Приватний заклад вищої освіти

Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

«ІНФОРМАЦІЙНА СИСТЕМА УЗГОДЖЕННЯ ВЗАЄМОДІЇ КОРИСТУВАЧІВ (адмін)»

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

Виконавці проекту

№	П.І.Б.	Група
1	Денисов Дмитро Дмитрович	КН-А-191
2	Кравчук Дмитро Вітольтович	КН-П-191
3	Сарієв Георгій Валентинович	КН-П-191
4	Ілля Лоскутніков Сергійович	КН-П-191
5	Самбаєва Карина Сергіївна	КН-Д-191

Автор звіту

_____ Денисов Дмитрий Дмитриевич

Одеса – 2023

АНОТАЦІЯ

УДК 004.92

Денисов Д. Д. Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп’ютерні науки». - Одеса: ОТУШ, 2023 - 13 с.

Дипломна робота присвячена розробці та впровадженню системи DevOps для аналогу проекта kabanchik.ua, відомого як BusyBee. Kabanchik.ua є популярною онлайн-платформою, яка забезпечує послуги та обмін ресурсами між користувачами. Однак, процес розробки та впровадження нових функцій на платформі може бути складним і малоефективним.

Основною причиною цього проекту є покращення процесу розробки та впровадження нового функціоналу якого немає на платформі kabanchik.ua. Ця робота ставить за мету використання практик та інструментів DevOps для автоматизації та оптимізації розробки, тестування, розгортання та моніторингу системи.

Основні задачі дипломної роботи включають:

- Вивчення існуючого процесу розробки та впровадження в проекті BusyBee.
- Дослідження практик, ключових проблем та інструментів DevOps для покращення ефективності процесу розробки.
- Налагодження постійної інтеграції та постійної доставки (CI/CD) для полегшення розробки та впровадження змін.
- Впровадження системи моніторингу та логування для підтримки надійності та доступності платформи.
- Оцінка результатів впровадження системи DevOps, аналіз ефективності та користь, отриманої від нового підходу.

Ціллю дипломної роботи є підвищення швидкості та якості розробки нових функцій на платформі BusyBee шляхом використання практик та інструментів DevOps. Очікується, що впровадження DevOps допоможе зменшити час циклу розробки, підвищить автоматизацію та стабільність розгортання програмного забезпечення.

У проекті BusyBee використовуються основні сервіси та програми, такі як Docker для контейнеризації, Git для керування версіями коду, Nginx-proxy, Jenkins для автоматизації процесу збірки та розгортання, Prometheus для моніторингу та збору метрик та Grafana для візуалізації цих метрик.

ЗМІСТ

ВСТУП

Мета проекту

Створити інформаційну систему узгодження взаємодії користувачів під назвою "Busy Bee" для надання платформи, де користувачі можуть розміщувати завдання та знаходити виконавців.

Основна функціональність:

Розміщення завдань, пошук виконавців, управління користувачами, платіжна система.

Вимоги до Backend:

- Архітектура:
  - Використання монолітної архітектури.
- Технології:
  - Мова програмування: C#
  - Фреймворк: ASP.NET WebApi.
  - ORM: Entity Framework Core для роботи з базою даних PostgreSQL.
  - Мапер: AutoMapper для перетворення даних між моделями.
  - Валідація: FluentValidation для перевірки вхідних даних.
  - Аутентифікація та авторизація: Використання Azure AD B2C для управління користувачами.
  - Шаблон проектування: Clean architecture.
  - Unit of Work / Repository Pattern: Для управління доступом до даних.
  - Документація API: Використання Swagger для автоматичного генерування документації.
  - Тестування: Використання Postman для тестування API.

Вимоги до DevOps:

- Контейнеризація: Використання Docker для контейнеризації додатка.
- Хостинг: Розгортання на Azure Virtual Machine.
- Система контролю версій: Використання Azure Repos для управління кодовою базою.
- CI/CD інструменти: Використання Jenkins для автоматизації процесів CI/CD.
- Моніторинг: Налаштування Prometheus і Grafana для моніторингу системи.
- Проксі-сервер: Використання Nginx в якості проксі-сервера.
- Планування та розгортання інфраструктури проекту.
- Автоматизація процесів розгортання, моніторингу та масштабування системи.
- Встановлення та налаштування інструментів для CI/CD.
- Забезпечення безпеки та контролю якості проекту.
- Управління конфігурацією та забезпечення надійності системи.

Вимоги до Design:

- Проектування користувацького інтерфейсу та користувацького досвіду.
- Використання інструментів для дизайну: Фігма, Ілюстратор, Фотошоп.

Вимоги до Frontend:

- Фреймворк: Використання Angular для розробки фронтенд-частини сайту.
- Інтегроване середовище розробки: Використання WebStorm для розробки на Angular.
- Управління залежностями: Використання npm для управління пакетами.

Загальний опис проекту:

Опис функціональності:

Розміщення завдань:

- Створення нового завдання з вказанням категорії, опису та інших необхідних полів.
- Редагування та видалення завдання.
- Управління статусом завдання (відкрите, в роботі, завершене).

Пошук виконавців:

- Пошук виконавців за категоріями, навичками, локацією та іншими параметрами.
- Відображення списку доступних виконавців з основною інформацією та рейтингом.
- Фільтрація результатів пошуку та сортування за різними критеріями.

Оцінки та відгуки:

- Можливість оцінювати виконавців та залишати відгуки після завершення завдання.
- Відображення рейтингу та відгуків на профілях виконавців.
  Цільова аудиторія:
- Визначення цільової аудиторії відповідно до виду завдань та послуг, пропонованих на сайті.
- Аналіз потреб і очікувань користувачів щодо функціональності та користувацького досвіду.

Розширюваність:

- Розробка модульної архітектури, що дозволяє легко додавати нові функції та можливості.
- Розгляд можливості інтеграції з іншими сервісами або платформами для розширення функціональності сайту.

Дизайн та користувацький інтерфейс:

Загальний стиль дизайну:

- Визначення бажаного стилю дизайну (мінімалістичний).
- Визначення колірної палітри, шрифтів та інших візуальних елементів, що відповідають обраному стилю.

Макети та прототипи:

- Створення макетів користувацького інтерфейсу для різних сторінок та функціональних елементів сайту.
- Розробка інтерактивних прототипів для демонстрації користувацької взаємодії та навігації по сайту.

База даних та зберігання даних:

Вибір та опис бази даних:

- Розробка структури бази даних, включаючи таблиці, зв'язки між ними та ключові поля.
- Врахування вимог до продуктивності та масштабованості бази даних.
- Зберігання даних:
- Визначення даних, які повинні зберігатися в базі даних, включаючи інформацію про користувачів, завдання, коментарі тощо.
- Розробка моделей даних та визначення зв'язків між ними.

Безпека:

Аутентифікація та авторизація:

- Реалізація процесів аутентифікації користувачів з використанням Azure AD B2C.
- Встановлення прав доступу та ролей для різних типів користувачів.

Захист даних:

- Застосування заходів безпеки для захисту користувацьких даних, включаючи шифрування зберігання та передачі даних.

Інфраструктура:

Контейнеризація:

- Використання Docker для контейнеризації бекенд- та фронтенд-частини додатка.
- Створення Docker-контейнерів для різних компонентів додатка та їх оркестрація.

Хостинг:

- Розгортання додатка на Azure Virtual Machine або іншому облачному сервису.

CI/CD:

- Використання Jenkins для налаштування процесів CI/CD.
- Автоматизація процесів розгортання, тестування та випуску нових версій додатка.

Моніторинг:

- Налаштування Prometheus та Grafana для моніторингу системи.
- Відстеження метрик продуктивності, завантаження та інших параметрів для забезпечення стабільності та швидкодії сайту.

Система контролю версій:

Використання системи контролю версій:

- Інтеграція проекту з Git для управління версіями та історією змін.
- Створення репозиторію Git та налагодження процесу спільної роботи розробників.

Гілкування та злиття:

- Визначення стратегії гілкування та злиття коду, наприклад, використання моделі Git Flow.
- Створення та керування різними гілками для розробки нових функцій, виправлення помилок та випуску стабільних версій.

Робота з командою розробників:

- Опис угод щодо найменування гілок, коментарів до комітів та інших правил для спільної роботи над проектом.
- Визначення процесу рецензування коду та схвалення змін.

Інтеграція з CI/CD:

- Використання інструментів CI/CD, таких як Jenkins, для автоматизації процесів збірки, тестування та розгортання додатка.
- Налаштування пайплайнів CI/CD, що включають етапи збірки, тестування, аналізу коду та розгортання на сервері.

Управління версіями та тегування:

- Опис правил для присвоєння версій релізам, виправленням помилок та іншим змінам.
- Використання тегів Git для маркування важливих точок в історії проекту, таких як релізи або майлстоуни.

Резервне копіювання:

- Розробка стратегії резервного копіювання репозиторію Git для забезпечення збереженості коду та історії змін.

Управління проектом:

Створення плану проекту:

- Визначення етапів розробки, термінів та завдань.
- Встановлення пріоритетів та розподіл ресурсів. Комунікація та співпраця:
- Встановлення засобів комунікації та співпраці між членами команди.
- Використання інструментів для спільного редагування та обміну документами.

Звітність:

- 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.
- Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:
    - root
    - Backend
    - Web
- Підготовка звітів про стан проекту та результати роботи.
- Презентація результатів перед командою та замовником проекту.

