

«ІНФОРМАЦІЙНА СИСТЕМА КЕРУВАННЯ АВТОРСЬКИМИ МІКРОБЛОГАМИ»

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

№	П.І.Б.	Група
1	Батула Д. Ф.	КН-А-191
2	Синяков Д. О.	КН-П-191
3	Луста А. С.	КН-П-191
4	Антонішин Д. О.	КН-П-191
5	Потапенко К. В.	КН-Д-191
6	Кечко О. С.	КН-Д-191

Автор звіту

_____ Батула Даніїл Федорович

Одеса – 2023

АНОТАЦІЯ

Інформаційна система керування авторськими мікроблогами заснована на проекту Twitter, розроблена з метою створення платформи для обміну короткими повідомленнями та мікроблогінгу. Система надає користувачам можливість створювати облікові записи, де вони можуть публікувати свої повідомлення. Користувачі мають змогу підписуватися на інші облікові записи, отримувати оновлення з їхніх мікроблогів та взаємодіяти з ними шляхом відповідей.

Система також надає інтерфейс для пошуку та фільтрації повідомлень залежно від встановлених параметрів, що дозволяє користувачам знаходити та спілкуватися з іншими зацікавленими особами. Додатково, інформаційна система керування авторськими мікроблогами підтримує можливість вказувати хештеги у повідомленнях, що дозволяє їх класифікувати та знаходити за певними темами або ключовими словами.

Наш проект є інструментом для обміну короткими повідомленнями та спілкуванням з іншими користувачами. Він надає можливості, подібні до оригінального Twitter, та є відкритим для розвитку та розширення функціональності залежно від потреб користувачів.

Для розробки на стороні сервера використовується технологія ASP .NET Core 6, мова програмування C# і база даних MySQL. Для розгортання та хостування додатку використовуються технології AWS, Docker, GitHub Actions. Розробка на клієнтській стороні використовує мову програмування C# і технологію .NET MAUI.

ЗМІСТ

ВСТУП

Розвиток і поширення інформаційних технологій супроводжується перенесенням великої кількості послуг у цифровий простір. Комунікаційні технології стають необхідним інструментом соціального прогресу та одним з основних чинників інноваційного розвитку IT індустрії. В цьому контексті виникають нові інформаційні системи, які сприяють ефективному обміну інформацією та комунікації у цифровому світі.

Одним з цікавих проектів, що базується на вищезазначених принципах, є інформаційна система керування авторськими мікроблогами. Вона надає можливість користувачам відчути привабливість і потужність популярної соціальної мережі Twitter, створюючи свої облікові записи та публікуючи короткі повідомлення, відомі як мікроблоги.

У даній роботі ми розглянемо проект зроблений на основі Twitter. Інформаційну систему керування авторськими мікроблогами, її реалізацію зі сторони DevOPS. Детальніше опишемо загальну характеристику системи, проаналізуємо її технічне завдання та дослідження в області інформаційної безпеки, розгортання, та оновлення.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

Загальний опис

Програмне забезпечення (ПЗ) для поширення інформації шляхом спілкування з іншими користувачами.

Призначення

Надання можливості поширення та розповсюдження інформації в мережі Інтернет. Збереження даних та надання конфіденційності користувачам, забезпечуючи безпеку та захист особистої інформації

Вимоги до функціоналу:

- Реєстрація та авторизація
    - Реєстрація
    - Підтвердження особистості
    - Авторизація
- Особистий акаунт користувача
    - Особистий профіль
    - Зміна особистих даних
    - Зміна паролю
- Інформаційні пости
    - Створення постів
    - Поширення особистих постів
    - Поширення постів, що сподобалися
    - Поширення постів інших користувачів
    - Зберігання постів
    - Видалення постів
    - Коментування постів
- Підпис та підписники
    - Підпис на інших користувачів
    - Відписка від інших користувачів

Архітектура:

- Сервер бази даних - серверне програмне забезпечення, що забезпечує збереження та обробку поточних даних.

- Сервіс Web-API – серверне програмне забезпечення, що надає програмний інтерфейс для взаємодії (REST-API) з іншим програмним забезпеченням.
- Мобільний додаток керування контентом - додаток що встановлюється на мобільні пристрої та надає користувачам інтерфейс для взаємодії з функціоналом.

Технології:

- Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології та методології створення:
    - Бази даних – MySQL
    - Сервіс Web-API – REST-API, ASP.NET Core, C#
    - Мобільний додаток – .NET MAUI, C#
- При розгортанні програмного забезпечення рекомендується задіяти хмарні технології
- При розробці програмного забезпечення необхідно задіяти зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту).
- Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Архітектура платформи складається з таких частин:

- «Frontend»
- «Design»
- «Backend»
- «DevOps»

«FRONTEND»

Основна мета «Frontend» полягає в забезпеченні зручного та інтуїтивно зрозумілого інтерфейсу для користувачів. Він повинен бути добре організованим, з урахуванням принципів UI/UX дизайну, щоб забезпечити зручну навігацію та взаємодію з функціями системи. Інтерфейс повинен бути адаптивним до різних типів пристроїв, та забезпечуючи зручне відображення на різних екранах мобільних пристроїв.

Окрім цього, «Frontend» забезпечує візуалізацію даних та взаємодію з серверною частиною системи. Це означає, що він повинен вміти взаємодіяти з API системи, отримувати та передавати дані, оновлювати інформацію у реальному часі, а також обробляти користувацькі запити та дії.

«DESIGN»

Секція «Design» складається з колекції моделей та детальних описів, які представляють поведінку користувачів у системі. Ці моделі охоплюють різні типи інтерфейсів взаємодії між користувачами та інформаційною системою, а також розкривають шляхи переходів між різними інтерфейсами. При створенні цих інтерфейсів ми дотримувалися принципів розумної навігації, і використовували найновітніші підходи у сфері UI/UX дизайну для створення зручного та естетичного користувацького досвіду.

Застосування принципів UI/UX дизайну дало змогу ретельно розрахувати всі елементи інтерфейсів, забезпечуючи зрозумілу та зручну взаємодію з системою. Колірні рішення були ретельно підібрані, щоб створити

гармонійний та привабливий зовнішній вигляд. Приділено особливу увагу ергономіці мобільних пристроїв, щоб забезпечити легкий доступ та зручне використання інтерфейсів на різних пристроях.

«BACKEND»

Частина «Backend» є невід'ємною складовою нашої інформаційної системи і відповідає за оброблення та збереження даних, що створюються та модифікуються в процесі роботи системи. Вона забезпечує ефективне збереження і організацію інформації, а також програмний інтерфейс (API), який надає доступ до основних функцій системи.

Використання програмного забезпечення .NET та СУБД MySQL дозволяє нам створити потужний та швидкодіючий "Backend". Платформа .NET дозволяє побудувати масштабовані та ефективні додатки, а СУБД MySQL забезпечує надійне зберігання та доступ до даних.

