

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Радіoeлектроніки і телекомунікацій

(повне найменування інституту, факультету)

Мікро- та наноелектроніки

(повне найменування кафедри)

Пояснювальна записка

до дипломної роботи

бакалавр

(ступінь вищої освіти)

на тему: Автоматизація розгортання контейнерного додатку в робочому
середовищі на базі Amazon Web Services

Виконав: студент IV курсу, групи РТ-318 сп
Спеціальності 153 – “Мікро- та наносистемна
техніка”

(код і найменування спеціальності)

Освітня програма (спеціалізація)

“Мікро- та наноелектронні прилади та
пристрої”

Пехотін Д. А.

(прізвище та ініціали)

Керівник Сніжної Г. В.

(прізвище та ініціали)

Рецензент Манукян О. А.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет Радіoeлектроніки і телекомунікацій

Кафедра Мікро- та нанoeлектроніки

Ступінь вищої освіти бакалавр

Спеціальність 153 – “Мікро- та наносистемна техніка”

(код і найменування)

Освітня програма (спеціалізація) “Мікро- та нанoeлектронні прилади та пристрої”

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри

мікро- та нанoeлектроніки

д.т.н., доц. Сніжної Г.В.

« _____ » _____ 2021 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Пехотіну Дмитру Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Автоматизація розгортання контейнерного додатку в робочому середовищі на базі Amazon Web Services.

керівник проєкту (роботи) Сніжної Генадій Валентинович, д.т.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від « 17 » травня 2021 року № _____

2. Строк подання студентом проєкту (роботи) 25 травня 2021 р.

3. Вихідні дані до проєкту (роботи) Служба управління контейнерами Amazon ESC, репозиторій GitHub, контейнеризатор додатків Docker, система контролю версій GIT, інструмент Google Lighthouse, мережеве підключення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Описано сучасні тенденції в керуванні обчислювальними ресурсами. Використано служби AWS для автоматизації розгортання контейнерного додатку в робоче середовище. Проведено аналіз швидкості роботи сайту після переходу на новий хостинг.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація виконаної роботи

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	Сніжної Г. В., завідувач. каф. МіНЕ		
Нормо-контроль	Коротун А. В., к.ф.-м.н., доцент каф. МіНЕ		

7. Дата видачі завдання «26» квітня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Огляд літературних джерел та ознайомлення зі статтями за заданою тематикою.	26.04 – 05.05	Виконано
2	Встановлення та налаштування програм Docker і GIT.	06.05 – 12.05	Виконано
3	Створення репозиторію GitHub.	13.05 – 14.05	Виконано
4	Розгортання контейнерного додатка Docker в робоче середовище Amazon ECS.	15.05 – 20.05	Виконано
5	Аналіз швидкості роботи сайту.	21.05 – 22.05	Виконано
6	Оформлення пояснювальної записки і презентації.	23.05 – 25.05	Виконано

А.

Студент

_____ Пехотін Д.

(підпис)

(прізвище та ініціали)

Керівник проекту (роботи) _____ Сніжної Г. В.

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

ПЗ: 56 стор., 43 рис., 17 джерел.

Об'єкт – Контейнерний додаток (сайт) розгорнутий в робоче середовище на базі AWS.

Мета – ознайомлення зі складовими роботи додатків на хмарному хостингу AWS, впровадження контейнерного додатку за допомогою служби Amazon ECS.

Методи – описовий, аналітичний, системне адміністрування.

Результати: додаток розгорнуто в новому середовищі, виконано оновлення вихідного коду в працюючій контейнерній службі без її зупинки з використанням AWS CodePipeline і тестів AWS CodeBuild, збільшення ефективності роботи сайту.

АВТОМАТИЗАЦІЯ, ДОДАТОК, AMAZON ECS, ХМАРНИЙ
ХОСТИНГ, РОБОЧЕ СЕРЕДОВИЩЕ, СИСТЕМНЕ АДМІНІСТРУВАННЯ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
1 СУЧАСНІ ТЕНДЕНЦІЇ В КЕРУВАННІ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ	.8
1.1 Використання хмарний технологій для хостинга корпоративних сайтів	.8
1.2 Сучасні технології доставки коду і управління інфраструктурою . .	12
2 АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ КОНТЕЙНЕРНОГО ДОДАТКУ В РОБОЧЕ СЕРЕДОВИЩЕ НА БАЗІ AWS	.14
2.1 Контейнеризатор додатків Docker і система контролю версій GIT.	.14
2.2 Особливості налаштування служби управління контейнерами Amazon ECS	17
2.3 Створення репозиторію GitHub	22
2.4 Розгортання контейнерного додатка Docker в робоче середовище Amazon ECS за допомогою AWS CodePipeline і тестів AWS CodeBuild	.25
3 АНАЛІЗ ШВИДКОСТІ РОБОТИ САЙТУ	.46

3.1 Порівняння швидкості роботи сайту після переходу на AWS	46
3.2 Економічні аспекти переходу на нову платформу	48
3.3 Заходи щодо оптимізації роботи сайту	50
ВИСНОВКИ	.54
ПЕРЕЛІК ПОСИЛАНЬ	55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОП - оперативна пам'ять;

ЦП - центральний процесор;

DevOps - Development-Operation;

CI - Continuous integration;

IaC - Infrastructure-as-Code;

AWS - Amazon Web Services;

ECS - Elastic Container Service;

CD - Continuous delivery або Continuous deployment;

LCP - Largest Contentful Paint;

CLS - Cumulative Layout Shift.

ВСТУП

Хмарний хостинг це нове слово в індустрії високих технологій. Це рішення, яке працює через мережу і дозволяє компаніям використовувати об'єднані обчислювальні ресурси як утиліту.

Сучасні темпи розвитку інтернет технологій змусили звернути увагу на проблему в роботі корпоративного сайту меблевої фабрики. Це недостатня швидкість оброки PHP коду ресурсами хостинг провайдеру і обмежені можливості з адміністрування веб-сервера. Ці проблеми суттєво зупиняють розвиток веб-додатка, обмежуючи його швидкість оновлення і ефективність роботи.

Для рішення цієї проблеми можливо залучити сервіс Amazon Elastic Container Service. Це високомасштабована і швидка служба управління контейнерами, яка спрощує запуск, зупинку і управління контейнерами в кластері. За допомогою програми Docker і системи контролю версій GIT було налаштовано роботу конвеєра CI/CD для контейнерного додатку, що є основою для перенесення і роботи на базі середовища Amazon ECS.

Для цього потрібно зробити оновлення вихідного коду для образу Docker і розгортання нового образу Docker в працюючій контейнерній службі без її зупинки з використанням AWS CodePipeline і тестів AWS CodeBuild.

Перевірка ефективності і швидкості завантаження веб-сторінок, можливо виконати за допомогою інструменту Google Lighthouse.

Додатково будуть запропоновані покращення для роботи сайту та приведено економічні аспекти переходу на нову платформу.

1. СУЧАСНІ ТЕНДЕНЦІЇ В КЕРУВАННІ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ

1.1 Використання хмарний технологій для хостинга корпоративних сайтів

У світі веб-хостингу існує безліч варіантів, які дозволяють розмістити веб-сайт в Інтернеті. Хоча всі вони діють як місце зберігання для веб-сайту, вони розрізняються обсягом сховища, контролем, вимогами до технічних знань, швидкістю і надійністю сервера. Загальний хостинг ідеально підходить для хостингу веб-сайтів початкового рівня. Веб-сайт буде зберігатися на тому ж сервері, що і кілька інших веб-сайтів. В схемі роботи загального хостингу всі домени використовують одні і ті ж ресурси сервера, такі як ОП (оперативна пам'ять) і ЦП (центральний процесор). Оскільки всі ресурси є спільними, вартість планів загального хостингу відносно невисока і підходить для власників веб-сайтів, які не отримують великого обсягу веб-трафіку. Але якщо обсяг користувачів значно збільшується, сплеск використання сайту може в кінцевому підсумку вплинути на його доступність і стабільну роботу.

В дипломному проекті описано як перехід на хмарний хостинг вирішив проблему доступності корпоративного сайту меблевої фабрики.

Хмарний хостинг - це нове слово в індустрії високих технологій. Що стосується веб-хостингу, це означає, що безліч комп'ютерів працюють разом, виконуючи задачі з використанням об'єднаних обчислювальних ресурсів. Це рішення, яке працює через мережу і дозволяє компаніям використовувати обчислювальні ресурси як утиліту.

Це дозволяє користувачам використовувати стільки ресурсів, скільки їм потрібно, без необхідності створювати і підтримувати власну обчислювальну інфраструктуру. Використовувані ресурси розподілені на декількох серверах, що знижує ймовірність простою через збій сервера.

Хмарний хостинг є масштабованим, тобто сайт може з часом збільшуватися, а власник веб-сайту буде платити тільки за ті ресурси, які були використані.

В дипломному проекті буде розглянуто хмарний хостинг на базі Amazon Web Services та етапи його налаштування.

Головна проблема з якою зіткнулася компанія це недостатня швидкість оброки PHP коду ресурсами хостинг провайдеру і в підсумку довге завантаження веб-сторінок сайту. Хмарний хостинг ідеально підходить для вирішення цієї проблеми, але використання AWS вимагає змінити принципи розробки, що може бути великою складністю для компаній з застарілою моделлю бізнесу.