РОЗДІЛ 1. Загальна характеристика системи

Архітектура платформи складається з таких частин:

- Frontend
- Design
- Backend
- Devops

Кожна з цих частин виконує важливі функції і має свою суттєву роль у проекті:

Frontend

Фронтенд є тим, що користувачі бачать та взаємодіють з платформою BusyBee. Його суттєва частина полягає в наступному:

- Розробка інтерфейсу користувача: Фронтенд-розробники відповідають за розробку естетичного та інтуїтивно зрозумілого дизайну, який забезпечує зручну навігацію та взаємодію з платформою для користувачів.
- Реалізація клієнтської логіки: Фронтенд-розробники виконують програмування на мові JavaScript та використовують фреймворки, такі як React або аналогічні, для створення функціональних компонентів та обробки подій користувача.
- Взаємодія з бекендом: Фронтенд комунікує з бекендом через API, взаємодіючи з базою даних та отримуючи необхідні дані для відображення на сторінках платформи.

Frontend необхідний, оскільки він створює враження користувача про платформу BusyBee і забезпечує зручну та ефективну взаємодію з нею. Візуальний дизайн, логіка frontend та взаємодія з бекендом роблять платформу привабливою та інтуїтивно зрозумілою для користувачів.

Design

Дизайн є ключовим елементом платформи BusyBee і відповідає за створення естетичного та функціонального інтерфейсу для користувачів. Його суттєва частина полягає в наступному:

- Розробка макетів: Дизайнер створює макети веб-сторінок та мобільних додатків, враховуючи потреби користувачів, принципи взаємодії та сучасні дизайн-стандарти.
- Візуальне оформлення: Дизайнер виконує вибір кольорової палітри, шрифтів, іконок та інших елементів дизайну, що сприяють створенню привабливого та зрозумілого інтерфейсу.
- Подання інформації: Дизайнер розробляє ефективні способи відображення даних та важливої інформації, забезпечуючи логічну та зручну навігацію користувачів.

Design необхідний для того, щоб платформа BusyBee була привабливою та зрозумілою для користувачів. Естетичний дизайн, правильна організація інформації та інтуїтивно зрозумілі елементи дозволяють користувачам легко орієнтуватись та взаємодіяти з платформою.

Backend

Бекенд відповідає за обробку бізнес-логіки, збереження даних та взаємодію з фронтендом та базою даних. Суттєва частина backend включає:

- Розробка серверної логіки: Бекенд-розробники створюють серверну частину, яка обробляє запити від frontend та забезпечує відповідну взаємодію з базою даних.
- Безпека та автентифікація: Бекенд-розробники впроваджують механізми автентифікації та авторизації користувачів, а також заходи безпеки для захисту даних.
- Інтеграція з базою даних: Бекенд-розробники виконують розробку та оптимізацію бази даних, забезпечуючи ефективне зберігання та отримання інформації.
- Кожна з цих частин - Frontend, Design, Backend - має свою важливу роль у реалізації платформи BusyBee. Робота цих елементів в комбінації дозволяє створити функціональну та привабливу платформу для користувачів.

Backend є важливим компонентом платформи BusyBee, оскільки він забезпечує обробку даних, безпеку та взаємодію з фронтендом. Від правильної реалізації backend залежить швидкодія платформи та безпека користувачів.

Devops

DevOps - це важлива частина команди проекту BusyBee, яка відповідає за забезпечення ефективної розробки, впровадження та управління інфраструктурою та середовищем розробки. Основні завдання DevOps включають:

- Налаштування інфраструктури: DevOps-інженери створюють та налаштовують середовища розробки, тестування та виробництва, використовуючи автоматизацію та інструменти інфраструктури як коду.
- Конфігураційне управління: DevOps-інженери використовують інструменти, такі як Ansible або Puppet, для управління конфігураціями серверів та забезпечення стабільності та складності середовища.
- Автоматизація процесів: DevOps-інженери автоматизують процеси розробки, випуску та впровадження програмного забезпечення, використовуючи інструменти, такі як Jenkins або GitLab CI/CD.
- Моніторинг та логування: DevOps-інженери встановлюють механізми моніторингу та логування, щоб відстежувати продуктивність, виявляти проблеми та забезпечувати надійну роботу платформи.

DevOps-інженери необхідні для платформи BusyBee, оскільки вони забезпечують безперебійну розробку та впровадження програмного забезпечення, автоматизоване управління інфраструктурою та оптимізацію процесів. Їхні зусилля спрямовані на забезпечення швидкої, стабільної та безпечної роботи платформи для задоволення потреб користувачів.

РОЗДІЛ 2
Реалізація DevOps

У цьому розділі описується процес реалізації DevOps-практик для проекту створення сайту busybee.space. Розглянуто основні кроки, які були здійснені для вибору хостингу, налаштування серверів, установки необхідних компонентів, налаштування автоматизації складання та розгортання, а також налаштування моніторингу та оповіщень.

1. Вимоги до проекту
2. Вибір інструментів розробки
3. Вибір хостингу в Azure та початкове налаштування серверів
4. Налаштування проксі-сервера з Nginx Proxy
5. Купівля доменного імені та налаштування DNS-записів
6. Налаштування серверів фронтенду та бекенду
7. Створення Docker-файлів та Docker Compose-файлів
8. Налаштування сервера Jenkins та CI/CD
9. Налаштування сервера моніторингу та оповіщень

Вимоги до проекту

При розробці проекту busybee вимоги до devops-інженера включали такі аспекти:

- Хостинг: Потрібно вибрати та налаштувати хостинг-провайдера для розміщення інфраструктури та програм. Хостинг повинен забезпечувати високу доступність, масштабованість та надійність, а також надавати необхідні можливості для розгортання та управління інфраструктурою.
- DNS: Потрібно налаштувати DNS для реєстрації домену busybee.space та його піддоменів, а також для зв'язку з IP-адресами серверів. DNS повинен забезпечувати коректну та надійну роздільну здатність імен та перенаправлення трафіку на відповідні сервери.
- Безперервна інтеграція та розгортання (CI/CD): Потрібно налаштувати систему безперервної інтеграції та розгортання, щоб автоматично збирати, тестувати та розгортати програми після кожного оновлення коду.

- Моніторинг та аналітика: Потрібно налаштувати систему моніторингу, яка відстежуватиме стан інфраструктури, додатків та сервісів. Також необхідно передбачити аналітичні інструменти для отримання корисної інформації із зібраних даних.

Вибір інструментів розробки

У проекті busybee були обрані наступні інструменти, необхідні для ефективної роботи інженера девопс:

- Docker: Docker дозволяє створювати та керувати контейнерами, забезпечуючи ізоляцію та переносимість додатків. Він дозволяє розробникам та операторам працювати в однорідному середовищі та спрощує процес розгортання додатків.
- Jenkins: Jenkins є популярним інструментом для безперервної інтеграції та розгортання (CI/CD). Він автоматизує процеси складання, тестування та розгортання додатків, забезпечуючи швидкий зворотний зв'язок розробникам та полегшуючи управління кодовою базою.
- Grafana: Grafana надає можливості візуалізації даних та створення дашбордів. Він дозволяє моніторити та аналізувати стан інфраструктури та додатків за допомогою графіків, діаграм та повідомлень.
- Prometheus: Prometheus - це система моніторингу та оповіщень, яка збирає та зберігає метрики від різних компонентів інфраструктури та додатків. Він забезпечує масштабованість, гнучкість і можливість створення запитів користувача для аналізу даних.
- Prometheus Exporters: Prometheus Exporters надають можливість збору метрик від різних систем та сервісів, таких як Node Exporter для моніторингу хостових систем та Jenkins Exporter для збору метрик від сервера Jenkins. Вони дозволяють розширити можливості моніторингу та збирати дані про стан різних компонентів системи.
- Nginx Proxy: Nginx Proxy є проксі-сервером, який перенаправляє трафік на інші сервери. Він використовується для балансування навантаження, забезпечення безпеки та налаштування SSL-сертифікатів для сайту busybee.space.

- Certbot: Certbot – це інструмент для автоматичної генерації та оновлення SSL-сертифікатів за допомогою Let's Encrypt. Він дозволяє забезпечити безпечне з'єднання з сайтом та автоматично продовжувати сертифікати.

Вибір даних інструментів обумовлений їхньою функціональністю, популярністю, гнучкістю та можливістю інтеграції між собою. Вони дозволяють ефективно керувати інфраструктурою, автоматизувати процеси та забезпечувати надійність, масштабованість та безпеку проекту busybee.space.

Вибір хостингу в Azure та початкове налаштування серверів

В рамках дипломної роботи було проведено вибір хостингу для проекту busybee.space та налаштовано початкову інфраструктуру за допомогою сервісу Azure. Вибір Azure був зроблений з урахуванням його широкого функціоналу, масштабованості та надійності.

Для ефективного розгортання додатка було створено п'ять серверів на платформі Azure з різними ролями: reverse proxy server, CI/CD server, monitoring server, backend server та frontend server. Кожен сервер виконує свої унікальні функції, що сприяють стабільності та ефективності роботи системи.

Reverse Proxy Server:

Цей сервер виконує роль проксі-сервера і забезпечує розподіл доступу до всіх інших серверів. Він приймає всі вхідні запити від клієнтів і маршрутизує їх на відповідний сервер залежно від запиту.

CI/CD Server:

Цей сервер відповідає за процес неперервної інтеграції та доставки (CI/CD). На ньому налаштовано автоматичне збірки та деплоя проектів. Він дозволяє здійснювати автоматичні оновлення інфраструктури та коду додатка безпосередньо з репозиторію.

Monitoring Server:

Цей сервер відповідає за моніторинг та логування системи. Він забезпечує постійний нагляд за працездатністю та продуктивністю системи, збирає метрики процесора, пам'яті, мережі та інші параметри. Також він здійснює аналіз логів для виявлення можливих проблем та помилок.

Backend Server:

Цей сервер виконує функції обробки даних та бізнес-логіки. Він відповідає за забезпечення взаємодії з базою даних, обробку запитів користувачів та надання результатів frontend server.