«DEVOPS»

«DevOps» займає важливу роль у розробці нашого проекту. Його функції включають реалізацію постійної інтеграції та безперервного розгортання додатку на популярній платформі GitHub. Ми використовуємо сервіси AWS для створення інфраструктури, що дозволяє нам забезпечити надійну та масштабовану роботу проекту.

Один із важливих аспектів «DevOps» - це забезпечення доступу клієнтів до віддаленого сервера. Для цього ми використовуємо зворотній проксі-сервер nginx, який гарантує безпеку та ефективність зв'язку з сервером.

Ми використовуємо сервіс CloudWatch для створення моніторингу системи, що дозволяє нам стежити за споживанням ресурсів, аналізувати дані та вчасно виявляти будь-які проблеми чи незвичайні відхилення у роботі системи.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ DEVOPS ЧАСТИНИ

Ми розпочали з визначення меж проекту в контексті застосування практики DevOps. Під час обговорення з командою ми визначили, які компоненти і процеси повинні бути включені до області практики DevOps, щоб досягти найкращих результатів:

- Монолітний додаток
- Хмарні технології
- Контроль версий
- Постійна інтеграція та безперервне розгортання (CI/CD)
- Моніторинг системи
- Безпека та захист даних

Монолітний додаток

Перше що ми обрали це використання монолітного додатку. Монолітний додаток є єдиним суттєвим елементом, в якому весь функціонал знаходиться всередині одного додатку. Це спрощує процес розробки, оскільки команда може зосередитися на створенні функціоналу без необхідності управління складною інфраструктурою та мікросервісами. Також, він може бути більш ефективним у використанні ресурсів, оскільки відсутні витрати на комунікацію та взаємодію між різними сервісами.

Вибір використання монолітного додатку базується на зручності розробки та підтримки, простоті масштабування та оптимізації ресурсів, особливо на ранніх етапах проекту. Однак, варто враховувати, що зі зростанням проекту та збільшенням складності функціоналу може знадобитися перехід до більш гнучкої архітектури, наприклад, мікросервісної.

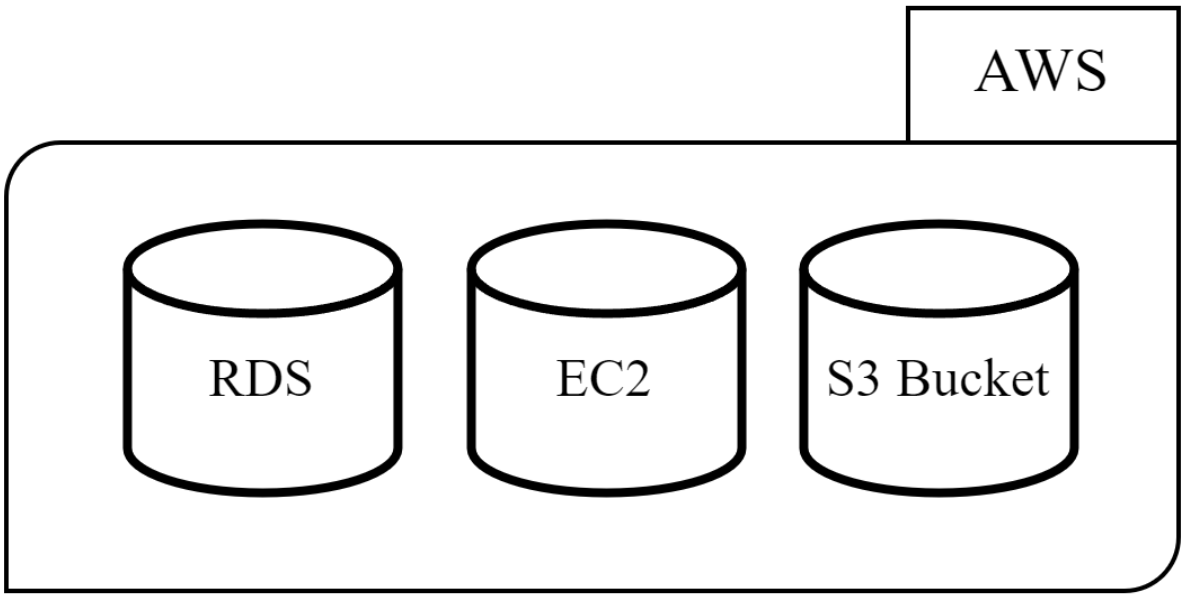
Хмарні технології

Ми вибирали, яким хмарним сервісом користуватись. На даний момент на ринку присутні 3 найрозвинутіші компанії (Google Cloud Platform, Microsoft

Azure, Amazon Web Services), тому ми вибирали з них. Google Cloud Platform поступалася з порівнянням інших. Azure та AWS були на ринку довше, що дозволило їм набрати більшу популярність та розгорнутих центрів обробки даних у світі, що може забезпечити більш широку географічну доступність. Також, велику кількість розширених сервісів та інструментів, які дозволяють користувачам розгортати та управляти широким спектром додатків та інфраструктури. У них розвинена екосистема партнерів, що допомагає підтримувати багато рішень та інтеграційних можливостей.

Спочатку ми хотіли обрати Microsoft Azure платформу як головну, оскільки вона містить всі необхідні технології в одному місці. Ми мали можливість зробити всю організацію додатку в одному місці. Але, ми залишили дану ідею через те, що безкоштовний аккаунт Azure дозволяє долучати тільки п’ять людей до проекту. За більшу кількість людей потрібно доплатити, що ми не зовсім хотіли.

Головною ідеєю було створення всієї інфраструктури в одному місці, але із-за наведеної причини вище ми відмовились від цієї платформи. На наш погляд AWS і Azure дуже схожі по кількості представлених можливостей. І було вирішено використовувати AWS. В ній була створена RDS і S3 Bucket, та EC2.



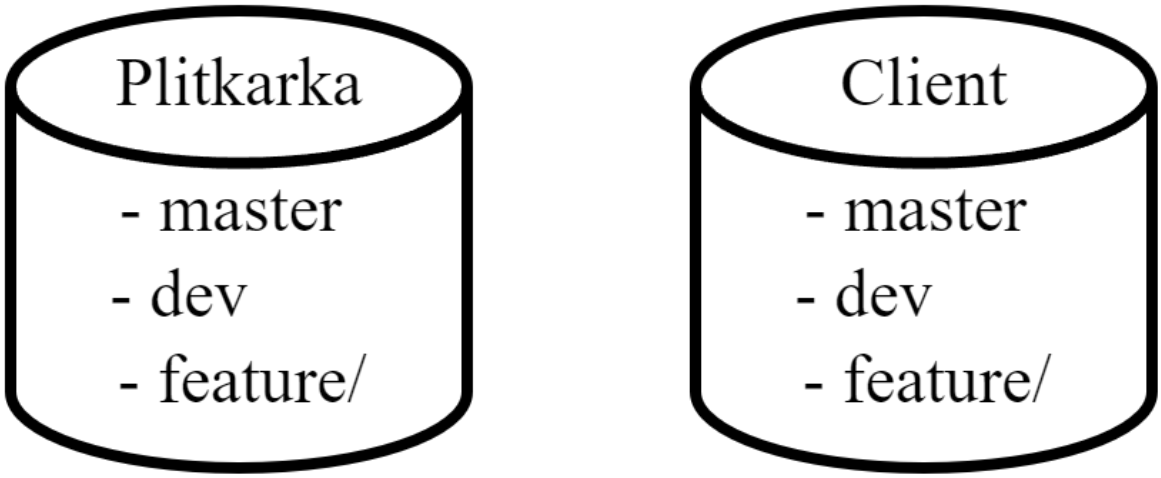
- RDS - база даних, яка використовується для зберігання усіх важливих даних. Таких як: користувачі, пости, коментарі, лайки, підписки та інші.
- S3 Bucket - сховище, яке використовується для зберігання даних у формі зображень. Наприклад, при публікації постів, чи редагування персонального зображення на головній сторінці профілю.
- EC2 - віртуальна автівка, яка використовується для хостування частини додатку, який відповідає за зв'язок RDS і S3 Bucket з клієнтом.