Зміна моделі розробки на DevOps надала компанії такі переваги як: швидкість розробки і інтеграція оновлень, зменшена кількість помилок і їх постійних виправлень, безперебійна доступність веб-ресурсу та можливість його подальшого розширення.

DevOps (від Development-Operation, розробка-експлуатація) - це поєднання культурних принципів, підходів і засобів, які покращують здатність компаній створювати додатки і сервіси на високій швидкості. З DevOps розробка і оптимізація продуктів відбувається швидше, ніж при використанні традиційних процесів роботи над програмним забезпеченням і управління інфраструктурою.

У моделі DevOps кордони між групами розробки та експлуатації стираються. Іноді ці дві групи об'єднуються в одну загальну, де інженери працюють над усім життєвим циклом додатка - від розробки і тестування до розгортання і експлуатації - і розвивають цілий ряд навичок, не обмежуючись вузькою спеціалізацією. Ці групи використовують спеціальні методи для автоматизації процесів, які традиційно виконувалися повільно і вручну.

Переваги моделі розробки DevOps:

- Швидкість

Можливість працювати з високою швидкістю, щоб швидше впроваджувати нові можливості для клієнтів, краще адаптуватися до швидко змінним ринкам.

- Швидка доставка

Збільшується частота і швидкість релізів, щоб швидше оновлювати і покращувати свій продукт. Безперервна інтеграція і безперервна доставка допомагають автоматизувати процес випуску програмного забезпечення - від збірки до розгортання.

- Надійність

Контролюйте якість оновлень додатків і змін в інфраструктурі, щоб надійно і швидко розробляти продукти, а також зберігати лояльність кінцевих користувачів.

- Масштабування

Можливість керувати процесами розробки та підтримки інфраструктури, а також забезпечувати їх стабільну роботу при будь-якому масштабі. Інфраструктура як код сприяє більш ефективному управлінню середовищами розробки.

- Безпека

Модель DevOps можна впровадити без шкоди для безпеки за допомогою автоматизованої політики дотримання вимог, точного налаштування, а також методик управління конфігураціями. Наприклад, використовуючи інфраструктуру як код і політику як код, можна визначити вимоги, а потім відстежувати їх дотримання при будь-якому масштабі.

Впровадження нової моделі значно збільшує число розгортань, в зв'язку з чим можуть виникати проблеми на рівні експлуатації. Тому необхідні такі методи DevOps, як безперервна інтеграція і безперервна доставка, які вирішують ці проблеми і дозволяють компаніям доставляти оновлення швидко, безпечно і надійно. Методики автоматизації інфраструктури - інфраструктура як код і управління конфігураціями

допомагають підтримувати гнучкість обчислювальних ресурсів і адаптуватися до частих змін.

AWS надає сервіси, які призначені і створені для DevOps, і роботою з хмарою AWS. На базі цих сервісів була реалізована модель DevOps.

1.2 Сучасні технології доставки коду і управління інфраструктурою

Перш ніж більш детально розглядати концепції DevOps, слід зазначити, що значить "безперервний" в області програмного забезпечення. Простіше кажучи, термін "безперервності" відноситься до змін програмного забезпечення, які відбуваються протягом всього процесу розробки ПЗ.

Звичайно ж є частка неточності в терміні "безперервний". Насправді після реалізації функціоналу може пройти деякий час до того, як код потрапить в робоче середовище, але цей час, все-таки, набагато менше, ніж було до появи DevOps.

Continuous delivery (безперервна доставка).

У більшості випадків безперервна доставка - це серія практик, спрямованих на те, щоб оновлення програмного забезпечення відбувалися практично постійно. Розробник, написавши новий код, відправляє його QA-інженерам для тестування. Тестувальникам легше буде протестувати невеликий новий функціонал і написати до нього тест-кейси. Як тільки всі перевірки пройдені, новий код потрапляє на подальше тестування авто-тестами і потім вже потрапляє в релізну гілку системи контролю версій.

Continuous integration (безперервна інтеграція).

Основою даної практики є постійне потрапляння коду в центральний репозиторій після успішного запуску тестів. Основні цілі CI - пошук і усунення потенційних проблем якомога швидше, поліпшення якості ПЗ і скорочення часу для випуску оновлень. До того, як безперервна інтеграція стала широко поширеною, розробники зазвичай працювали ізольовано, а

тільки по закінченню роботи об'єднували свої напрацювання. При безперервній інтеграції розробники часто завантажують свої зміни в центральний репозиторій, виконуючи до цього unit-тести. Потім система контролю версій автоматично перевіряє код на можливість безпечної інтеграції з існуючим в репозиторії. При цьому йде постійне надходження коду, що полегшує тестування і зводить до мінімуму можливі ризики.

Continuous deployment (безперервне розгортання).

Безперервна доставка забезпечує постійний випуск оновлень користувачам. А безперервне розгортання відповідає за те, щоб весь новий функціонал після тестування відразу ж потрапив в основну програму без ручного втручання інженерів DevOps.

Наприклад, програма Docker створена для безперервного розгортання. DevOps інженери можуть оновлювати контейнери і розгортати їх відразу в робоче середовище в автоматичному режимі. Такий процес є ключем до безперервної доставки, тому що весь процес може зайняти всього лише кілька хвилин.

Короткий вигляд процесу доставки коду:

- Код відправляється в центральний репозиторій;
- На сервері безперервної інтеграції зміни поєднуються з основним кодом, виконуються unit-тести і все завантажується на staging (тестове) середовище;
- В staging середовищі тестується додаток, все перевіряється і готується до потрапляння в робоче середовище;
- Розгортання в робочому середовищі.

Інфраструктура як код.

У минулому управління IT-інфраструктурою було складним завданням. Системним адміністраторам доводилося вручну керувати та налаштовувати все обладнання і програмне забезпечення, необхідне для роботи додатків. Не дивно, що цей ручний процес часто приводив до кількох проблем, а саме:

- Масштабованість і доступність.

Оскільки ручне налаштування виконується дуже повільно, сайтам часто доводиться стикатися з різкими стрибками доступності, в той час як системні адміністратори відчайдушно намагаються налаштувати сервери для управління навантаженням. Це обов'язково впливає на доступність сайту.

- Непослідовність.

Якщо є кілька інженерів, які вручну розгортають конфігурації, розбіжності будуть неминучими.

Однак за останні роки все кардинально змінилося. Такі тенденції, як хмарні обчислення, зробили революцію і поліпшили спосіб проектування, розробки та обслуговування своєї ІТ-інфраструктури організаціями.

Інфраструктура як код - це спосіб постачання та керування обчислювальними та мережевими ресурсами методом їх опису у вигляді програмного коду, на відміну від налаштовування необхідного обладнання власноруч чи з допомогою інтерактивних інструментів.

Якщо переписати більш просту версію цього терміну:

Інфраструктура як код (IaC) означає управління ІТ-інфраструктурою за допомогою файлів конфігурації.

З IaC конфігурація інфраструктури приймає форму файлу коду. Оскільки це всього лише текст, його легко редагувати, копіювати і поширювати.

Тепер потрібно перерахувати основні переваги, які організація отримала, прийнявши рішення IaC:

- Швидкість.

Інфраструктура як код дозволяє швидко налаштувати всю інфраструктуру, запустивши сценарій. Це можливо зробити для будь-якого середовища, від розробки до робочого середовища, проходження стадії тестування, контролю якості тощо.

- Підвищення ефективності протягом усього циклу розробки.

Ручні процеси призводять до помилок. IaC вирішує цю проблему, оскільки самі файли конфігурації є єдиним джерелом. Таким чином

гарантується, що одні й ті ж конфігурації будуть розгортатися знову і знову без розбіжностей.

2. АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ КОНТЕЙНЕРНОГО ДОДАТКУ В РОБОЧЕ СЕРЕДОВИЩЕ НА БАЗІ AWS

2.1 Контейнеризатор додатків Docker і система контролю версій GIT

Для того щоб додаток працював на базі AWS ECS його потрібно помістити в контейнерне середовище (контейнеризувати), для цього буде використовуватися програма Docker.

Docker - це програмне забезпечення з відкритим кодом, принцип роботи якого найпростіше порівняти з транспортними контейнерами, адже колись транспортні компанії стикалися зі схожими проблемами:

- Перевозка різних (несумісні) типи товарів разом.
- Обробка пакетів різних розмірів одним і тим же транспортним засобом.

З введенням контейнерів стало можливим перевозити багато різних товарів разом. Вантаж різного розміру може бути розподілений по стандартизованим контейнерам, які завантажуються і вивантажуються одним і тим же транспортним засобом.

Для розробки додатків, потрібно надати код разом з усіма його складовими, такими як бібліотеки, сервер, бази даних і т. п. Якщо цього не робити, можливо опинитися в ситуації, коли додаток працює на вашому комп'ютері, але відмовляється включатися на пристрою іншого користувача.

Ця проблема вирішується через створення незалежності ПЗ від системи.

Docker же просто розділяє ядро ОС на різні контейнери (Docker container), що працюють як окремі процеси.

- Контейнер - це процес, який інкапсулює необхідне програмне забезпечення. Контейнер складається з образів. Його можна легко видалити і знову створити за короткий проміжок часу.
- Образ - базовий елемент кожного контейнера.

Платформи на яких працює Docker.