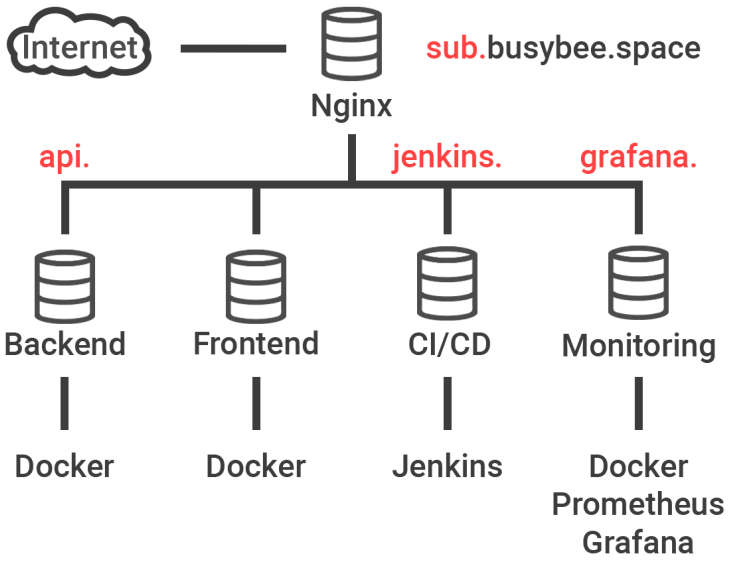
Frontend Server:

Цей сервер відповідає за відображення та взаємодію з користувачами. Він надає доступ до веб-інтерфейсу, який дозволяє користувачам взаємодіяти з додатком та отримувати необхідну інформацію.

Крім створення цих серверів, було налаштовано мережеві правила, файрвол та доступ до відповідних ресурсів. Це забезпечує безпеку та обмежує доступ до серверів лише для авторизованих користувачів. Завдяки Azure було досягнуто високої доступності, масштабованості та стабільності інфраструктури проекту busybee.space. Однак, надалі можна провести подальші налаштування та оптимізацію, щоб забезпечити ще більшу продуктивність та надійність системи.

Налаштування проксі-сервера з Nginx Proxy

На цьому етапі слід створити кореневий сервер, який розподілятиме весь вхідний трафік по внутрішніх серверах, на наступному слайді спрощена ілюстрація інфраструктури, де наочно зображено як nginx проксує трафік по серверах.



Для реалізації такого підходу, після встановлення nginx, слід налаштувати конфігураційні файли для кожного сервера. Очевидно, що доступ до серверів буде через субдомени:

- Backend – api.busybee.space/
- Frontend – busybee.space/
- CI/CD – jenkins.busybee.space/
- Monitoring – grafana.busybee.space/

Backend

Конфігураційний файл:

```
server {
    listen 80;
    server_name api.busybee.space;
    return 301 https://$server_name$request_uri;
}
server {
    listen 443 ssl;
    server_name api.busybee.space;

    ssl_certificate /etc/letsencrypt/live/busybee.space/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/busybee.space/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    access_log /var/log/nginx/api.busybee.access.log;
    error_log /var/log/nginx/api.busybee.error.log;

    error_page 404 /404.webp;
    location = /404.webp {
        root /var/www/images/;
        internal;
    }
    location / {
        proxy_pass http://192.168.0.100:80/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

При спробі клієнта звернутися на 80 порт (http) відбувається перенаправлення на https. Також визначено файли access і error логів, сторінка 404 і проксування запиту на потрібну ip-адресу backend сервера, де хоститься відповідний контейнер.

Для коректної роботи https слід налаштувати ssl сертифікат за допомогою Let's Encrypt та Certbot:

```
certbot certonly --manual --preferred-challenges=dns --email
admin@busybee.space --server https://acme-v02.api.letsencrypt.org/directory
--agree-tos -d busybee.space -d *.busybee.space
```

Після виконання команди на сервері DNS був створений TXT файл _acme-challenge с значенням необхідним виконаною командою.

```
_acme-challenge   IN    TXT   3EV1ULmZw78IE8oRoCu8gLq4d-FDyp0DJ6zvNg47suQ
```

А також налаштовано автоподовження сертифіката додаванням наступної команди до планувальника cron:

```
0 1 * * * /usr/bin/certbot renew >> /var/log/letsencrypt/renew.log
```

Frontend

Конфігураційний файл:

```
server {
    listen 80;
    server_name busybee.space;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name busybee.space;
    ssl_certificate /etc/letsencrypt/live/busybee.space/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/busybee.space/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    access_log /var/log/nginx/busybee.access.log;
    error_log /var/log/nginx/busybee.error.log;

    error_page 404 /404.webp;
    location = /404.webp {
        root /var/www/images/;
        internal;
```

```
    }

    location / {
        proxy_pass http://192.168.0.101:80/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /images/ {
        alias /var/www/images/;
        autoindex on;
    }

    location /files/ {
        alias /var/www/files/;
        autoindex on;
    }
}
```

У випадку з Frontend конфігурація доповнена location блоками, в яких, як було прийнято в команді, будуть розміщені статичні файли мінімального обсягу, для яких наявність хмарного сховища не потрібно.

CI/CD

Конфігураційний файл:

```
server {
    listen 80;
    server_name jenkins.busybee.space;
    return 301 https://$server_name$request_uri;
}

server {

    listen 443 ssl;
    server_name jenkins.busybee.space;

    ssl_certificate /etc/letsencrypt/live/busybee.space-0001/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/busybee.space-0001/privkey.pem;
```

```
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    access_log /var/log/nginx/jenkins.busybee.access.log;
    error_log /var/log/nginx/jenkins.busybee.error.log;

    error_page 404 /404.webp;
    location = /404.webp {
        root /var/www/images/;
        internal;
    }

    location / {
        proxy_pass http://192.168.0.123:8080/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Monitoring

Конфігураційний файл:

```
server {
    listen 80;
    server_name grafana.busybee.space;
    return 301 https://$server_name$request_uri;
}

server {

    listen 443 ssl;
    server_name grafana.busybee.space;

    ssl_certificate /etc/letsencrypt/live/busybee.space-0001/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/busybee.space-0001/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    access_log /var/log/nginx/grafana.busybee.access.log;
    error_log /var/log/nginx/grafana.busybee.error.log;
```

```
    error_page 404 /404.webp;
    location = /404.webp {
        root /var/www/images/;
        internal;
    }

    location / {
        proxy_pass http://192.168.0.127:3000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Купівля доменного імені та налаштування DNS-записів

Для проекту створення сайту busybee.space необхідно було придбати доменне ім'я та налаштувати DNS-записи, щоб встановити зв'язок між доменом та IP-адресами серверів. Для цього було обрано веб-сервіс ukraine.com.ua для купівлі домену busybee.space.

Купівля доменного імені:

Після переходу на веб-сайт ukraine.com.ua були перевірені різні варіанти на доступність, з них найоптимальнішим і найдоступнішим за мінімального бюджету виявився busybee.space. Після заповнення необхідних даних придбання домену завершено.

Налаштування DNS-записів:

После покупки доменного имени, стали доступны панели управления DNS-записями и произведены следующие настройки:

```
$ORIGIN busybee.space
$TTL 900
@     IN    A     78.26.204.195
www   IN    A     78.26.204.195
@     IN    MX    0 mx.ukraine.com.ua.
@     IN    TXT   v=spf1 include:_spf.ukraine.com.ua ~all
api   IN    A     78.26.204.195
jenkins     IN    A     78.26.204.195
grafana     IN    A     78.26.204.195
_acme-challenge   IN    TXT   3EV1ULmZw78IE8oRoCu8gLq4d-FDyp0DJ6zvNg47suQ
```

Були налаштовані записи A для субдоменів, відповідно до значень, які ми вказували на попередньому етапі в конфігураційних файлах наших серверів, а також TXT запис для ssl сертифіката. Після створення та збереження DNS-записів та очікування їх поширення сайт став доступним за доменним ім'ям.

Таким чином, після покупки доменного імені на ukraine.com.ua та налаштування DNS-записів, зв'язок між доменом busybee.space та IP-адресами серверів, необхідних для роботи проекту, був успішно встановлений.

Налаштування серверів Frontend та Backend

Після встановлення docker на обох серверах та клонуванні репозиторіїв проектів слід їх зібрати в docker-контейнери. Було складено короткий план дій:

- Створення файлу docker
- Складання та пуш образу на DockerHub
- Створення docker-compose файлу та запуск

При створенні docker-файлу необхідно врахувати всі залежності, оголосити порти, так само коректно виконати необхідні команди, необхідні для коректної роботи проекту в необхідних робочих директоріях:

Створення Backend docker-файлу

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["BusyBee.Api/BusyBee.Api.csproj", "BusyBee.Api/"]
COPY ["BusyBee.Persistence.Design/BusyBee.Persistence.Design.csproj",
"BusyBee.Persistence.Design/"]
COPY ["BusyBee.Persistence/BusyBee.Persistence.csproj",
"BusyBee.Persistence/"]
COPY ["BusyBee.Core/BusyBee.Core.csproj", "BusyBee.Core/"]
RUN dotnet restore "BusyBee.Api/BusyBee.Api.csproj"
COPY . .
WORKDIR "/src/BusyBee.Api"
RUN dotnet build "BusyBee.Api.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "BusyBee.Api.csproj" -c Release -o /app/publish
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "BusyBee.Api.dll"]
```

Спочатку визначається базовий образ, який буде використовуватися як основа для поточного образу. У разі використовується образ "aspnet" версії 7.0 від Microsoft. Далі встановлюємо робочу директорію /app усередині контейнера, де виконуватимуться наступні команди. Оголошуємо порти, які контейнер прослуховуватиме при запуску. У цьому випадку оголошено порти 80 та 443.