Контроль версий

Далі ми вирішували якою системою контроля версій користуватися. Перша ідея була зав'язана з платформою Azure, але ми відмовились від її реалізації за вже наведеної причини. Тому ми обрали всім відомий GitHub. Для реалізації нашого додатку було створено 2 репозиторія: Plitkarka(backend), client(frontend).

Спочатку, ми використовували один репозиторій для всього проекту загалом. Але, зі збільшенням додатку з’явилася потреба розділити проект для більш зручного користування.

Для реалізації безпеки в нашому репозиторії було створено ієрархію гілок. Ми маємо 2 головні гілки в нашому проекті, та додаткові:



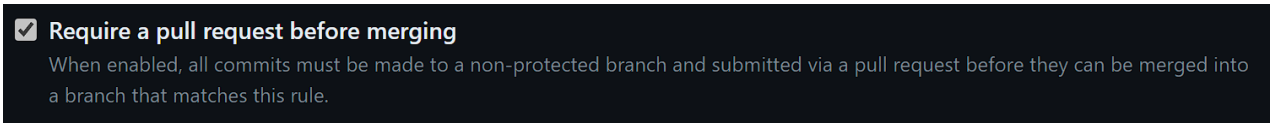
- **master** відповідає за стабільну версію проекта, який може використовуватися в продакшені.
- **dev** використовується для розробки нових версій та тестування. На основі цього створюється стабільна версия, яка вже потім додається до основної гілки master.
- **feature/** використовується для додавання нових фішок, і злиття з гілкою dev

Після написання нових функцій в гілках feature/, вони додаються на dev гілку, там вони тестуються, та виправляються якщо це необхідно. Коли нова версія додатку проходить всі перевірки, вона додається в master і створюється нова стабільна версія.

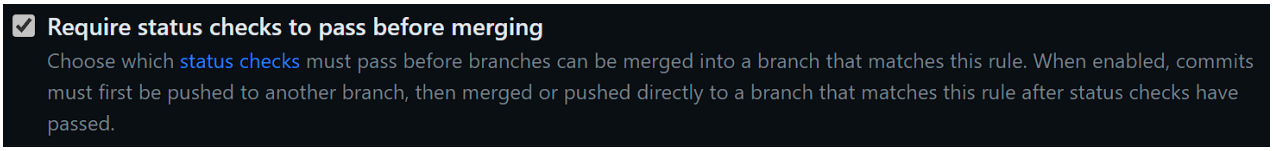
Для створення безпеки, цілісності та якості коду в репозиторіях було реалізовано механізм “Правила безпеки гілок”. Branch protection rules в GitHub - це механізм, який дозволяє встановлювати обмеження та обов'язкові правила для використання гілок у репозиторіях на GitHub.

Для dev гілки було зазначено налаштування:

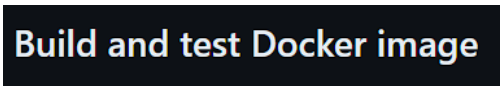
1. Вимагати запит на витягування перед об’єднанням



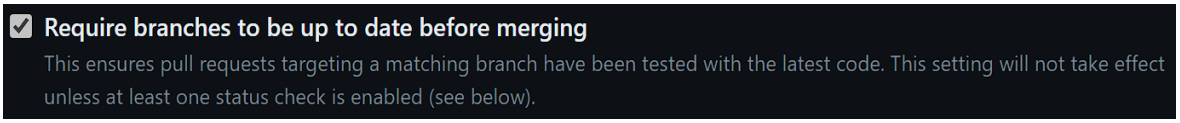
2. Вимагати проходження перевірки статусу перед об’єднанням



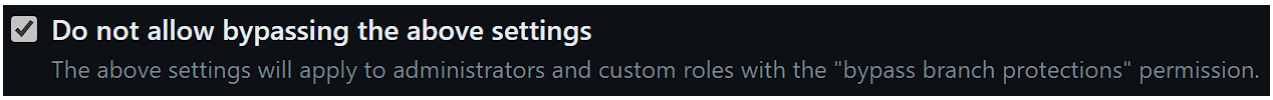
2.1. Build and test Docker image



2.2. Перед об’єднанням гілок потрібно оновлювати

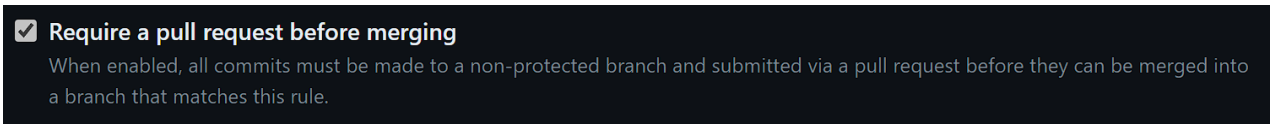


3. Не дозволяйте обходити наведені вище налаштування

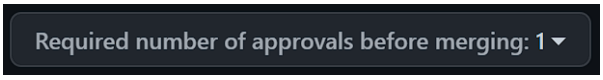


Для master гілки було зазначено налаштування:

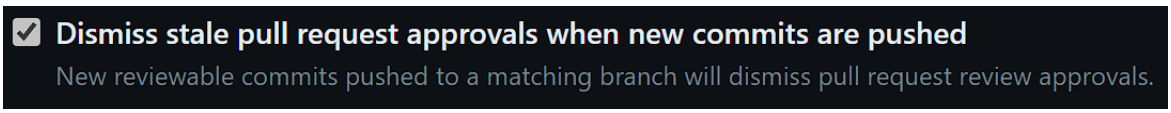
1. Вимагати запит на витягування перед об’єднанням