Docker працює не тільки на його рідній ОС, Linux, але також підтримується Windows і macOS. Єдина відмінність від взаємодії з Linux в тому, що на macOS і Windows платформа інкапсулюється в крихітну віртуальну машину. На даний момент Docker для macOS і Windows досяг значного рівня зручності у використанні.

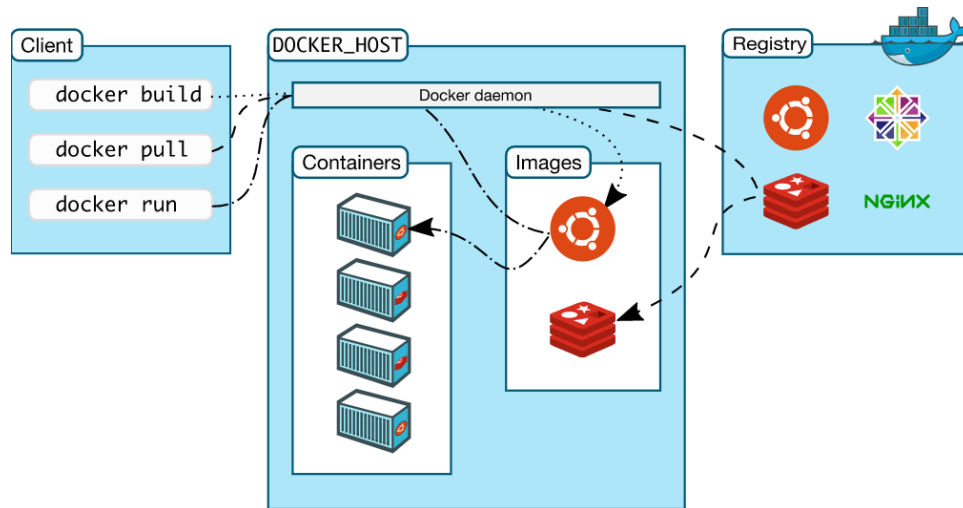


Рисунок 2.1 – Базова архітектура роботи Docker

Архітектура Docker. Docker використовує архітектуру клієнт-сервер. Клієнт Docker спілкується з Docker daemon, який виконує роботу по створенню, запуску і розширенню контейнерів Docker. Клієнт (Client) і Docker daemon можуть працювати в одній системі, або є можливість підключити клієнт Docker до віддаленого Docker daemon. Клієнт і daemon обмінюються даними за допомогою REST API, через сокети UNIX або мережевий інтерфейс. Інший клієнт Docker - це Docker Compose, який дозволяє працювати з додатками, що складаються з набору контейнерів.

Відзначимо найбільш важливі складові роботи Docker:

- Щоб створити образ, потрібно створити Dockerfile - це текстовий файл з інструкціями та аргументами.
- Docker Hub - публічний репозиторій з інтерфейсом, що надається Docker Inc. Він зберігає безліч образів. Ресурс є джерелом «офіційних»

образів, зроблених командою Docker або створених у співпраці з розробником програмного забезпечення.

- Том - описується як загальна папка. Тома ініціалізуються при створенні контейнера і призначені для збереження даних, незалежно від життєвого циклу контейнера.
- Реєстр (registry) - це сервер, на якому зберігаються образи.

Система GIT - це розподілена система контролю версій програмного коду, яка використовується, щоб відстежувати зміни, які відбуваються з часом.

Тобто крок за кроком ми бачимо, які файли змінилися і як. Особливо це важливо, коли аналізуєш, що було зроблено в рамках одного завдання: це дає можливість повернутися на крок назад і не втрачати багато часу на безглузду роботу.

Програма сумісна як з Linux, так і з Windows. Для початку роботи з GIT потрібно тільки завантажити exe файл з офіційного сайту, запустити його і встановити на комп'ютер.

З GIT можна працювати не тільки з локальним репозиторієм, а і з віддаленим. Вся інформація знаходиться на віддаленому сервері, локально зберігається тільки копія проекту, зміни якої можна відправити в віддалений репозиторій.

Віддалений репозиторій - це якесь місце, де зберігається вихідний код і звідки можна його клонувати. GitHub - це найбільше сховище для репозиторіїв і спільної розробки.

Основні поняття в роботі з Git:

- Репозиторій Git (git repository) - каталог файлової системи, в якому знаходяться: файли конфігурації, файли журналів операцій виконуваних над репозиторієм, індекс розташування файлів і репозиторій, що містить самі контрольовані файли.
- Комміт (Commit) - фіксація змін або запис змін в репозиторій. Це основний об'єкт в управлінні контролю версій. Він містить всі зміни за

час цього комміту. Комміти пов'язані між собою як однозв'язний список. А саме: Є перший комміт. Коли створюється другий комміт, то він (другий) знає, що йде після першого. І таким чином можна відстежити інформацію.

- Гілка (Branch) - це паралельна версія репозиторію. Вона включена в цей репозиторій, але не впливає на головну версію, тим самим дозволяючи вільно працювати в паралельній. Також гілка - це вказівник деякого комміту. Комміт фіксує, який комміт був до нього, коли гілка вказує на якийсь комміт, до неї відносяться і всі попередні. Робота відбувається в гілках, тому коли створюється новий комміт, гілка переносить свій вказівник на більш новий.
- Мєрдж (Merge) - злиття змін з будь-якої гілки сховища з будь якої гілкою цього ж сховища.
- Пуш (Push) - відправка всіх невідправлених коммітів на віддалений сервер репозиторію.
- Ігнорування (.gitignore) – ігнорування будь яких типів файлів.

2.2 Особливості налаштування служби управління контейнерами Amazon ECS

Amazon Elastic Container Service (Amazon ECS) - це високомасштабована і швидка служба управління контейнерами, яка спрощує запуск, зупинку і управління контейнерами в кластері. Контейнери визначені в конфігураційних налаштуваннях (task definition), які використовуються для виконання окремих задач або задач в рамках служби. У цьому контексті служба - це конфігурація, яка дозволяє запускати і підтримувати певну кількість задач одночасно в кластері. В службі Amazon ECS можна запускати свої задачі і сервіси в безсерверній інфраструктурі, керованій процесом AWS Fargate.

ECS дає можливість запланувати розміщення контейнерів в кластері в залежності від потреб до ресурсів, політик ізоляції і вимог до доступності. Завдяки цьому не потрібно використовувати власні системи управління кластером і конфігурацією або турбуватися про масштабування інфраструктури.

Зазвичай Amazon ECS використовують для таких задач:

- Створення узгодженого середовища розробки и робочого середовища;
- Створення складних додатків на основі моделі мікросервісів.

Amazon EC2 розміщений у кількох місцях по всьому світу. Ці місцеположення складаються з регіонів (Region). Amazon ECS – це регіональний сервіс, який забезпечує роботу контейнерів з високою доступністю в кількох зонах доступності в межах одного регіону. Зони доступності (Availability Zones) - це декілька ізольованих місцеположень в кожному регіоні.

Для правильної роботи кластерів треба створити файл конфігурації, який визначає, які образи контейнерів будуть працювати у кластерах. Завдяки конфігураціям можна запускати нові задачі або створювати служби. Для зберігання образів контейнерів та їх подальшого завантаження треба використовувати реєстри контейнерів, наприклад, рідний сервіс Amazon Elastic Container.