Далі визначається другий образ, що використовується для етапу збирання (build). У цьому випадку використовується образ "SDK" версії 7.0 від Microsoft, який містить інструменти для компіляції та складання програми на платформі .NET. Встановлюється робоча директорія /src для наступних команд усередині контейнера. Далі йдуть вказівки копіювання файлів проектів .csproj з локальної файлової системи всередину контейнера. Вони вказують шляхи до файлів проектів щодо контексту складання. Команда "dotnet restore" виконується для відновлення залежностей проекту. Команда dotnet restore завантажує та встановлює всі залежності, вказані у файлі проекту .csproj. Копіюємо всі інші файли з поточного контексту збирання всередину контейнера. Встановлюємо нову робочу директорію "/src/BusyBee.Api". Далі виконується команда dotnet build для збирання проекту. Опції -c Release -o /app/build вказують на складання проекту в режимі Release та збереження результатів складання в директорії /app/build усередині контейнера.

Створюється новий образ, який використовуватиметься для етапу публікації publish. Він базується на попередньому образі збирання build. Виконуємо команду dotnet publish, яка публікує додаток, створюючи файли, що виконуються, і всі необхідні файли для його виконання. Опції -c Release -o /app/publish вказують на публікацію програми в режимі Release та збереження результатів публікації в директорії /app/publish усередині контейнера.

Створюється новий образ final, який буде використовуватися як фінальний образ для запуску програми. Він базується на базовому образі base, визначеному на початку Dockerfile. Встановлюється робоча директорія /app для наступних команд усередині контейнера. Копіюємо файли з попереднього образу з публікацією publish усередину поточного фінального образу. Опція --from=publish вказує на те, що джерелом копіювання є контейнер з ім'ям

"publish". І нарешті задаємо точку входу для контейнера. При запуску контейнера буде виконано команду dotnet BusyBee.Api.dll, яка запускає виконуваний файл програми всередині контейнера.

Таким чином, цей Dockerfile описує процес складання, публікації та запуску програми на платформі .NET у контейнері Docker.

Створення Frontend docker-файлу

```
FROM node:latest AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build --prod
FROM nginx:latest
COPY --from=builder /app/dist/busy-bee/ /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Визначаємо базовий образ, який буде використовуватися як основа для етапу складання builder. В даному випадку використовується образ "node" з останньою версією. Встановлюємо робочу директорію /app усередині контейнера, де виконуватимуться наступні команди.

Копіюємо файли package.json та package-lock.json із локальної файлової системи всередину контейнера. Вони необхідні для встановлення залежностей проекту. Встановлюємо залежність проекту. Командою npm install завантажуємо та встановлюємо всі необхідні пакети Node.js, вказані у файлах package.json та package-lock.json. Копіюємо всі інші файли та папки з поточного контексту збирання всередину контейнера.

Виконуємо команду збирання програми. Команда npm run build запускає скрипт складання, вказаний у файлі package.json, з опцією --prod, яка говорить про складання в режимі production.

Створюється новий образ, який буде використовуватися як основа для фінального образу. У разі використовується образ "nginx" останньої версією.

Копіюємо файли вихідного коду програми з попереднього способу складання builder всередину контейнера з nginx. Файли вихідного коду розміщуються у директорії /usr/share/nginx/html, яка є стандартною директорією для роздачі статичного контенту nginx. Створюємо простий конфігураційний файл nginx усередині проекту:

```
server {
  listen 80;
  server_name localhost;
  root /usr/share/nginx/html;
  index index.html;

  location / {
    try_files $uri $uri/ /index.html;
  }
}
```

Копіюємо файл конфігурації nginx з локальної файлової системи всередину контейнера. Файл конфігурації nginx розміщується в директорії /etc/nginx/conf.d/default.conf, де nginx шукатиме свою конфігурацію під час запуску.

Таким чином, цей Dockerfile описує процес складання та розгортання статичної веб-додатки з використанням Node.js та сервера nginx у контейнері Docker.

Складання та пуш образу на DockerHub

Виконуємо складання Docker-образу з Dockerfile, що знаходиться в директорії. Опція -t loldc1/busybee.api:latest задає тег для створюваного образу. Тег використовується для ідентифікації та звернення до образу:

```
docker build -t loldc1/busybee.api:latest .
```

Виконуємо аутентифікацію в Docker-реєстрі із зазначеним ім'ям користувача. Вона потрібна для доступу до приватних Docker-реєстрів, куди буде завантажено створений образ:

```
docker login --username loldc1
```

Завантажуємо Docker-образ з тегом latest в Docker-реєстр, який пов'язаний з ім'ям користувача/репозиторія loldc1:

```
docker push loldc1/busybee.api:latest
```

Створення docker-compose файлу та запуск

Ми використовуємо docker compose тому, що з його допомогою можна зібрати один файл, в якому описуються всі контейнери, структуровано визначаються всі необхідні команди для запуску контейнерів. А також через можливість збирати, зупиняти та запускати docker-файли однією командою.

Backend:

```
version: "3.9"

services:
  busybee_api:
    image: loldc1/busybee.api
    container_name: busybee.api
    restart: always
    volumes:
      - /var/www/images:/app/images
      - /var/www/files:/app/files
    depends_on:
      - postgres_db
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      -
ConnectionStrings:DefaultConnection=Host=postgres;Database=busy_bee_db;User
name=postgres;Password=******
      - AzureAdB2C:Instance=https://busybeeua.b2clogin.com
```

```
      - AzureAdB2C:Domain=busybeeua.onmicrosoft.com
      - AzureAdB2C:ClientId=44d3647b-fe43-4b84-bc50-7b99f37fd8f2
      - AzureAdB2C:SignedOutCallbackPath=/signout/B2C_1_SignupSignin
      - AzureAdB2C:SignUpSignInPolicyId=B2C_1_SignupSignin
      - AzureAdB2C:OpenApiClientId=0e271133-d0bf-493a-976d-8146db6770a4
      -
AzureAdB2C:OpenApiScopeApi=https://busybeeua.onmicrosoft.com/api/main
      - LocalStorage:ImagesUrlPrefix=/images/
      - LocalStorage:ImagesFolderPath=images
      - LocalStorage:FilesUrlPrefix=/files/
      - LocalStorage:FilesFolderPath=files
      - LocalStorage:CategoryIconFilenamePrefix=category_icon_
      - LocalStorage:UserPhotoFilenamePrefix=user_photo_
      - LocalStorage:PortfolioFilenamePrefix=portfolio_
    ports:
      - "192.168.0.99:80:80"

  postgres_db:
    image: postgres:latest
    container_name: postgres
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=12345678
    ports:
      - "5432:5432"
```

Визначаємо список сервісів, у нашому випадку сервіс з іменем "busybee_api" та "postgres_db". Для кожного вказуємо Docker-образ, який буде використаний для створення контейнера, у разі сервісу "busybee_api" буде використано образ з ім'ям "loldc1/busybee.api". У разі сервісу "postgres_db" буде використано образ з ім'ям "postgres: latest". Визначаємо ім'я контейнера, яке використовуватиметься для відповідного сервісу. Вказуємо автоматичний перезапуск контейнера. Визначаємо прив'язку томів для контейнера "busybee_api". Вони пов'язують директорії на локальній машині (/var/www/images та /var/www/files) з директоріями всередині контейнера (/app/images та /app/files). Це дозволяє зберігати та обмінюватися даними між локальною машиною та контейнером.

Далі "depends_on", що визначає залежність сервісу "busybee_api" від сервісу "postgres_db". Це гарантує, що сервіс postgres_db буде запущений перед сервісом busybee_api.
Далі визначаються змінні оточення, які будуть доступні усередині контейнерів. Усі перелічені змінні оточення будуть встановлені зі значеннями, вказаними у файлі.
Прокидаємо порти між хостом та контейнером. В даному випадку порт 80 nginx проксі-сервера буде прив'язаний до порту 80 всередині контейнера "busybee_api". Так ми змушуємо наш контейнер слухати лише поводження з нашого проксі-сервера. У випадку "postgres_db" залишаємо параметри за замовчуванням.
Таким чином, цей docker-compose файл визначає два сервіси: "busybee_api" та "postgres_db". Він зв'язує контейнери з відповідними образами, встановлює залежність між сервісами, налаштовує змінні оточення та прив'язує порти для доступу до сервісів ззовні контейнерів.

Frontend:

У випадку з Frontend docker-compose нам не потрібний, оскільки програма складається всього з одного контейнера, а набір команд, необхідний для запуску контейнера, досить обмежений.
Завантажуємо образ з Docker Hub, і запускаємо контейнер у фоновому режимі, вказуючи прокидання портів, опцію автоматичного перезапуску, ім'я образу та ім'я контейнера:

```
docker pull loldc1/busybee.web
docker run -dit -p 192.168.0.99:80:80 --restart=always --name
loldc1/busybee.web busybee.web
```

Налаштування сервера Jenkins та CI/CD

При розробці проекту створення сайту busybee.space важливою частиною процесу devops є налаштування сервера Jenkins і CI/CD (Continuous Integration/Continuous Deployment). Цей розділ представляє детальний план дій для налаштування Jenkins, створення завдання, налаштування тригера, складання проекту та налаштування деплою.

План дій:

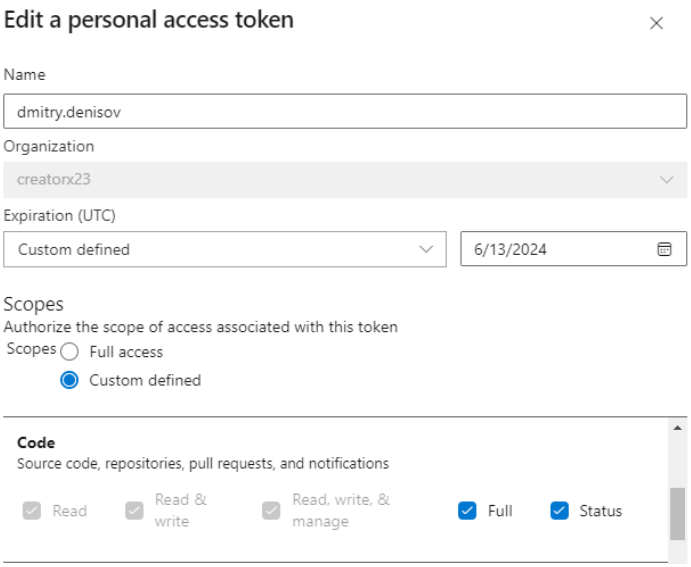
- Створення завдань:
    - Створення завдань Jenkins, які будуть виконувати CI/CD процес для конкретного проекту.
    - Створення PAT, вказати джерело коду проекту, репозиторій Azure Repos, вказати параметри доступу до репозиторію.
- Налаштування тригера:
    - Вибір тип тригера. Наприклад, за розкладом або тригером при кожній зміні коду в репозиторії webhook.
    - Встановлення та налаштування необхідних параметрів для вибраного тригера.
- Налаштування складання проекту:
    - Визначення кроків складання, які мають виконуватися під час кожного запуску завдання.
    - Налаштування середовища виконання, устазати залежності та параметри складання проекту.
- Налаштування деплою:
    - Установка плагіна "Publish Over SSH" для Jenkins, який дозволить розгорнути збирання на потрібному сервері.
    - У налаштуваннях завдань додати крок деплою, вказавши сервер та дані для доступу до SSH.
    - Визначення кроків деплою, які мають виконуватися після кожного завершення етапу складання проекту.
- Перевірка та тестування:
    - Збереження та перевірка процесу складання та деплою.

Создание задачи

Після початкового налаштування основних параметрів Jenkins приступив до створення перших freestyle завдань нашого проекту, окремо для frontend, окремо для backend. Кожна з них має суворо виконувати певний набір завдань під час реалізації складання та деплою проекту

При створенні завдань було їм задано ім'я, попередньо обмежена кількість записів, що зберігаються, про завершені білди до п'яти. Насамперед потрібно вказати джерело коду проекту, репозиторій Azure Repos.

Щоб отримати доступ до нього потрібно попередньо згенерувати PAT (personal access token) в налаштуваннях облікового запису Azure, визначивши повні дозволи доступу до даних репозиторію (read, write, manage, status) і налаштувавши закінчення терміну на певну дату, в даному випадку на рік. Це потрібне для забезпечення доступу до репозиторію.



Коли PAT був згенерований у завданнях, був налаштований доступ до певного репозиторію, а також визначена конкретна гілка, з якої клонуватиме проект наше завдання, у нашому випадку "master" в обох випадках.

Налаштування тригера

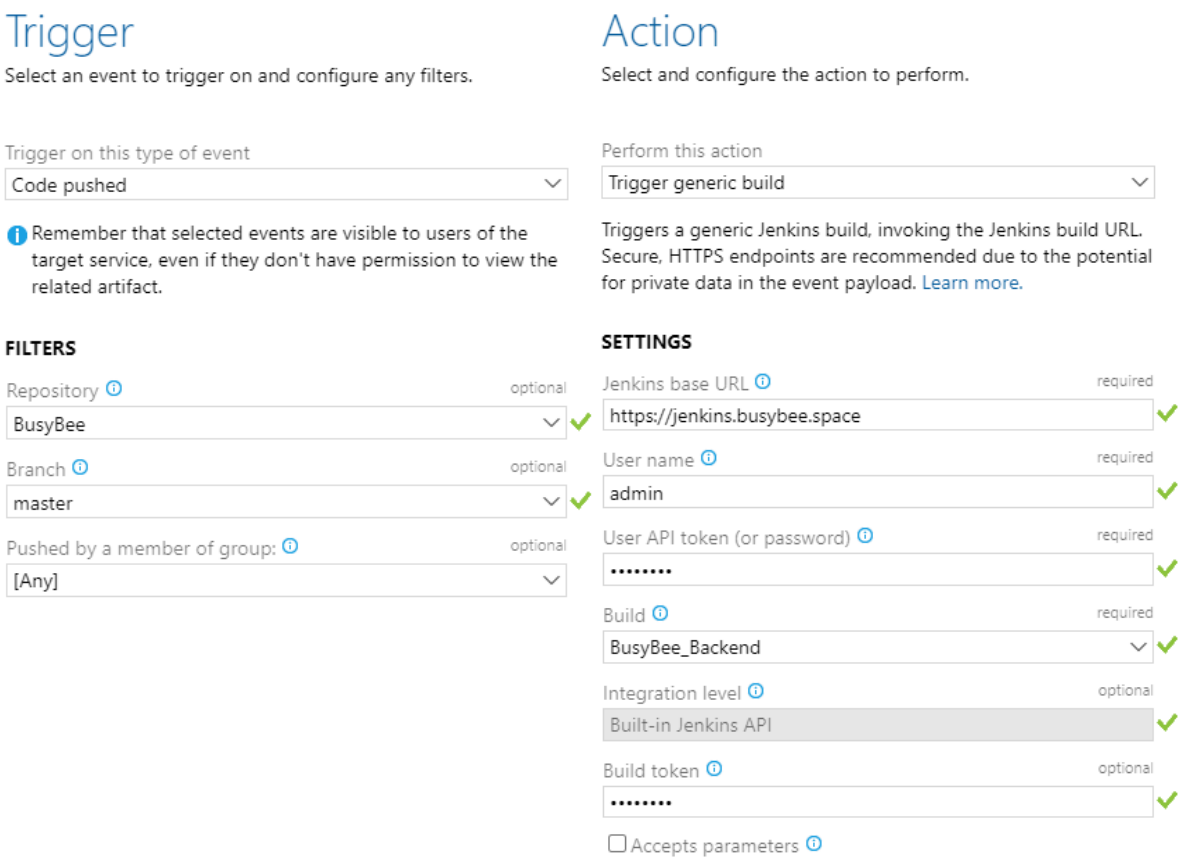
На наступному етапі потрібно було налаштувати тригер, у нашому випадку найефективніше використовувати webhook тригер, який тригеритиме наш репозиторій при кожному оновленні проекту.

Насамперед у секції Trigger builds remotely – Authentication Token був створений токен, цей токен визначає ім'я запиту нашого webhook. Після отримання нашого посилання на наш тригер вона була визначена в Azure repos:

Ім'я проекту – Project settings – General – Service hooks, було створено Service hook для кожного завдання відповідно. На прикладі backend:

У секції Trigger було визначено умову для тригера (Code pushed), ім'я необхідного репозиторію та гілка.

У секції Action було визначено дію для тригера, задали URL Jenkins, дані для входу, завдання.



Налаштування складання проекту

Автоматизація складання проекту полягає у простій послідовності команд, які виконувались при першій спробі хоста наших контейнерів. Були написані скрипти із послідовностей команд виконуваних раніше, для кожної із завдань, розглянемо на прикладі Backend:

Оголошуємо глобальні змінні:

```
echo "BUILDING THE PROJECT"

DOCKER_IMAGE_NAME="busybee.api"
DOCKER_TAG="latest"
DOCKER_USERNAME="loldc1"
DOCKER_PASSWORD="*****"
DOCKER_IMAGE_NAMETAG="${DOCKER_USERNAME}/${DOCKER_IMAGE_NAME}"
```

Так як ми процес складання автоматизуємо мається на увазі, що при кожній збірці старі версії будуть залишатися, тому необхідно цю проблему заздалегідь вирішити. Була створена маленька умова, яка спочатку перевіряє список усіх образів на наявність образів з необхідним ім'ям і якщо знаходить превентивно їх видаляє:

```
echo "Removing old images..."
if [[ $(docker images -a | grep $DOCKER_IMAGE_NAME | awk '{print $3}') ]];
then
      docker rmi -f $(docker images -a | grep $DOCKER_IMAGE_NAME | awk
'{print $3}')
fi
echo "Done!"
```

Складання контейнера:

```
echo "Building Docker container..."
docker build -t $DOCKER_IMAGE_NAMETAG:$DOCKER_TAG .
echo "Done!"
```

Виконали аутентифікацію та завантажуємо образ на Docker Hub:

```
echo "Logging to DockerHub..."
echo $DOCKER_PASSWORD | docker login --username $DOCKER_USERNAME
--password-stdin
echo "Done!"

echo "Pushing project to DockerHub..."
docker push $DOCKER_IMAGE_NAMETAG:$DOCKER_TAG
echo "Done!"

echo "PROJECT HAS BEEN BUILT"
```

Цей скрипт практично повністю ідентичний скрипту Frontend завдання, за винятком іншої назви контейнера змінної DOCKER_IMAGE_NAME.

Налаштування деплою

На даному етапі було прийнято рішення деплоїти образ за допомогою плагіна Publish Over SSH, оскільки його функціонал повністю відповідав моєму уявленню про те, як я буду деплоїти образи. Publish Over SSH дозволяє виконувати команди і передавати файли по ssh, що відмінно підходить нашому проекту, оскільки крім запуску контейнерів потрібно ще передати docker-compose файл на необхідний сервер.

Для роботи з плагіном в системній конфігурації були визначені SSH сервери, де були налаштовані: ім'я користувача, ip-адреса, директорія, пароль. Після перевірки на з'єднання з серверами можна повертатися в конфігурацію завдання.

Backend

Для деплоя Backend проекта, сперва передается docker-compose файл, в секции Transfer Set – Source files задано значение "**/docker-compose.yml", в секции Transfer Set – Remote directory задано значение "jenkins_remote_dir", в конечном итоге на сервере появится соответствующий файл в директории "jenkins_remote_dir".

Далі запуск контейнера. Для цього етапу використовується невеликий скрипт. Спершу оголошуються змінні які парять docker-compose файл, дістаючи з них назву образу та контейнера. Потрібно це для того, щоб позбутися існуючих образів і контейнерів з такою самою назвою:

```
echo "STARTING DEPLOYMENT THE PROJECT"
IMAGE_NAME=$(cat ~/jenkins_remote_dir/docker-compose.yml | grep image | awk
'{print $2}' | tr -d '\r')
CONTAINER_NAME=$(cat ~/jenkins_remote_dir/docker-compose.yml | grep
container_name | awk '{print $2}' | tr -d '\r')
```

Зупинка та видалення контейнера, якщо він уже був запущений:

```
echo "Removing old containers..."
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME
docker rmi $IMAGE_NAME
echo "Done!"
```

Активація контейнера через файл docker-compose:

```
echo "Docker-compose up..."
docker-compose -f ~/jenkins_remote_dir/docker-compose.yml up -d
echo "Done!"

echo "PROJECT HAS BEEN DEPLOYED"
```

Frontend

У випадку з Frontend нам не потрібно було передавати файли docker-compose, тому був створений наступний скрипт:

```
echo "STARTING DEPLOYMENT THE PROJECT"
IMAGE_NAME="loldc1/busybee.web"
CONTAINER_NAME="busybee.web"

echo "Removing old containers..."
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME
docker rmi $IMAGE_NAME
echo "Done!"
```

Завантаження образу з Docker Hub та запуск контейнера:

```
echo "Pulling the image from dockerhub..."
docker pull $IMAGE_NAME
echo "Done!"

echo "Docker run..."
docker run -dit -p 192.168.0.99:80:80 --restart=always --name
$CONTAINER_NAME $IMAGE_NAME

echo "PROJECT HAS BEEN DEPLOYED"
```

Налаштування сервера моніторингу та оповіщень

Моніторинг є невід'ємною частиною розробки та підтримки будь-якого проекту. Він дозволяє відстежувати стан системи, компонентів та ресурсів, а також виявляти потенційні проблеми та збої, що виникають у процесі роботи. Було прийнято рішення використовувати зв'язку Prometheus та Grafana. Prometheus надає гнучкі можливості для збору та зберігання тимчасових рядів метрик, а також створювати складні запити для аналізу даних.

У рамках розробки проекту busybee.space було виконано етап налаштування сервера моніторингу та оповіщень. У цьому розділі було виконано такі кроки:

- Установка Prometheus та Grafana
- Встановлення та налаштування експортерів
- Створення дашбордів у Grafana
- Налаштування алертів

Встановлення Prometheus та Grafana

Був створений docker-compose файл, в якому визначено два сервіси: prometheus і grafana. Розберемо кожну частину файлу по порядку:

Визначили два томи для використання у контейнерах. Ці томи будуть використовуватися для збереження даних Prometheus та Grafana відповідно.

```
volumes:
  prometheus-data:
    driver: local
  grafana-data:
    driver: local
```

Визначено образ Docker для сервісу Prometheus. Використовується зображення з тегом latest. Задано ім'я контейнера, прокидається порт 9090 контейнера Prometheus на порт 9090 хост-системи. Визначено примонтовані томи для контейнера Prometheus, /etc/prometheus і prometheus-data. Визначено команду, яка вказує конфігураційний файл prometheus.yml всередині контейнера. Задано автоматичний перезапуск:

```
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - 9090:9090
    volumes:
      - /etc/prometheus:/etc/prometheus
      - prometheus-data:/prometheus
    command: "--config.file=/etc/prometheus/prometheus.yml"
    restart: unless-stopped
```

Визначено образ Docker для сервісу Grafana. Використовується зображення з тегом latest. Задано ім'я контейнера, прокидається порт 3000. Визначено примонтовані томи для контейнера Prometheus, для grafana-data. Встановлено змінні середовища, які визначають URL-адресу та налаштування шляху для доступу до Grafana. Задано автоматичний перезапуск:

```
  grafana:
    image: grafana/grafana-oss:latest
    container_name: grafana
    ports:
      - 3000:3000
    volumes:
      - grafana-data:/var/lib/grafana
    environment:
      - GF_SERVER_ROOT_URL=https://grafana.busybee.space:443/
      - GF_SERVER_SERVE_FROM_SUB_PATH=true
    restart: unless-stopped
```

Перед запуском docker-compose файлу було створено конфігураційний файл "/etc/prometheus/prometheus.yml" для Prometheus, в якому визначаємо цілі (endpoints), які будуть моніторитися, та інтервали збору метрик.

Визначено інтервал, з яким Prometheus збиратиме метрики з цілей у 5 секунд. Визначено декілька завдань моніторингу. Кожне завдання має своє ім'я, інтервал збору метрик та шлях до метриків, якщо він відрізняється від значення за замовчуванням. І ціль збору, ip-адреса, порт.

Визначено чотири завдання збору метрик: Prometheus, Jenkins, Backend, Frontend:

```
global:
  scrape_interval: 5s
scrape_configs:

  - job_name: 'prometheus'
      scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'jenkins'
    scrape_interval: 5s
    metrics_path: /prometheus
    static_configs:
      - targets: ['192.168.0.123:8080']

  - job_name: 'backend'
    scrape_interval: 5s
    metrics_path: /metrics
    static_configs:
      - targets: ['192.168.0.101:9100']