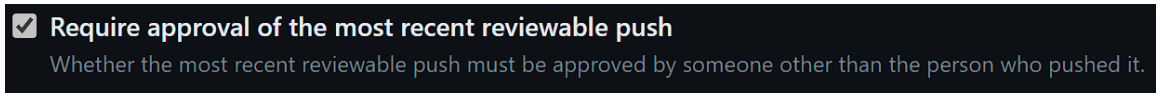
1.1. Потрібні погодження інших: 1



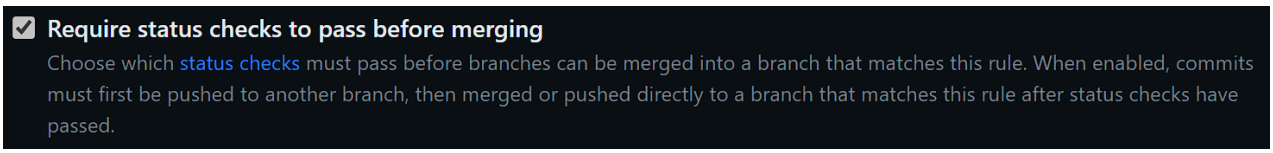
1.2. Відхиляти застарілі схвалення запитів на отримання під час надсилання нових комітів



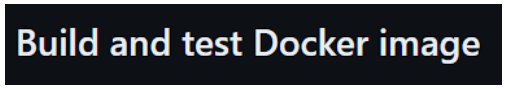
1.3. Вимагати схвалення останнього надсилання, яке можна переглянути



2. Вимагати проходження перевірки статусу перед об’єднанням

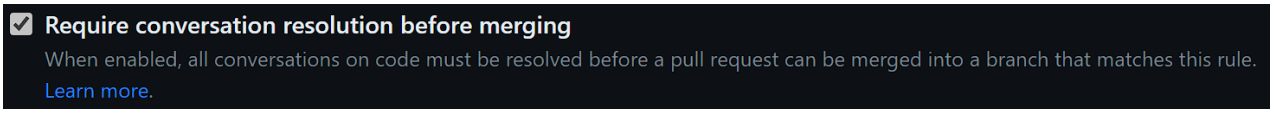


2.1. Build and test Docker image

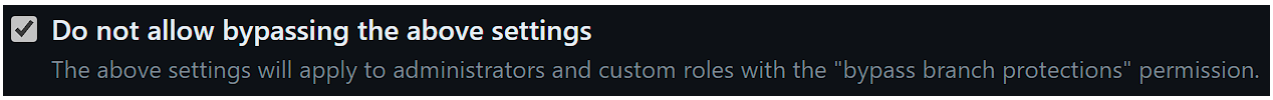


2.2. Перед об’єднанням гілок потрібно оновлювати

3. Вимагати вирішення розмови перед об’єднанням

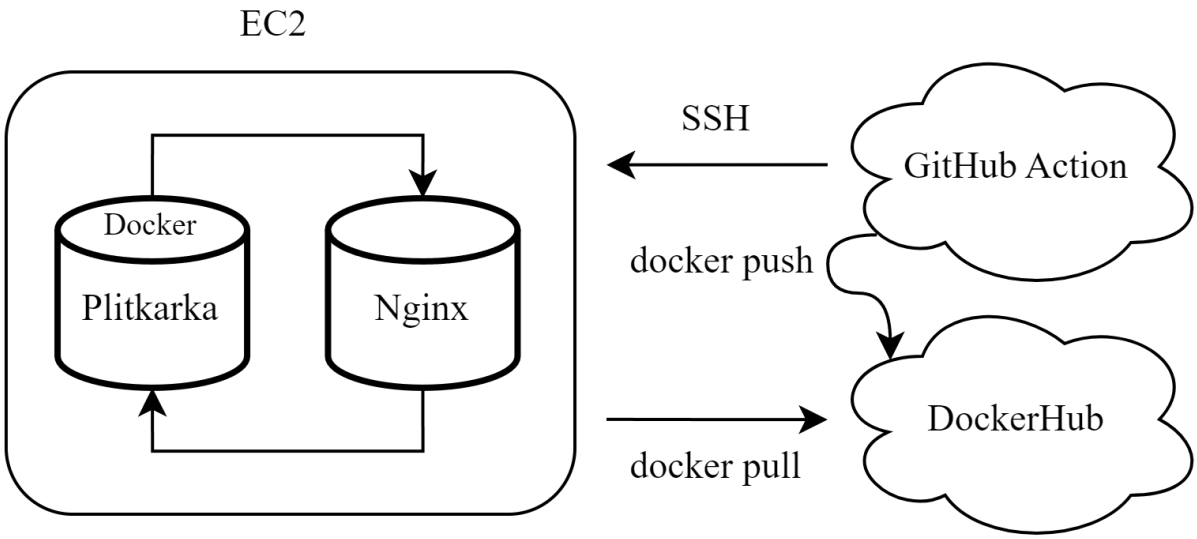


4. Не дозволяйте обходити наведені вище налаштування



Постійна інтеграція та безперервне розгортання (CI/CD)

Для реалізації цього етапу було обрано сервіс GitHub Actions. Головною причиною стало те, що раніше ми вже обрали GitHub, як систему контроля версій. І дуже зручно використовувати сервіс який вже є в самому GitHub. Також було цікаво обрати нову систему для вивчення.



Було створено два головних сценария для CI/CD:

1. pull_request.yml
2. push.yml

Перший використовується для тестування додатку, перед злиттям гілок.

1. Спрацьовує при створенні нового запиту на вилучення

```yaml
on:
  pull_request:
    branches:
      - master
      - dev
```

2. Завантаження репозиторія на віртуальну автівку GitHub

```yaml
- name: Check out the repository
  uses: actions/checkout@v3
```

3. Створення метаданих для Docker

```yaml
- name: Extract metadata (tags, labels) for Docker
id: metadata
uses: docker/metadata-action@v4
with:
```

```
  images: ${{ secrets.DOCKER_USERNAME }}/${{
secrets.DOCKER_IMAGE }}
  tags: |
    # Latest tag for master branch in DockerHub
type=raw,value=latest,enable=${{ github.ref ==
format('refs/heads/{0}', 'master') }}
    # Tag for dev branch in DockerHub
type=raw,value=dev-{{sha}},enable=${{ github.ref ==
format('refs/heads/{0}', 'dev') }}
```

4. Будування нового образу Docker

```
- name: Build Docker image
  id: build_image
  uses: docker/build-push-action@v4
  with:
    context: .
    push: false
    tags: ${{ steps.metadata.outputs.tags }}
    labels: ${{ steps.metadata.outputs.labels }}
```

5. Стартування Docker контейнера для перевірки

```
- name: Run Docker container
  env:
    MySql__ConnectionString: ${{
secrets.MYSQL__CONNECTIONSTRING }} ...
  run: |
    docker run --name ${{ secrets.DOCKER_CONTAINER
}} -d \
    -e MySql__ConnectionString \ ...
... -p 80:80 \
${{ steps.build_image.outputs.imageid }}
```

6. Очікування для підняття контейнеру

```
- name: Sleep for 10s
```

```
  run: sleep 10
```