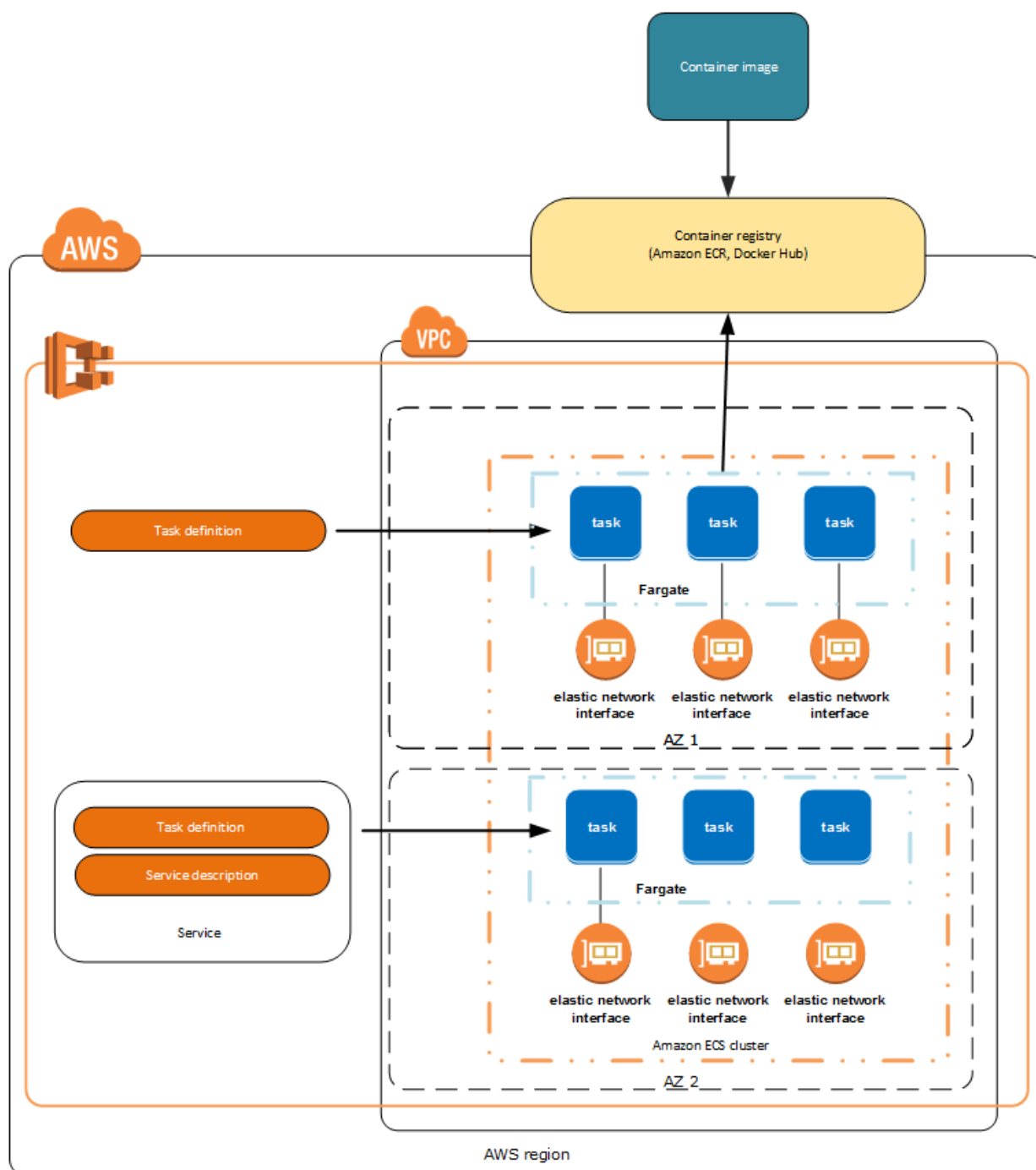


Рисунок 2.2 - Архітектура середовища Amazon ECS, що працює на AWS Fargate

Конфігураційний файл в Amazon ECS називається task definitions. Щоб підготувати додаток до запуску в Amazon ECS, необхідно створити файл конфігурації - це текстовий файл (в форматі JSON), який описує один або декілька контейнерів (до десяти), які формують додаток. Або використовувати веб-інтерфейс для швидкого створення конфігурації.

Визначення task definitions можна розглядати як основу для додатку, що визначає багато важливих параметрів для його роботи. Наприклад, ці параметри можуть використовуватися, щоб вказувати:

- які контейнери слід використовувати;
- які порти слід відкривати;
- які томи даних слід використовувати з контейнерами в задачах.

Конкретні параметри, які слід використовувати, залежать від потреб конкретного додатку і можуть бути значно ускладнені.

Також в середовищі Amazon ECS використовують поняття задачі і планувальник задач. Задача є конкретизацією task definition в кластері. Після створення файлу конфігурації, з'являється можливість вказати кількість задач, які будуть виконуватися в кластері, за це відповідає планувальник задач.

Планувальник забезпечує дотримання заданої стратегії планування та перепланує задач в разі їх збою (наприклад, при збою базової інфраструктури).

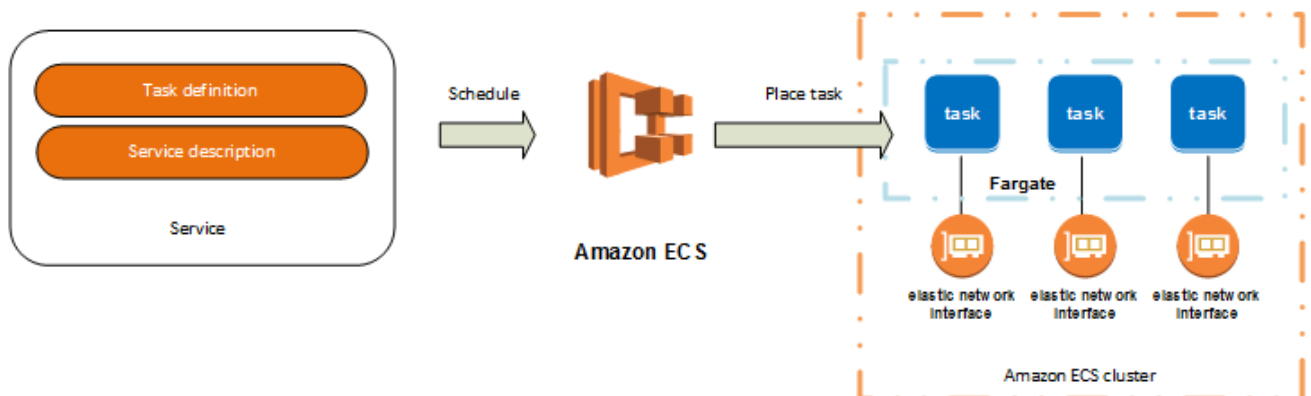


Рисунок 2.3 – Робота планувальника задач

Кластер Amazon ECS - це логічна група задач або послуг, на якій можливо зареєструвати один або кілька інстанцій Amazon EC2 (так звані container instances) в своєму кластері для виконання на них задач.

Або використовувати тільки безсерверну інфраструктуру, яку надає Fargate, для запуску задач всередині неї. Коли задачі виконуються на Fargate, ядро керує ресурсами кластера самостійно.

При першому використанні Amazon ECS спершу створюється кластер за замовчуванням. Якщо цього буде мало, можливо створити додаткові кластери в обліковому записі, щоб ресурси були більш розділені.

ECS agent – це агент контейнерів, який працює на кожному екземплярі контейнера в межах кластера Amazon ECS. Агент відправляє інформацію про поточні виконувані задачі і використані ресурсів в Amazon ECS. Він запускає і зупиняє задачі щоразу, коли отримує запит від Amazon ECS.

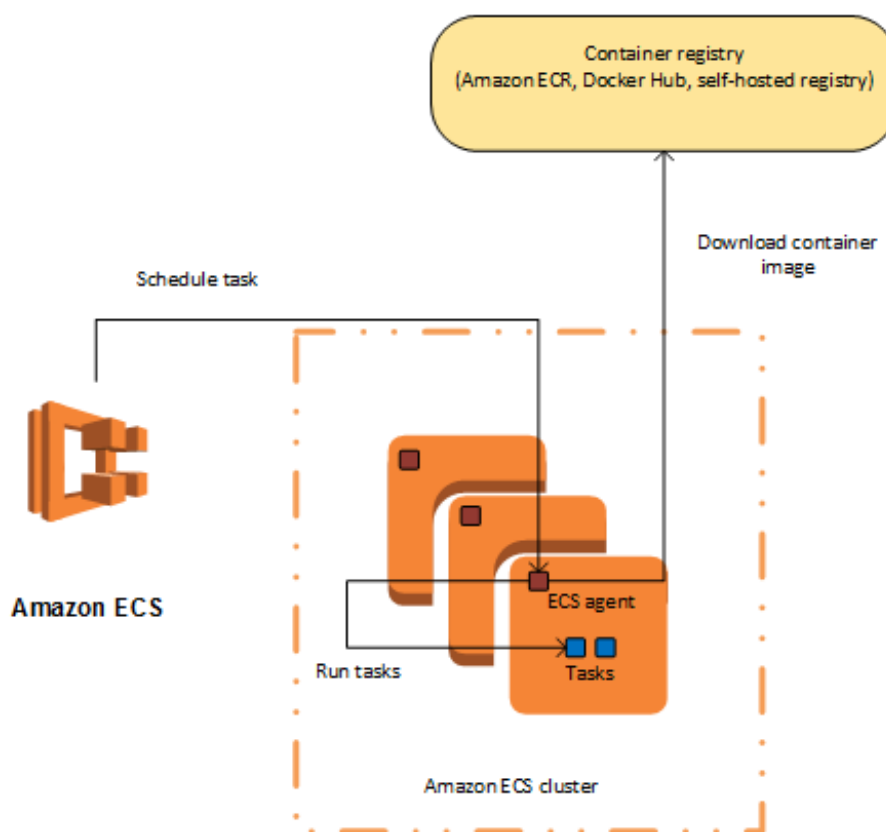


Рисунок 2.4 – Зв'язок роботи Amazon ECS і ECS agent

2.3 Створення репозиторію GitHub

Щоб приступити до створення компонентів, необхідних для побудови архітектури, знадобиться репозиторій GitHub.

Створення нового репозиторію:

1. Реєструємо особистий кабінет на сайті github.com.
2. У правому верхньому куті будь-якої сторінки використовуємо спливаюче меню і вибираємо «New repository».

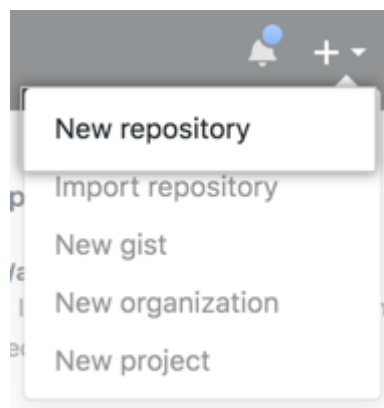


Рисунок 2.5 – Створення нового репозиторію

3. Щоб створити репозиторій зі структурою каталогів і файлами існуючого репозиторію (якщо він є), тиснемо кнопку «Choose a template» и вибираємо репозиторій з шаблонів, якщо немає - вибираємо «No template».