  - job_name: 'frontend'
    scrape_interval: 5s
    metrics_path: /metrics
    static_configs:
      - targets: ['192.168.0.100:9100']
```

Встановлення та налаштування експортерів

Для збору метрик з різних компонентів системи були встановлені та налаштовані експортери Prometheus. На фронтенд та бекенд серверах були встановлені та налаштовані Node експортери, які надають метрики про ресурси та стан серверів. На Jenkins-сервері було встановлено та налаштовано Jenkins експортер для збору метрик про стан збірок та завдань CI/CD. Експортери були взяті з офіційного джерела Prometheus.

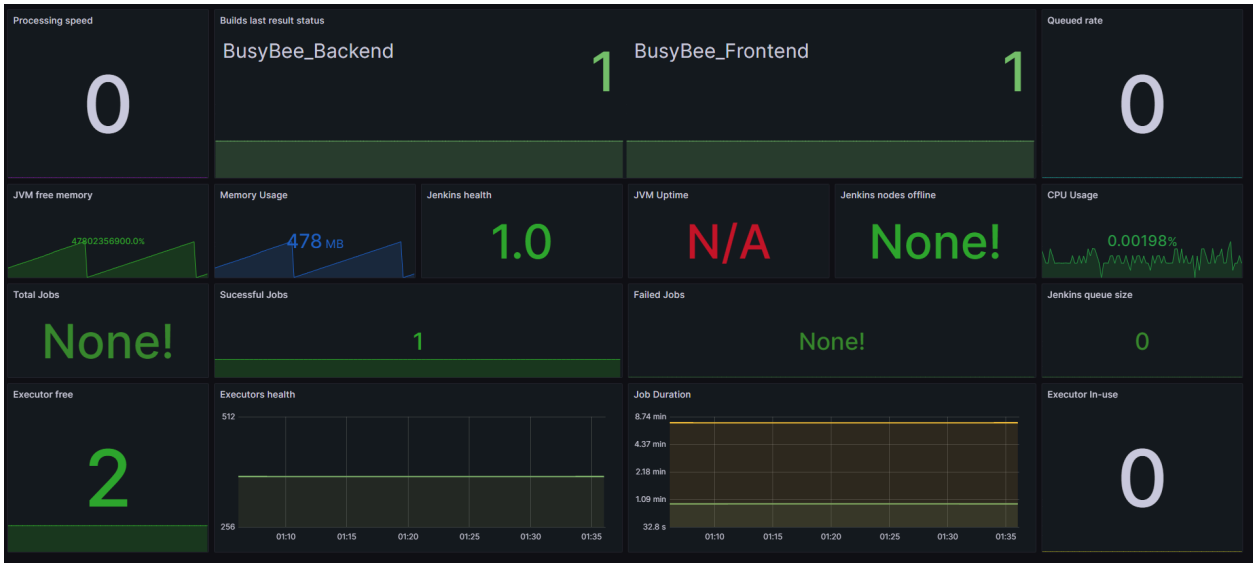
Створення дашбордів у Grafana

На офіційному сайті Grafana існує розділ з дашбордами, з якого було ухвалено рішення імпортувати такі дашборди:

- Node Exporter Full
- Jenkins: Performance and Health Overview

На сторінках даних дашбордів було взято ID і вказано при імпортуванні нового дашборду в Grafana, так само ім'я, папка і UID. Наступним чином виглядають дашборди:

Jenkins

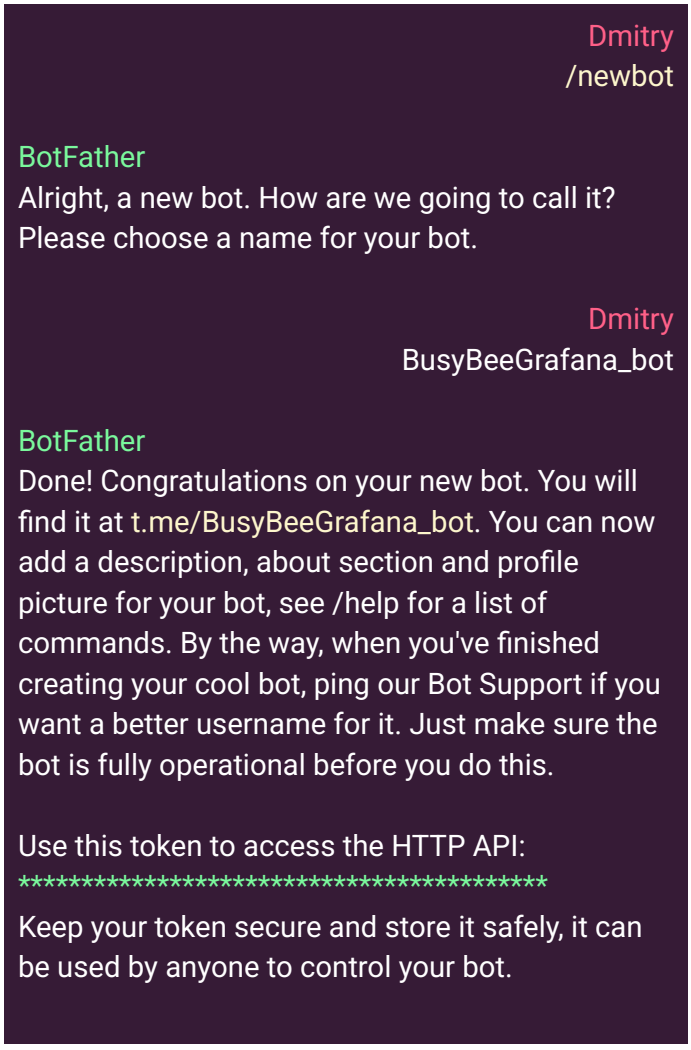


Backend

Налаштування алертів

Для моніторингу та оповіщення про проблеми та події були налаштовані алерти у Prometheus. За допомогою мови запитів PromQL можна визначити правила алертів, які активуються за певних умов. Наприклад, можна створити правило, яке спрацьовує, коли завантаження процесора перевищує певний поріг або кількість помилок API-запитів перевищує заданий рівень.

Для оповіщення про спрацьовування алертів було налаштовано бот у Telegram. Таким чином, при виникненні проблеми алерт відправляється в чат або групу Telegram, щоб адміністратори могли швидко реагувати на проблеми.
За допомогою використання бота @BotFather і відправлення йому наступного повідомлення було створено новий бот і отримано HTTP API токен:



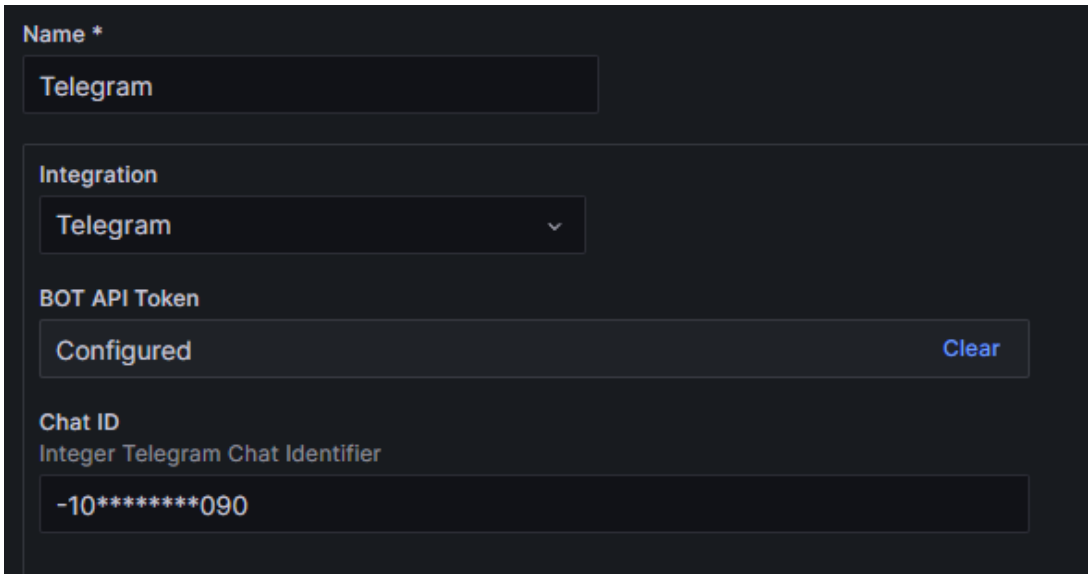
Токен нам потрібен, щоб створити канал і отримати ідентифікатор чату каналу.

Було створено канал у телеграмі BusyBeeMonitoring. Доданий @BotFather в цей канал як адміністратор і написано рандомне текстове повідомлення, це потрібно для того, щоб @BotFather оновив своє апі з даними даного каналу, в яких ми зможемо знайти наш ідентифікатор чату каналу.

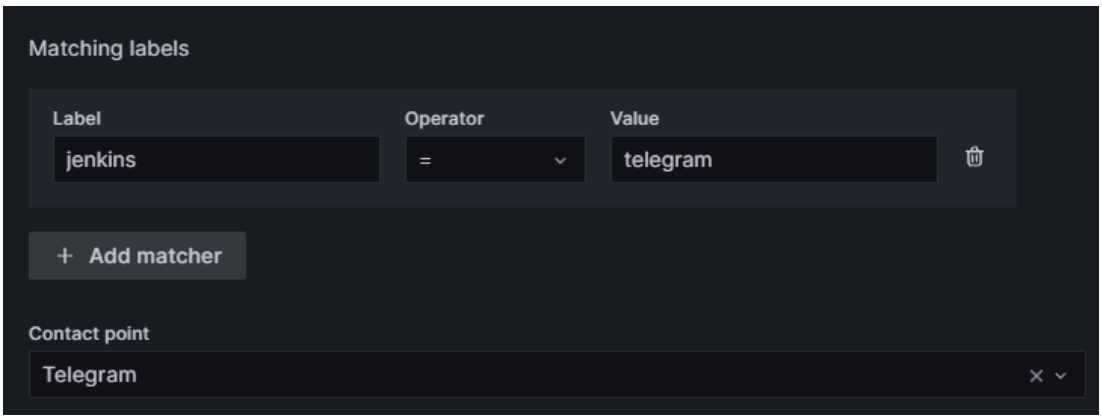
Після переходу по https://api.telegram.org/bot<ОТРИМАНИЙ API ТОКЕН>/getUpdates було отримано схоже повідомлення:


```
{"ok":true,"result":[{"update_id":1112223334445,
"channel_post":{"message_id":1,"chat":{"id":-<ОТРИМАНИЙ ID ЧАТА>,
"title":"ShelleyMonitoring","type":"channel"},"date":1576534122,"text":"/bo
t","entities":[{"offset":0,"length":4,"type":"bot_command"}]}}]}
```

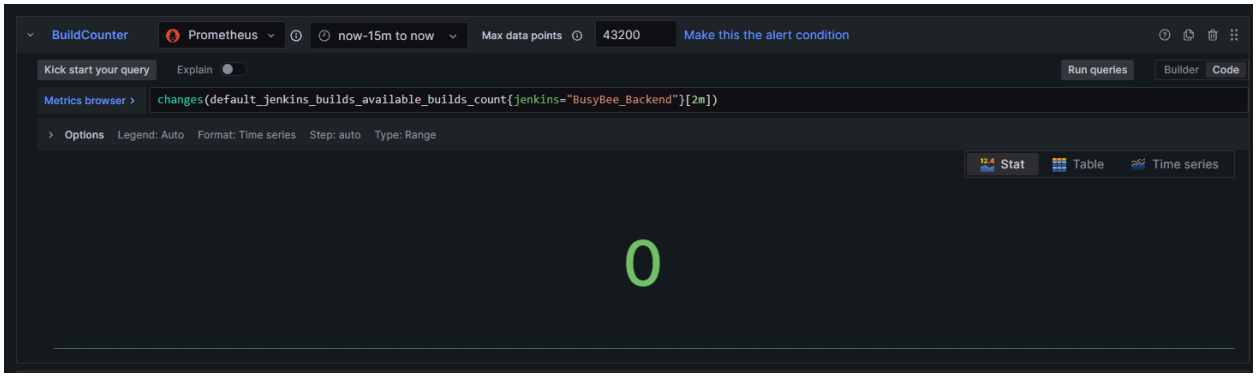

З отриманими даними була налаштована контактна точка з телеграмом в Grafana на шляху: Home – Alerting – Contact points – Telegram. Внесены данные HTTP API токену та ID чата:



Зайшовши в Home – Alerting – Notification policies була налаштована політика повідомлень "jenkins = telegram" яка була пов'язана з нашим телеграм контактною точкою, це потрібно для того, щоб алерти знали куди саме їм потрібно надсилати повідомлення:

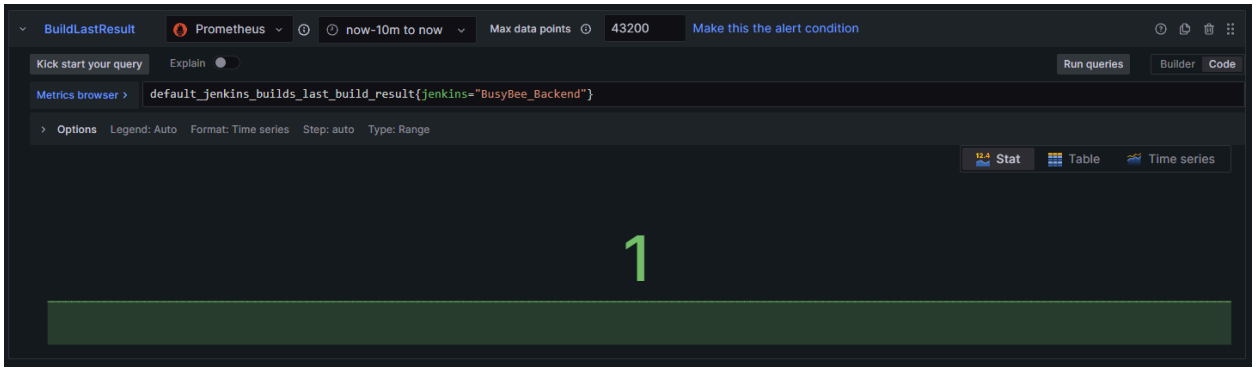


На цьому етапі зв'язок з телеграмом встановлений і можна переходити до алертів. Було створено два Jenkins алерту як для Frontend, так для Backend. На прикладі Backend розглянемо алерт:

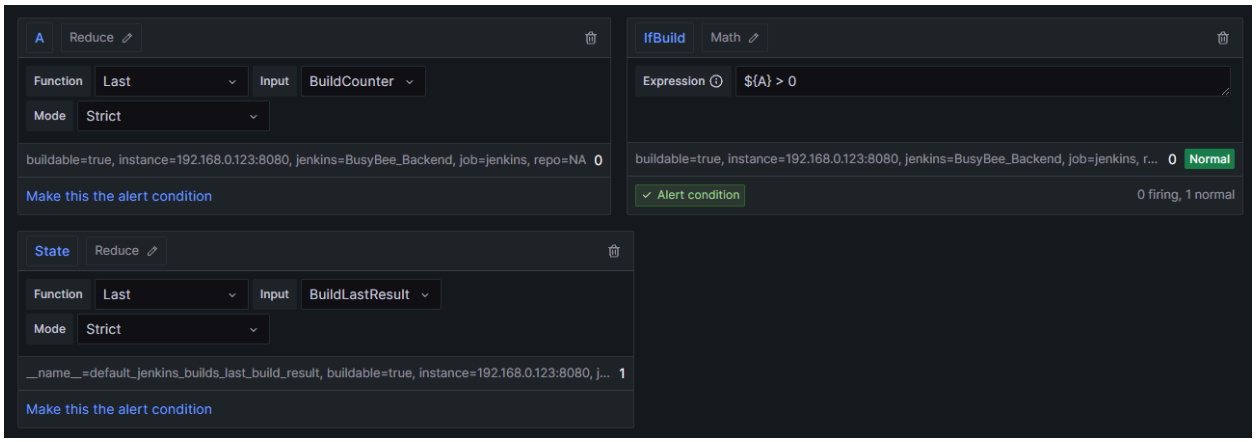


Суть така, що насамперед було створено запит "BuildCounter", завдання якого визначити чи змінювався лічильник кількості суми всіх збірок даного проекту. Була взята метрика "default_jenkins_builds_available_builds_count", відсортована на виведення інформації тільки для "BusyBee_Backend" і додана операція "changes", яка визначає чи відбулася зміна в лічильнику збірок чи ні, оскільки нам потрібно визначити не суму всіх збірок, а чи відбулося їх збільшення. У результаті виводиться "1", якщо відбулося складання, "0" якщо ні.

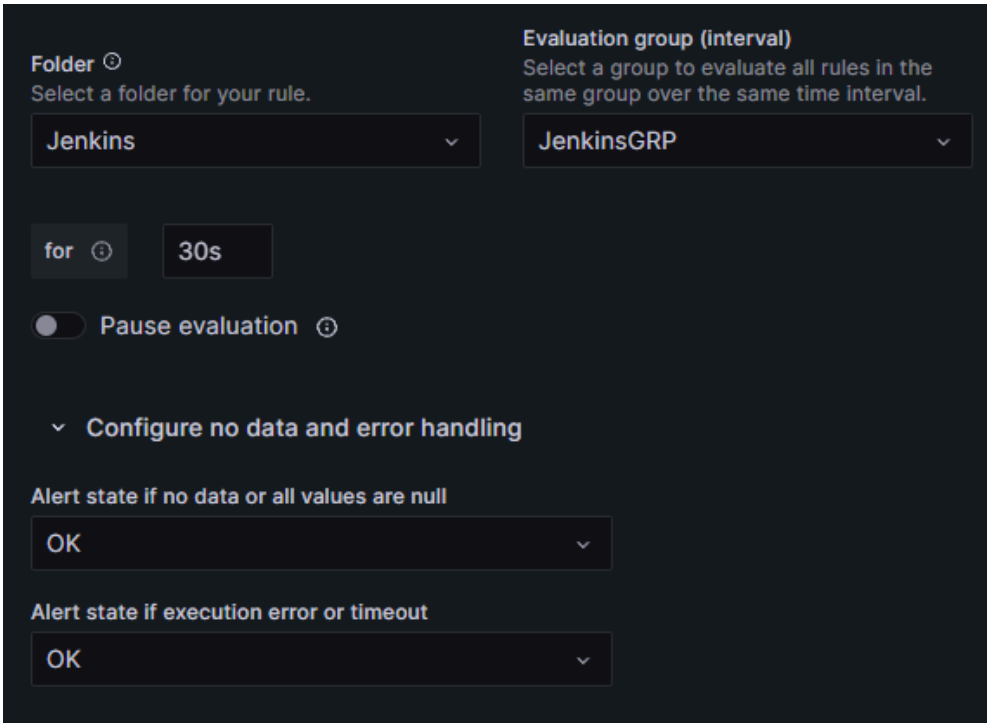
Створюємо ще один запит "BuildLastResult", де потрібно визначити успішно чи невдало завершилося наше складання. Дістаємо результат із метрики "default_jenkins_builds_last_build_result", сортуємо за назвою нашого проекту і бачимо значення: "1" якщо успішно, 0 у всіх інших випадках:



Було створено два вирази "Reduce" з функцією "Last", завдання яких діставати останнє значення з потоку даних наших запитів.
А так само створено тригер умову, яка запускатиме алерт коли наш запит "BuildCounter" на перевірку збільшення кількості збірок дорівнюватиме "1":

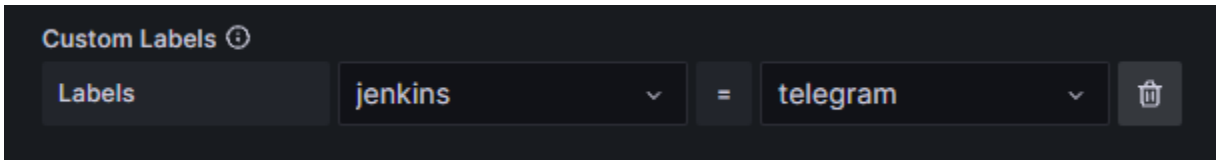


Далі було створено інтервал групи оцінювання в 10 секунд, що дозволить задавати нашим алертам час, який їм потрібен на спрацювання в 10 секунд, нам підходить значення у 30 секунд. Задаємо групу, а так само відключаємо відправлення повідомлень, коли значення наших тригерів дорівнюють "null" або при помилках, подібні фальшиві алерти з'являються, наприклад, при запуску сервера, що корисної інформації в даній ситуації не має:



У налаштуваннях опису задано коротку умову, яка набирає значення статусу збірки "BuildLastResult" запиту, і підміняє "1" на "SUCCESS", "0" на "FAIL" та додаємо посилання на панель статусів із дашборду:

Підв'язуємо політику повідомлень "jenkins = telegram":



Приклад повідомлення алерту:

BusyBeeMonitoring
Firing

Value: A=1, IfBuild=1, State=1
Labels:
 - alertname = BusyBee.API
 - jenkins = telegram
Annotations:
 - summary = Build status = SUCCESS
Dashboard:
https://grafana.busybee.space:443/d/haryan-jenkins?orgId=1
Panel:
https://grafana.busybee.space:443/d/haryan-jenkins?orgId=1&viewPanel=4

ВИСНОВКИ

У ході виконання даної дипломної роботи було проведено значну кількість роботи з впровадження принципів DevOps в веб-сайт проекту busybee.space. Цей процес вимагав спільної роботи розробників та оперативної команди з метою покращити якість розгортання та обслуговування веб-додатка.

Одним з ключових завдань впровадження DevOps було автоматизоване розгортання та управління додатком. Було розроблено та налаштовано засіб неперервної інтеграції та доставки (CI/CD), який дозволяє автоматично збирати, тестувати та розгортати додаток на серверах. Це значно скоротило час та ресурси, витрачені на розгортання нових версій додатка, а також знизило ймовірність помилок та проблем, пов'язаних з ними.

Протягом роботи було також впроваджено систему моніторингу та логування, що дозволяє оперативно виявляти проблеми та здійснювати аналіз даних для подальшого вдосконалення додатка. Це забезпечує ефективне виявлення і усунення помилок, а також забезпечує стабільну роботу сайту.

Однак, не зважаючи на проведену вже велику роботу, перед нами стоїть ще багато завдань. Один з найважливіших кроків - впровадження Kubernetes і IaC (Infrastructure as Code). Kubernetes дозволить нам автоматизувати управління контейнерами та забезпечити масштабованість додатка, а IaC дозволить нам визначати і конфігурувати інфраструктуру веб-серверів у вигляді коду, що спростить процес розгортання та забезпечить однорідність середовищ.

Крім того, для підвищення надійності та стійкості до відмов необхідно впровадити механізми резервного копіювання серверів та балансувальників навантаження. Це дозволить забезпечити швидке відновлення роботи в разі виникнення неполадок або відмов у системі.

У підсумку, впровадження принципів DevOps в проект busybee.space є складним і тривалим процесом, який вимагає спільних зусиль команди розробників, тестувальників та оперативної команди. Проделана робота

доводить, що впровадження DevOps може покращити якість та ефективність розгортання веб-додатків. З метою подальшого вдосконалення процесу необхідно продовжувати працювати над впровадженням Kubernetes, IaC та забезпечення високої доступності та надійності системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

№ п.п.	**Ресурс**
1.	Docker, Docker Compose, DockerHub Розробник: Docker, Inc. Документація
2.	Nginx Розробник: F5, Igor Sysoev, Netcraft Документація
3.	Jenkins Розробник: Kohsuke Kawaguchi Документація
4.	Publish Over SSH Розробник: Gavin McDonald, Olivier Lamy Документація
5.	Prometheus Розробник: SoundCloud, Cloud Native Computing Foundation Документація
6.	Grafana Розробник: Raintank Inc. Документація
7.	ChatGPT Розробник: OpenAI Документація
8.	Azure DevOps Розробник: Microsoft Corporation Документація
9.	Let’s Encrypt Розробник: ISRG Документація

10.	Ukraine.com.ua Розробник: ТОВ "Хостинг «Україна»" Документація
11.	Telegram Bot API Розробник: Telegram Messenger Inc. Документація