7. Тестування працездатності додатку

```
- name: Test Docker container
  id: test
  run: echo "STATUS=$(curl
http://localhost/api/health)" >> "$GITHUB_OUTPUT"
```

8. Показати інформацію про тестування

```
- name: Information about Testing
  run: echo Status of working Database - ${{
steps.test.outputs.STATUS }}
```

9. Перевірка тестування

```
- name: ERROR !!!
if: ${{ steps.test.outputs.STATUS != 'Healthy' }}
uses: myrotvorets/set-commit-status-action@master
with:
token: ${{ secrets.GITHUB_TOKEN }}
status: error
context: Problems with database or code
```

Другий використовується для оновлення образів в DockerHub, та AWS EC2

1. Спрацьовує на злиття гілок, та ще раз проходить всі попередні кроки.

```
on:
  push:
    branches:
      - master
      - dev
```

2. Якщо тестування пройшло успішно

```
needs: build
if: ${{ needs.build.outputs.health_check ==
'Healthy' }}
```

3. Завантаження репозиторія на віртуальну автівку GitHub

```
- name: Check out the repository
  uses: actions/checkout@v3
```

4. Вхід до акаунту DockerHub

```
- name: Login to DockerHub
  uses: docker/login-action@v2
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_TOKEN }}
```

5. Створення метаданих для Docker

```
- name: Extract metadata (tags, labels) for Docker
id: metadata
uses: docker/metadata-action@v4
with:
  images: ${{ secrets.DOCKER_USERNAME }}/${{
secrets.DOCKER_IMAGE }}
  tags: |
    # Latest tag for master branch in DockerHub
type=raw,value=latest,enable=${{ github.ref ==
```

```
format('refs/heads/{0}', 'master') }}
    # Tag for dev branch in DockerHub
type=raw,value=dev-{{sha}},enable=${{ github.ref ==
format('refs/heads/{0}', 'dev') }}
```

6. Завантаження нового образу Docker на хмарне сховище DockerHub

```
- name: Push Docker image
  uses: docker/build-push-action@v4
  with:
    context: .
    push: true
    tags: ${{ steps.metadata.outputs.tags }}
    labels: ${{ steps.metadata.outputs.labels }}
```

7. Оновлення версии образу на сервері EC2, якщо попередні кроки завершилися успіхом, та злиття з гілкою master

```
needs: [ build, push ]
if: github.ref == 'refs/heads/master'
```

8. Оновлення версии образу на сервері EC2

```
- name: Executing remote SSH commands
  uses: appleboy/ssh-action@v0.1.10
  env:
    MYSQL__CONNECTIONSTRING: ${{
secrets.MYSQL__CONNECTIONSTRING }} ...
... ASPNETCORE_ENVIRONMENT: ${{
secrets.ASPNETCORE_ENVIRONMENT }}
  with:
    host: ${{ secrets.EC2_HOST }}
    username: ${{ secrets.EC2_USERNAME }}
    key: ${{ secrets.EC2_KEY }}
    port: ${{ secrets.EC2_PORT }}
    envs:
      MYSQL__CONNECTIONSTRING, ...
```

```
  ... EMAILSERVICE__PORT
script: |
  echo "Docker container started at
$(TZ='Europe/Kiev' date "+%A %d-%b-%Y %H:%M:%S
%Z")" >> deploy.txt
  docker stop ${{ secrets.DOCKER_CONTAINER }}
  docker rm ${{ secrets.DOCKER_CONTAINER }}
  docker rmi ${{ secrets.DOCKER_USERNAME }}/${{
secrets.DOCKER_IMAGE }}
  docker pull ${{ secrets.DOCKER_USERNAME }}/${{
secrets.DOCKER_IMAGE }}:latest
  docker run --name ${{ secrets.DOCKER_CONTAINER }}
-d \
-e S3Service__AccessKey="$S3SERVICE__ACCESSKEY"
\...
... -e ASPNETCORE_ENVIRONMENT \
-p 127.0.0.1:82:80 \
${{ secrets.DOCKER_USERNAME }}/${{
secrets.DOCKER_IMAGE }}
```

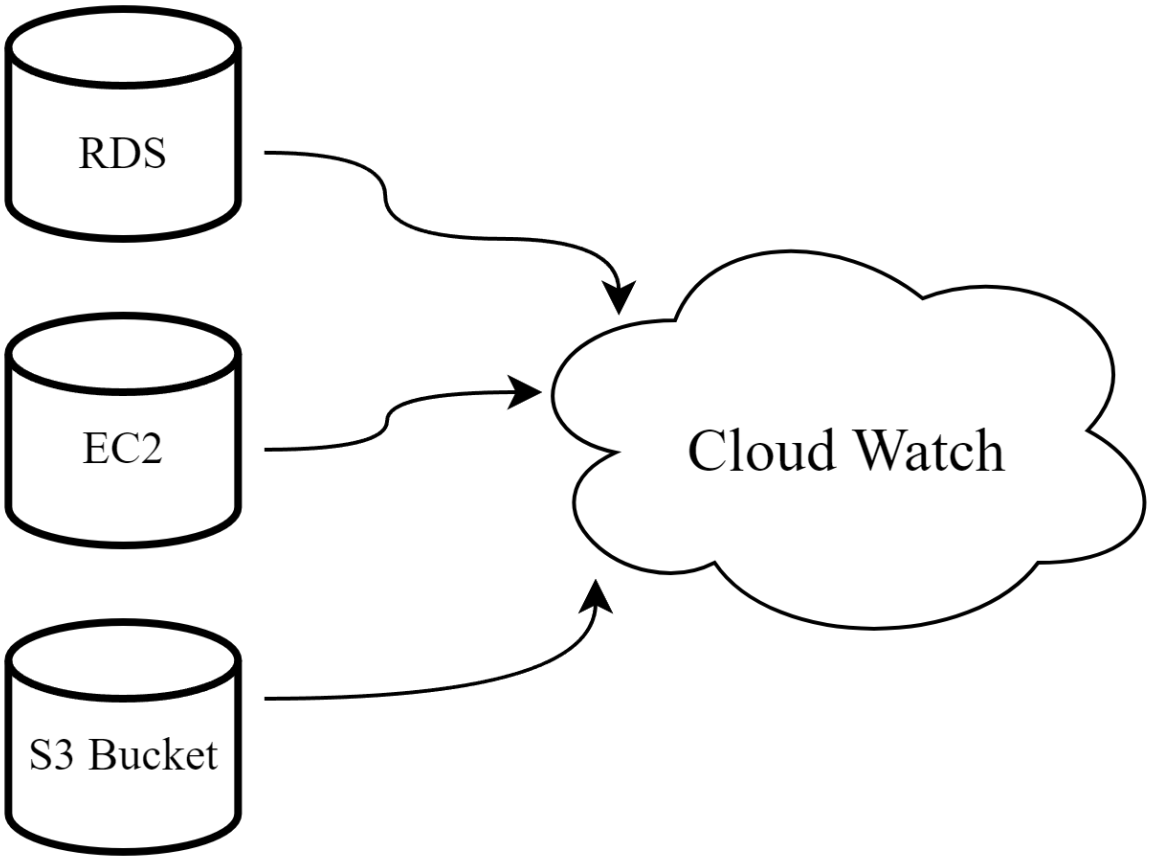
Також, ми використовуємо технологію Docker для нашого проекту. Це дозволяє створювати та швидко оновлювати нові версії додатку на віддаленому сервері. Всі нові образи зберігаються на DockerHub.