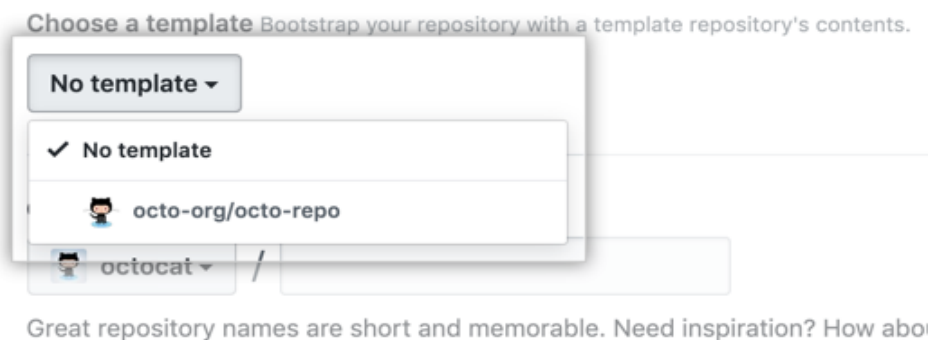


Рисунок 2.6 – Створення репозиторію з шаблону

4. Далі, у списку власників вибираємо обліковий запис, в який треба створити репозиторій.

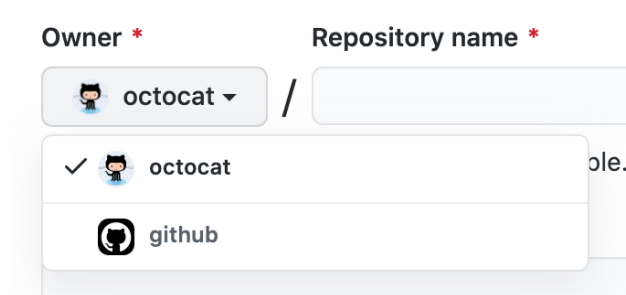


Рисунок 2.7 – Вибір облікового запису для створення репозиторію

5. Вводимо назву репозиторію і опис.

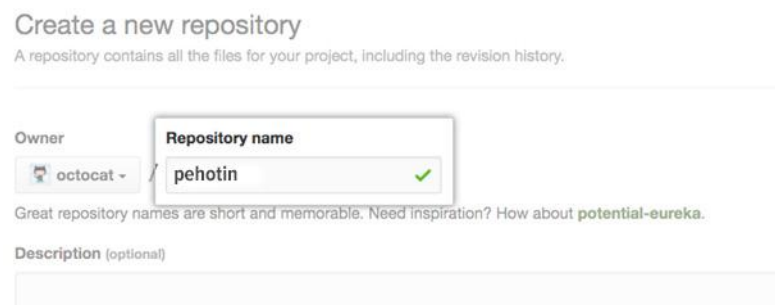


Рисунок 2.8 – Етап створення нового репозиторію

6. Вибираємо видимість репозиторію, він може бути публічним або приватним. Вибираємо приватний.

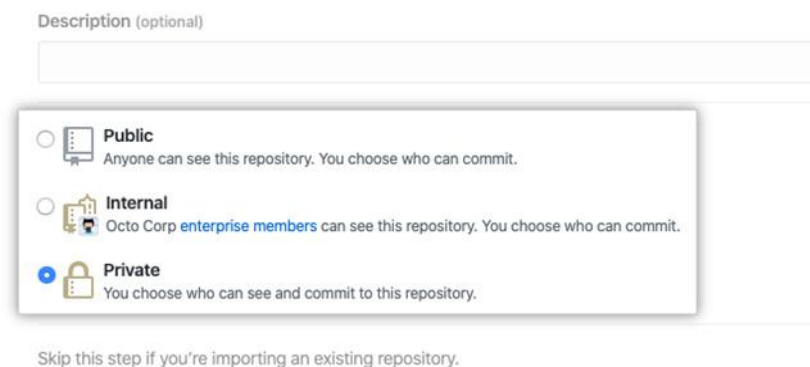


Рисунок 2.9 – Налаштування видимості репозиторію

7. Якщо не використовувати шаблон, є ряд додаткових елементів, які можливо заповнити. Додавати і змінювати файли можливо з графічного інтерфейсу або за допомогою командної строки.

Додаткові елементи які можна створити:

- Файл README, який являє собою документ, що описує проект.
- Файл .gitignore, який представляє собою набір правил ігнорування.
- Ліцензію на програмне забезпечення для свого проекту.

8. Натискаємо «Create repository».

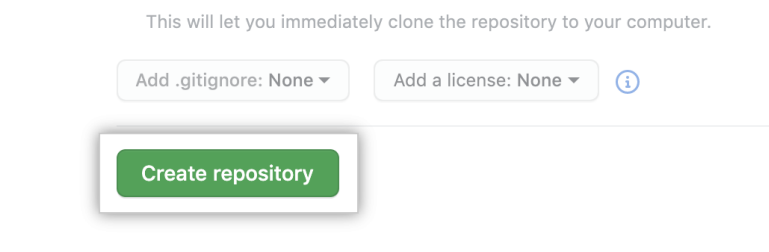


Рисунок 2.10 – Кнопка створення репозиторію

9. Знизу сторінки швидкого налаштування в розділі «Import code» можливо вибрати імпорт проекту в новий репозиторій.

10. Імпортуємо образ Docker з вихідним кодом сайту.

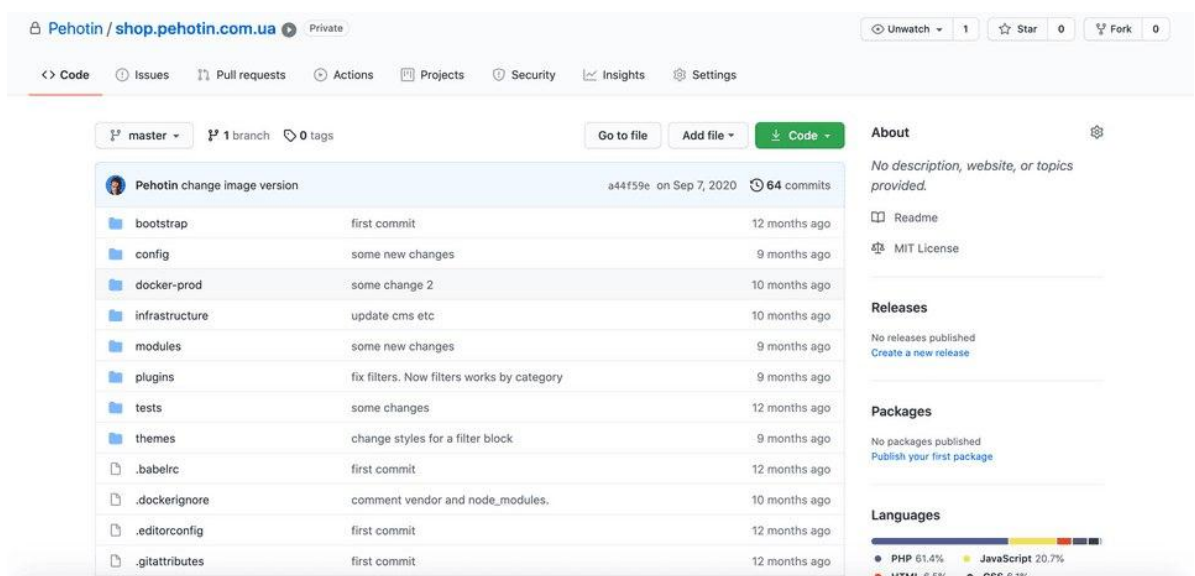


Рисунок 2.11 – Імпортований образ в приватному репозиторії GitHub

2.4 Розгортання контейнерного додатка Docker в робоче середовище Amazon ECS за допомогою AWS CodePipeline і тестів AWS CodeBuild

Створення ефективного та дієвого конвеєра CI/CD має вирішальне значення для контейнерних додатків, незалежно від платформи, яка використовується для управління контейнерами.

Було використано GitHub як репозиторії вихідного коду, AWS CodePipeline для створення повного конвеєра CI/CD для додатка, розгорнутого на Amazon ECS. Також буде показано, як AWS CodeBuild можна використовувати для виконання тестів додатків в рамках повного конвеєра CI/CD.

AWS CodeBuild підтримує webhooks, коли GitHub використовується в якості сховища вихідного коду. Це дозволяє CodeBuild працювати з вихідним кодом, що зберігається в GitHub, і використовувати webhooks для запуску процесу злиття (merge) вихідного коду кожного разу, коли новий код (комміт) відправляється (push) в репозиторій GitHub.

При побудові конвеєра CI/CD критично важливим для підтримки стандартів якості додатків є етап «тестування».

CodeBuild використовується не тільки для компіляції коду програми і створення образу контейнерного додатку, а також для запуску тестів для перевірки коду (такі як модульні тести, статичний аналіз коду і інтеграційні тести).

Вебхук (англ. webhook) в веб-розробці - метод розширення або зміни поведінки веб-сторінки або веб-додатки за допомогою зворотних викликів. Зазвичай вони запускаються якою-небудь подією, наприклад, відправкою коду в репозиторій.

AWS CodePipeline - це сервіс DevOps для безперервної інтеграції, безперервної доставки і безперервного розгортання додатків, розміщених на

різних платформах AWS. Перевага використання AWS CodePipeline для сервісу AWS ECS полягає в тому, що сервіс ECS продовжує працювати, поки створюється і розгортається новий образ Docker.

CodePipeline складається з трьох етапів: інтеграція вихідного коду, збірка вихідного коду і розгортання, як показано на рисунку.



Рисунок 2.12 - Етапи CodePipeline

Для вихідного коду буде використовуватися репозиторій GitHub. Для збірки вихідного коду проект AWS CodeBuild. Для розгортання сервіс ECS типу запуску Fargate.

Для налаштування середовища головна умова - це обліковий запис AWS, який попередньо створений. Буде використовуватися образ Docker rehotin/nginx з вихідним кодом сайту, який було завантажено на GitHub сервер.

Етап створення репозиторію описано в розділі «Створення репозиторію GitHub», тому в цьому розділі буде описано, які файли потрібні для роботи додатку. Репозиторій повинен включати dockerfile, з якого можна зробити образ Docker. Dockerfile для образу rehotin/nginx заснований на образі Docker nginx:alpine. Інструкції Dockerfile копіюють файли з папки /var/www/html, які використовуються для створення додатку, в поточний каталог відкривається порт 80 для прослуховування сервером Nginx.

Вихідний код Dockerfile:

```
FROM nginx:alpine
```

```
LABEL maintainer="Mahmoud Zalt <mahmoud@zalt.me>"
```

```

COPY ./docker-prod/nginx/nginx.conf /etc/nginx/
ARG CHANGE_SOURCE=false
RUN if [ ${CHANGE_SOURCE} = true ]; then \
    sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/' /etc/apk/repositories \
;fi
RUN apk update \
    && apk upgrade \
    && apk --update add logrotate \
    && apk add --no-cache openssl \
    && apk add --no-cache bash
RUN apk add --no-cache curl
RUN set -x ; \
    addgroup -g 82 -S www-data ; \
    adduser -u 82 -D -S -G www-data www-data && exit 0 ; exit 1
ARG PHP_UPSTREAM_CONTAINER=php-fpm
ARG PHP_UPSTREAM_PORT=9000
RUN touch /var/log/messages
COPY ./docker-prod/nginx/logrotate/nginx /etc/logrotate.d/
COPY ./docker-prod/nginx/sites /etc/nginx/sites-available
COPY ./docker-prod/nginx/ssl /etc/nginx/ssl
RUN      echo      "upstream      php-upstream      {      server
${PHP_UPSTREAM_CONTAINER}:${PHP_UPSTREAM_PORT};      }"      >
/etc/nginx/conf.d/upstream.conf \
    && rm /etc/nginx/conf.d/default.conf
WORKDIR /var/www/html
RUN chown -R www-data:www-data /var/www/html
ADD ./docker-prod/nginx/startup.sh /opt/startup.sh
RUN sed -i 's/\r//g' /opt/startup.sh
CMD ["/bin/bash", "/opt/startup.sh"]
EXPOSE 80 81 443

```

Інші файли в папці `\docker-prod\nginx`, які потрібні для правильної роботи додатку:

Имя	Размер	Сжат	Тип	Изменён	CRC32
..			Папка с файлами		
ssl	23	22	Папка с файлами	07.09.2020 18:53	
sites	8 220	3 045	Папка с файлами	07.09.2020 18:53	
logrotate	324	173	Папка с файлами	07.09.2020 18:53	
startup.sh	458	209	Файл ".SH"	07.09.2020 18:53	FE2B4017
nginx.conf	1 369	551	Файл ".CONF"	07.09.2020 18:53	8B674A72
Dockerfile	1 494	720	Файл	07.09.2020 18:53	54B0DE1B

Рисунок 2.13 – Інші конфігураційні файли для роботи модулів сайту

Додавання Build Spec для проекту CodeBuild. Build Spec - це файл синтаксису YAML з командами і налаштуваннями, які використовуються проектом CodeBuild для запуску збірки. Файл Build Spec називається «`buildspec.yml`» і повинен бути скопійований в корінь репозиторію вихідного коду. `Buildspec.yml` складається з пар ключ/значення для опису різних етапів збірки, представлені в послідовному порядку, які називають фазами (phases) збірки.

Вихідний код `buildspec.yml`:

CodeBuildProject:

Type: AWS::CodeBuild::Project

Properties:

Artifacts:

Type: CODEPIPELINE

Source:

Type: CODEPIPELINE

BuildSpec: |

version: 0.2

env:

variables:

APP_CODE_PATH_CONTAINER: "/var/www"
NETWORKS_DRIVER: "bridge"
COMPOSE_FILE: "docker-compose.yml"
COMPOSE_PATH_SEPARATOR: ":"
COMPOSE_PROJECT_NAME: "pehotin_ecommerce"
PHP_VERSION: "7.4"
PHALCON_VERSION: "3.4.5"
DOCKER_HOST_IP: "10.0.75.1"
PHP_IDE_CONFIG: "serverName=laradock"
COMPOSE_CONVERT_WINDOWS_PATHS: "1"
CHANGE_SOURCE: "false"
UBUNTU_SOURCE: "aliyun"
DOCKER_SYNC_STRATEGY: "native_osx"
PHP_FPM_FFMPEG: "false"
PHP_FPM_DEFAULT_LOCALE: "POSIX"
PHP_FPM_PUID: "1000" PHP_FPM_PGID: "1000"
NGINX_HOST_HTTP_PORT: "80"
NGINX_HOST_HTTPS_PORT: "443"
NGINX_HOST_LOG_PATH: "./logs/nginx/"
NGINX_SITES_PATH: "./nginx/sites/"
NGINX_PHP_UPSTREAM_CONTAINER: "php-fpm"
NGINX_PHP_UPSTREAM_PORT: "9000"
NGINX_SSL_PATH: "./nginx/ssl/"
phases:
install:
runtime-versions:
nodejs: 12
commands:
- curl -s https://getcomposer.org/installer | php
- mv composer.phar /usr/local/bin/composer

```

- php --version
- composer install
- cd themes/shopchik && npm install && npm run production && cd
../..
pre_build:
  commands:
    - $(aws ecr get-login --no-include-email)
    - TAG="$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION
    - IMAGE_URI="${REPOSITORY_URI}:${TAG}"
build:
  commands:
    - docker build --tag "$IMAGE_URI" .
post_build:
  commands:
    - docker push "$IMAGE_URI"
    - printf ' [{"name":"shopchik-app","imageUri":"%s"}]' "$IMAGE_URI"
> images.json

```

Для створення Task Definition спочатку треба увійти в обліковий запис за посиланням: <https://signin.aws.amazon.com/signin>.

1. На головній сторінці потрібно натиснути «Get started», щоб отримати доступ до консолі ECS.
2. Вибираємо «Task Definition» в області навігації.
3. У відкритому вікні натискаємо «Create new Task Definition».
4. У розділі «Create new Task Definition» вибираємо тип запуску «Fargate».
5. Натискаємо «Next step».
6. Тепер потрібно вказати налаштування для task definition і контейнерів. У діалоговому вікні «Add container» вказуємо назву контейнера (node-server) і вказуємо образ як: pehotin/nginx.

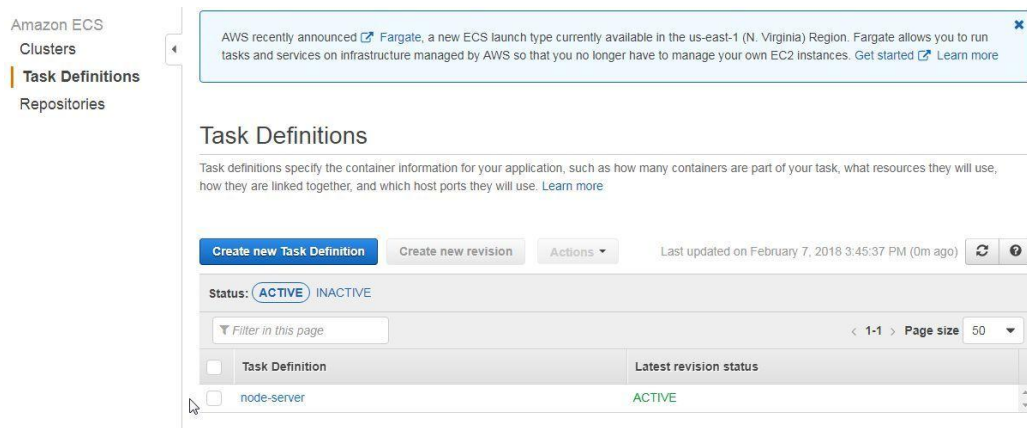


Рисунок 2.14 – Налаштування конфігурації (Task Definition)

Перед створенням служби необхідно налаштувати підключення до Інтернету в підмережах, які будуть використовуватися під час налаштування служби. У таблиці маршрутів (Route Table) перераховані маршрути. Потрібно додати маршрут з інтернет-шлюзом за замовчуванням:

1. Додаємо маршрут з пунктом призначення (Destination) як 0.0.0.0/0, який є IP-адресою.
2. Вибираємо Target як інтернет-шлюз.

Створення та тестування служби контейнерів (Container Service). Далі створюється служба контейнера ECS у кластері за замовчуванням, як показано на рисунку, по такому ж принципу як з task definition.