Конфігурація Dockerfile:

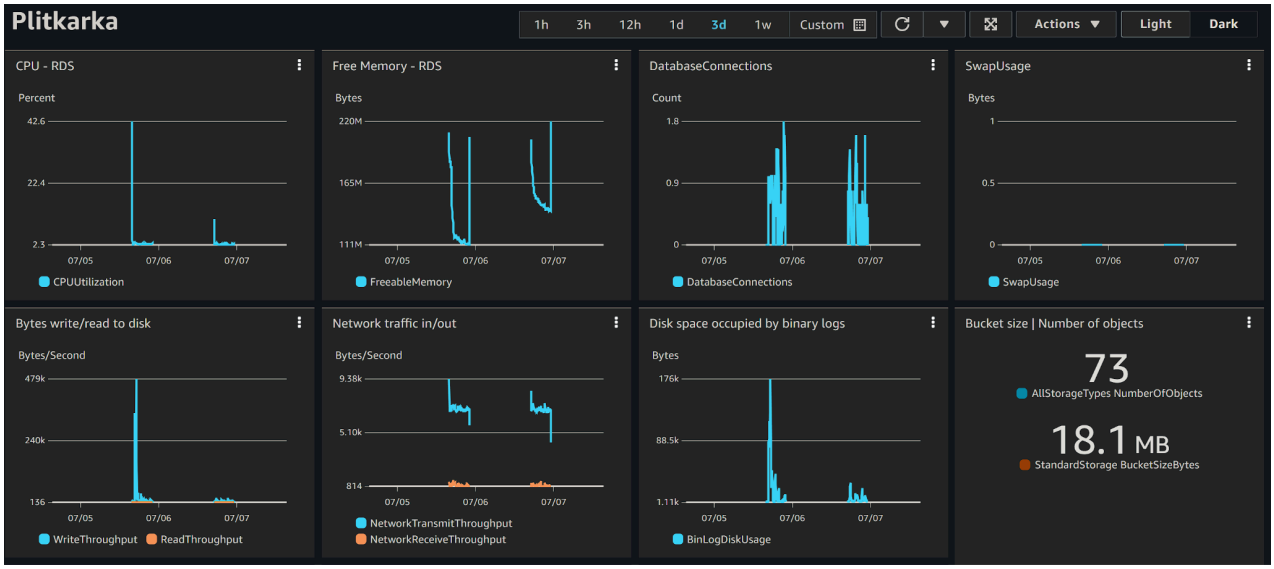
```
# Використання базового образу sdk:7.0
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env
# Перехід до робочого каталогу
WORKDIR /App
# Копіювання проекту в образ
COPY . ./
# Встановлення залежностей та інструментів
RUN dotnet restore
# Будування додатку
RUN dotnet publish -c Release -o out

# Використання образу aspnet:6.0
FROM mcr.microsoft.com/dotnet/aspnet:6.0
# Перехід до робочого каталогу
WORKDIR /App
# Копіювання нових файлів до образу
COPY --from=build-env /App/out .
# Команда для стартування додатку
ENTRYPOINT ["dotnet", "Plitkarka.Application.dll"]
```

Моніторинг системи

Для моніторингу, ми використовуємо AWS Cloud Watch. Ця технологія дозволяє слідкувати за показниками наших систем.



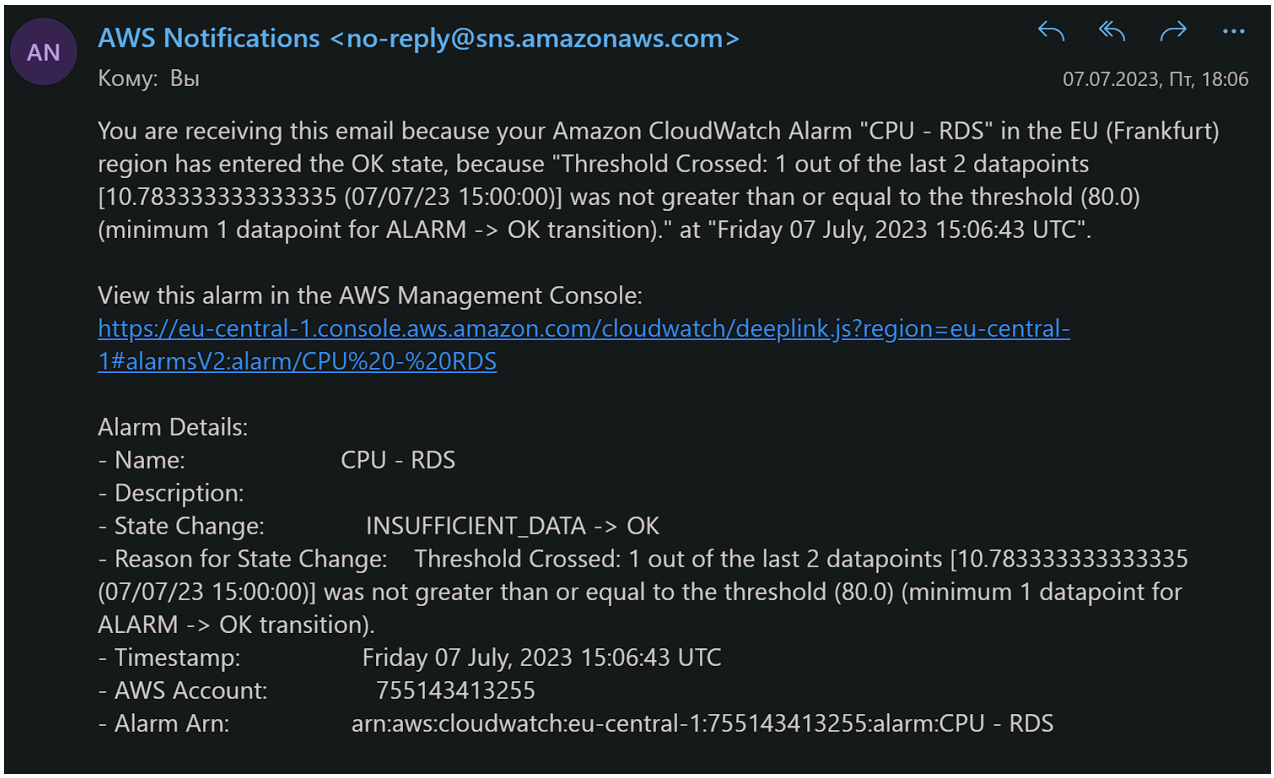
Сервіси AWS віддають метрики Cloud Watch. Що дозволяє нам використовувати їх, для відображення на панелі приладів.