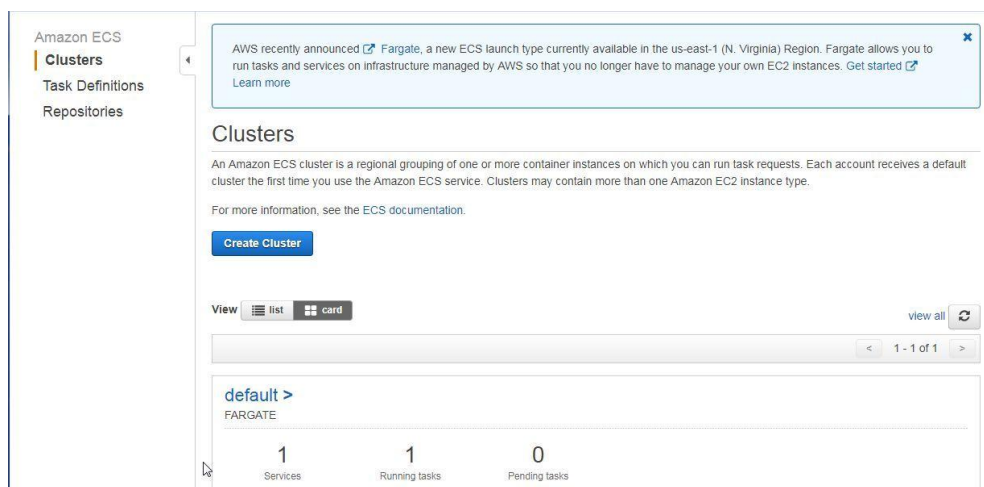


Рисунок 2.15 – Створення служби контейнера ECS

При типі запуску Fargate для кожної задачі надається еластичний мережевий інтерфейс (ENI). Копіюємо загальнодоступний IP-адреса задачі (Public IP of Task), який співпадає з загальнодоступною IP-адресою еластичного мережевого інтерфейсу, з розділу «Network» на сторінці відомостей про задачі або з консолі ENI. Відкриваємо URL-адресу <Public IP of Task>:80 в браузері, щоб викликати додаток rehotin.com.ua.

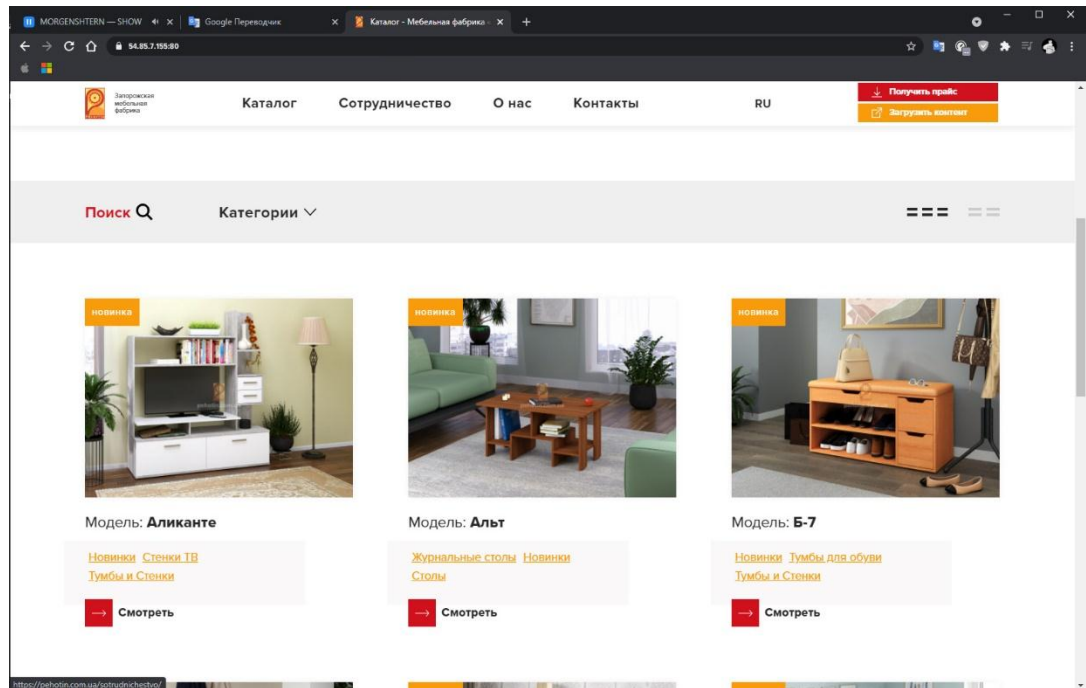


Рисунок 2.16 – Відповідь на виклик додатку

Створення і налаштування сегмента S3. CodePipeline для побудови вихідного коду для сайту rehotin.com.ua та розгортання образу Docker у службі ECS вимагає, щоб проект CodeBuild генерував “Output Artifacts”. Вихідні артефакти зберігаються у сегменті S3, який вибирається під час створення CodePipeline.

Створюємо новий сегмент S3 у консолі:

1. Вибираємо «Create bucket». Відкриється майстер створення.
2. В поле «Bucket name» вводимо DNS-сумісну назву.

Назва сегмента повинна бути: унікальною в усьому Amazon S3, довжиною від 3 до 63 символів, починатися з малої літери або цифри.

3. У розділі «Region» потрібно вибрати регіон AWS, в якому буде розміщений bucket. Вибираємо найближчий регіон, щоб звести до мінімуму затримки і витрати.
4. В налаштуваннях сегмента для блокування загального доступу вибираємо параметри «Block Public Access», які будуть застосовані до сегменту.
5. Натискаємо «Create bucket».

Наступний етап, створення проекту CodeBuild, який буде використовуватися для збірки вихідного коду в образ Docker. Після створення образу Docker він завантажується в концентратор Docker за допомогою CodeBuild. Щоб створити проект CodeBuild, відкриваємо сторінку <https://aws.amazon.com/ru/codebuild/>. Вибираємо «Build projects» і натискаємо «Create project».

Майстер створення проекту запускається. У розділі «Configure project» вказуємо назву проекту (node-server). У Source>Source Provider вибрали GitHub. Для репозиторію вибираємо пункт «Use a repository in my account» (Використовувати репозиторій в моєму профілі) і вибираємо rehotin/nginx як репозиторій. Значення для глибини клонування залишаємо за замовчуванням (Git clone depth) рівним 1.

Задаємо параметри для Webhook, Insecure SSL і Build Badge. При виборі Webhook код перебудовується кожен раз при оновленні коду в репозиторії GitHub. Параметр «Insecure SSL» викликає попередження SSL при збірці коду при підключенні до джерела проекту. Параметр «Build Badge» робить статус проекту видимим. У розділі «Environment: How to build» вибираємо «Environment image» в якості «Use an image managed by AWS CodeBuild». Операційну систему вибрано «Ubuntu». Для «Runtime» буде задіяно «Docker».

Для версії Runtime вибираємо «aws:codebuild/docker:17.09.0», що представляє версію Docker 17.9. Опція «Privileged» автоматично вибирається для середовища виконання Docker, оскільки вона потрібна для створення

образу Docker. Для специфікації збірки вибираємо «Use the buildspec.yml in the source code root directory». Назва Buildspec за замовчуванням buildspec.yml.

The screenshot shows the 'Environment: How to build' configuration page. It contains the following fields and options:

- Environment image***: Two radio buttons. The first, 'Use an image managed by AWS CodeBuild', is selected. The second is 'Specify a Docker image'.
- Operating system***: A dropdown menu with 'Ubuntu' selected.
- Runtime***: A dropdown menu with 'Docker' selected.
- Runtime version***: A dropdown menu with 'aws/codebuild/docker:17.09.0' selected. Below it is a link 'Choose a specific image version'.
- Privileged**: A checkbox that is checked, with the text 'Enable this flag if you want to build Docker images or want your builds to get elevated privileges.'
- Build specification**: Two radio buttons. The first, 'Use the buildspec.yml in the source code root directory', is selected. The second is 'Insert build commands'.
- Buildspec name**: A text input field containing 'buildspec.yml'.

Рисунок 2.17 - Налаштування середовища: How to build

Для «Certificate» вибираємо «Do not install any certificate» (Не встановлювати жодного сертифіката). Сертифікат не потрібен CodePipeline, але для більш безпечного CodeBuild може бути встановлений сертифікат із S3. Наступний крок, налаштування output artifacts в розділі «Artifacts». Так звані вихідні артефакти потрібні для CodePipeline. Вибираємо тип (type) як Amazon S3. Вказуємо назву папки сегмента S3 для використання, для цього задаємо шлях як “/”, який створює папку в корені сегмента. «Namespace type» вибрано як «Node». «Bucket name» виберемо із переліку, так як, він був налаштований раніше. Кеш вибрано як «No cache» (Без кешу).

Artifacts: Where to put the artifacts from this build project

Type* Amazon S3 ⓘ

Name node-server-fargate ⓘ

Path / ⓘ

Namespace type None ⓘ

Bucket name* Choose your S3 bucket ⓘ

codepipeline

codepipeline-us-east-1-342692972441

Cache

Type* No cache ⓘ

Рисунок 2.18 - Налаштування S3 Bucket для Artifacts

У розділі «Service role» вибирається параметр «Create a service role in your account». Ставимо чекбокс в пункті «Allow AWS CodeBuild to modify this service role» (Дозволити AWS CodeBuild змінювати цю роль служби), щоб її можна було використовувати з цим проектом збірки. Для VPC вибираємо «No VPC». Натискаємо «Continue».