Також, ми маємо оповіщення о значній нагрузці на нашу віртуальну автівку і базу даних, та повернення їх в попередній стан.

Name	State	Last state update	Conditions	Actions
RAM - RDS	Insufficient data	2023-07-07 21:34:14	FreeableMemory <= 80 for 2 datapoints within 10 minutes	Actions enabled
CPU - RDS	Insufficient data	2023-07-07 21:33:43	CPUUtilization >= 80 for 2 datapoints within 10 minutes	Actions enabled
CPU - EC2	OK	2023-07-06 16:17:03	CPUUtilization >= 80 for 2 datapoints within 10 minutes	Actions enabled

Всі оповіщення о зміні стану сервісів надсилаються на нашу електронну адресу. Там міститься вся важлива інформація: назва оповіщення, зміна стану, умови при яких відбулася зміна, час, аккаунт.



AN

AWS Notifications <no-reply@sns.amazonaws.com>

Кому: Вы

07.07.2023, Пт, 18:06

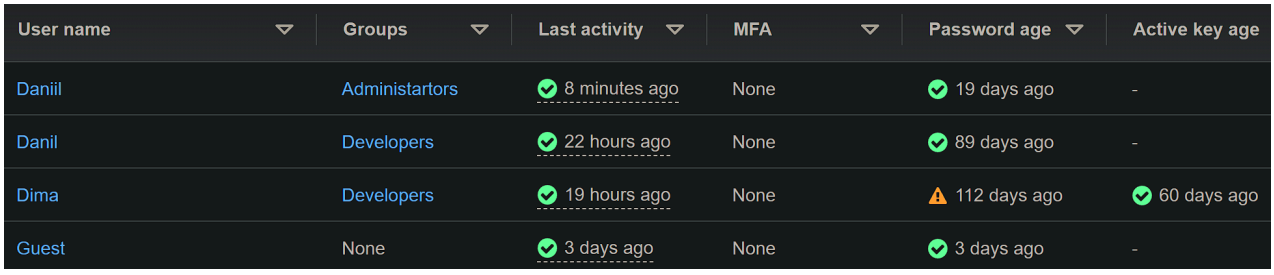
You are receiving this email because your Amazon CloudWatch Alarm "CPU - RDS" in the EU (Frankfurt) region has entered the OK state, because "Threshold Crossed: 1 out of the last 2 datapoints [10.783333333333335 (07/07/23 15:00:00)] was not greater than or equal to the threshold (80.0) (minimum 1 datapoint for ALARM -> OK transition)." at "Friday 07 July, 2023 15:06:43 UTC".

View this alarm in the AWS Management Console:
https://eu-central-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=eu-central-1#alarmsV2:alarm/CPU%20-%20RDS

Alarm Details:
- Name: CPU - RDS
- Description:
- State Change: INSUFFICIENT_DATA -> OK
- Reason for State Change: Threshold Crossed: 1 out of the last 2 datapoints [10.783333333333335 (07/07/23 15:00:00)] was not greater than or equal to the threshold (80.0) (minimum 1 datapoint for ALARM -> OK transition).
- Timestamp: Friday 07 July, 2023 15:06:43 UTC
- AWS Account: 755143413255
- Alarm Arn: arn:aws:cloudwatch:eu-central-1:755143413255:alarm:CPU - RDS

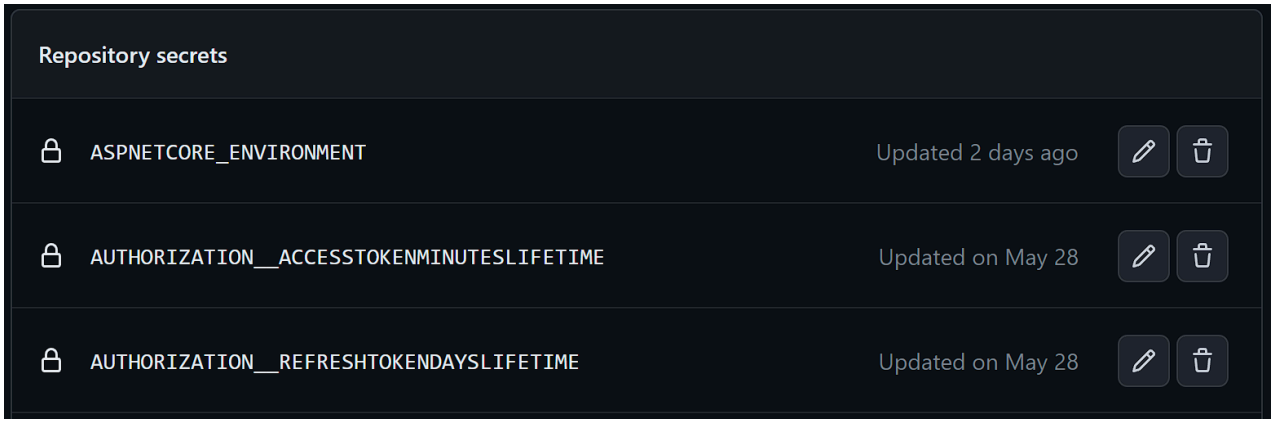
Безпека та захист даних

На кожну людину, яка має доступ до AWS сервісів, створено індивідуальний профіль, та налаштовані права доступу за необхідністю використання певних сервісів.

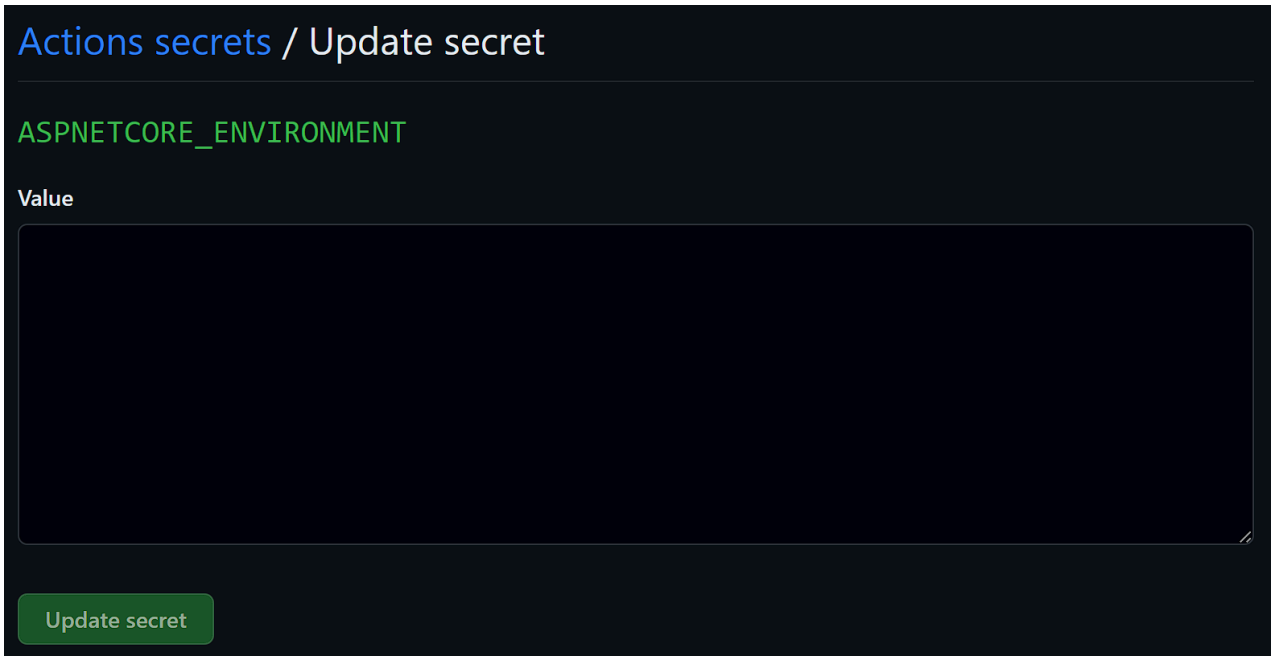


User name	Groups	Last activity	MFA	Password age	Active key age
Daniil	Administartors	8 minutes ago	None	19 days ago	-
Danil	Developers	22 hours ago	None	89 days ago	-
Dima	Developers	19 hours ago	None	112 days ago	60 days ago
Guest	None	3 days ago	None	3 days ago	-

Всі чутливі дані для бази даних або авторизації, які використовуються в GitHub Actions, зберігаються в GitHub у підрозділі secrets and variables.



При зміні даних, останні значення змінних не відображаються. Тому, при доступі сторонньої людини до профілю GitHub, вона не буди мати можливість дізнатись даних для авторизації або іншого.



Додаток який розташований на віддаленому сервері захищений від прямого взлому за допомогою проксі-сервера Nginx, та налаштованого SSL з підтвердженим сертифікатом від Let's Encrypt. Всі поступаючі дані надходять до проксі-серверу nginx через HTTPS, він перенаправляє запит на контейнер додатку, який в свою чергу, повертає відповідь на сервер, де вже він надсилає

їх до клиента.

Конфігурація проксі-nginx:

1. Прослуховування 80-го порта та переадресація на захищене з'єднання
```
server {
    listen 80;
    server_name plitkarka.store www.plitkarka.store;
    rewrite ^ https://$host$request_uri permanent;
}
```
2. Прослуховування 443-го порта, та визначення доменного імені
```
server {
listen 443 ssl;
server_name plitkarka.store;
```
3. Шляхи для знаходження сертифікатів Let's Encrypt
```
ssl_certificate
/etc/letsencrypt/live/plitkarka.store/fullchain.pem;
ssl_certificate_key
/etc/letsencrypt/live/plitkarka.store/privkey.pem;
include /etc/letsencrypt/options-ssl-nginx.conf;
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
ssl_session_cache builtin:1000 shared:SSL:10m;
```
4. Шлях для збереження логів
```
access_log /var/log/nginx/access.log;
```
5. Конфігурація nginx, та перенаправлення запитів на сервер
```
location / {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    proxy_pass http://127.0.0.1:82;
```

```
    proxy_read_timeout 90;
    proxy_redirect off;
}
```

6. Блокування API (для розробки)

```
    location ^~ /api/test {

    return 404;
    }
}
```

РОЗДІЛ 3
ОЦІНКА

На основі створенного проекта можна сказати, що проект реалізован за допомогую ключових принципів та практик DevOps для забезпечення ефективності та надійності процесів розробки та розгортання програмного забезпечення.

Використовується системи контролю версій для управління кодом та його інтеграції, дозволяючи команді ефективно співпрацювати та вносити зміни до проекту. Використання гілок для різних версій програмного забезпечення дозволяє забезпечити безпеку та стабільність процесу розробки.

Також, створення неперервної інтеграції та неперервного розгортання (CI/CD) для автоматизації тестування, збирання та розгортання програмного забезпечення. Це дозволяє забезпечити швидкий і безперервний процес розгортання, зменшуючи час та зусилля, необхідні для впровадження змін.

Крім того, використання моніторингу системи та реагування на події допомагає забезпечити стабільну роботу проекту та вчасно виявляти проблеми. Описані вище розділи також зазначають важливість збереження конфіденційних даних та застосування заходів безпеки, таких як використання проксі-серверів та шифрування SSL-з'єднань.

Узагалі, застосування цих практик сприяє покращенню продуктивності, якості та безпеки процесу розробки та розгортання програмного забезпечення. Проект є прикладом не поганої імплементації DevOps методології та може служити початковим прикладом для подальших проектів у сфері розробки програмного забезпечення.

ВИСНОВКИ

Ми розглянули DevOps-практику для розробки та розгортання додатку. Обрали систему контролю версій GitHub і створили два репозиторії для поділу проекту на фронтенд (client) і бекенд (Plitkarka). Створили ієрархію гілок з двома головними: master і dev.

Для забезпечення безпеки та контролю над гілками ми використовували "Правила безпеки гілок" в GitHub. Ці принципи встановлюють обмеження та обов'язкові правила для використання гілок в репозиторії, забезпечуючи безпеку, цілісність та якість коду.

Для реалізації неперервної інтеграції та неперервного розгортання (CI/CD) ми використовували сервіс GitHub Actions. Було створено два сценарії: pull_request.yml та push.yml, які автоматизували процеси збирання, тестування та розгортання додатку. Ці сценарії дозволяють перевіряти код, створювати Docker образи, запускати тести та оновлювати головний сервер на AWS при успішному злитті гілок.

Для моніторингу системи ми використовували AWS CloudWatch, що дозволяє відстежувати показники системи і надсилати повідомлення про стан системи на електронну пошту. Для збереження конфіденційних даних, таких як дані авторизації та доступу до бази даних, ми використовували GitHub.

Також ми забезпечили безпеку від прямих атак на віддалений сервер, використовуючи проксі-сервер Nginx і налаштування SSL з'єднання для шифрування даних.

В цілому, ми успішно впровадили DevOps практики для розробки та розгортання нашого додатку. Використання GitHub, Docker, GitHub Actions, AWS та інших інструментів дозволило нам автоматизувати процеси, підвищити ефективність розробки та забезпечити високу якість та безпеку додатку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

№	Ресурс
1.	GitHub Розробник: GitHub, Inc. Документація
2.	GitHub Actions Розробник: GitHub, Inc. Документація
3.	Docker Розробник: Docker, Inc. Документація
4.	Amazon Web Services Розробник: AWS, Inc. Документація
5.	Nginx Розробник: F5, Inc. Документація
6.	Let's Encrypt, CertBot Розробник: F5, Inc. Документація
7.	Namecheap Розробник: NameCheap, Inc. Документація
8.	SQL Database with Auto-pause (Solution) Автор: Dave Callan Документація