Service role

Specify a service role that enables AWS CodeBuild to call dependent AWS services on your behalf. [Learn more.](#)

☐ Create a service role in your account

☒ Choose an existing service role from your account

Role name* codebuild-node-server-service-role ⓘ

☒ Allow AWS CodeBuild to modify this service role so it can be used with this build project

VPC

VPC* Select a VPC that your AWS CodeBuild project will access. [Learn more](#) ⓘ

No VPC ⓘ

► Show advanced settings

*Required

Cancel Continue

Рисунок 2.19 - Налаштування «Service Role» та налаштувань VPC

У розділі «Review» переходимо вниз і натискаємо «Create», щоб створити проект CodeBuild. Проект CodeBuild створюється і відображається в списку проектів Build.

Створена за замовченням service role, не включає деякі з необхідних дозволів. Налаштування service role необхідно змінити, додавши політику, яка додає ці дозволи: s3:GetObject і s3:PutObject.

Список вбудованих політик які треба додати:

```
{ "Version":"2012-10-17",
  "Statement":[
    "Effect":"Allow",
    "Action":["s3:PutObject","s3:GetObject"],
    "Resource":"arn:aws:s3:::*" }
```

Перед створенням і налаштуванням CodeBuild в CodePipeline потрібно протестувати проект CodeBuild, щоб можливо було внести виправлення в разі виникнення будь-яких помилок.

Натискаємо «Start build», щоб розпочати збірку. Проект CodeBuild запускається, і починається збірка коду. Коли проект CodeBuild буде завершено, деталі етапів будуть виведені зі статусом «Succeeded».

Phase details

	Name	Status	Duration	Completed
▶	SUBMITTED	Succeeded		1 minute ago
▶	PROVISIONING	Succeeded	21 secs	55 seconds ago
▶	DOWNLOAD_SOURCE	Succeeded	1 sec	53 seconds ago
▶	INSTALL	Succeeded		53 seconds ago
▶	PRE_BUILD	Succeeded		53 seconds ago
▶	BUILD	Succeeded	24 secs	28 seconds ago
▶	POST_BUILD	Succeeded	4 secs	24 seconds ago
▶	UPLOAD_ARTIFACTS	Succeeded		24 seconds ago
▶	FINALIZING	Succeeded	3 secs	21 seconds ago
▶	COMPLETED	Succeeded		

Build logs

Showing the last 10000 lines of build log below. [View entire log](#)

```
670d566a0d: Preparing
787c930753b4: Preparing
85f12712c80b: Preparing
```

Рисунок 2.20 - Деталі етапів та журнали збірки вказують на те, що CodeBuild завершено

Образ Docker pehotin/nginx, створений та оновлений проектом CodeBuild в Docker Hub.



Рисунок 2.21 - Образ Docker, створений і завантажений CodeBuild в Docker Hub

Були створені проекти для кожного з етапів CodePipeline, а саме:

- Репозиторій коду GitHub для вихідного коду.
- CodeBuild для збірки коду.
- Служби ECS Fargate для проміжного зберігання.

Тепер потрібно створити безпосередньо сам CodePipeline. Консоль CodePipeline знаходиться за адресою «<https://aws.amazon.com/codepipeline/>» заходимо та натискаємо «Get started».

Майстер створення конвеєра запускається. Спочатку задаємо назву конвеєра: node-server-fargate і натискаємо «Next step».

Наступним етапом потрібно вказати розташування джерела. В якості «Source provider» (постачальника вихідного коду) вибираємо GitHub.

Потім встановлюємо з'єднання з GitHub за допомогою кнопки «Connect to GitHub».

На вкладці «Source» в розділі «Connect to GitHub» вибираємо попередньо створений репозиторій GitHub.

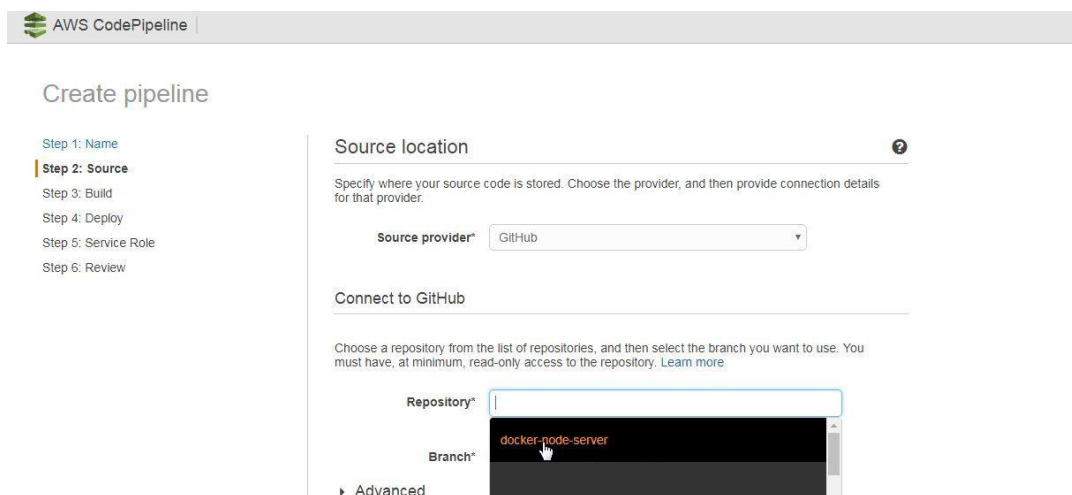


Рисунок 2.22 – Вибір попередньо створеного репозиторію

Задаємо гілку (branch) як головну (main).

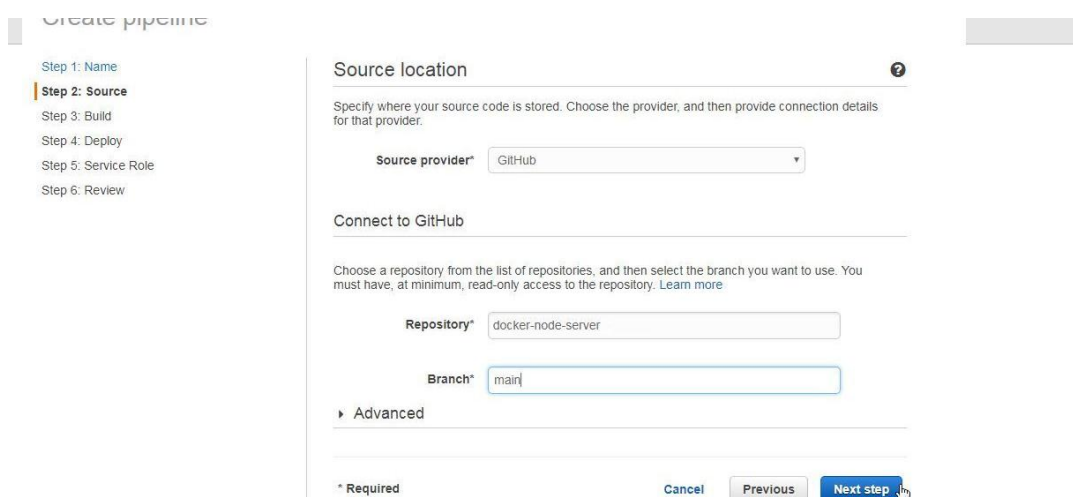


Рисунок 2.23 – Вибір гілки з GitHub

Налаштування Build. В якості постачальника збірки (Build provider) вибираємо AWS CodeBuild. Наступним відображається розділ який залежить від обраного постачальника збірки. Для AWS CodeBuild відображається розділ з докладною інформацією про CodeBuild. Для «Configure your project» вибираємо «Select an existing project». «Project name» задаємо як «node-server», назва проекту CodeBuild створеного раніше. Натискаємо «Next step».

Налаштування Deploy. Наступним потрібно вказати етап розгортання CodePipeline. У якості провайдера розгортання обираємо «Amazon ECS».

Тепер відображається розділ який залежить від обраного провайдера розгортання. У розділі Amazon ECS вибирається назва кластера в якому створюється служба ECS, вибираємо за замовчуванням (default).

В якості служби ECS для розгортання задаємо «Service name», тобто «node-server-service». Вказуємо назву файлу образу, як «pehotin/nginx». Натискаємо «Next step».

The screenshot shows the 'Deploy' step configuration in the AWS CodePipeline console. On the left, a sidebar lists steps from 'Step 1: Name' to 'Step 6: Review', with 'Step 4: Deploy' highlighted. The main panel is titled 'Deploy' and contains the following fields:

- Deployment provider***: A dropdown menu set to 'Amazon ECS'.
- Amazon ECS**: A section header with an information icon.
- Cluster name***: A text input field containing 'default'.
- Service name***: A text input field containing 'node-server-service'.
- Image filename**: A text input field containing 'pehotin/nginx'.

At the bottom, there are buttons for 'Cancel', 'Previous', and 'Next step' (which is highlighted with a mouse cursor). A legend indicates that fields with an asterisk are required.

Рисунок 2.24 – Відомості о конфігурації провайдера розгортання

Задаємо Service Role. Нова служба service role створюється в IAM автоматично при першому створенні CodePipeline. Згодом вона буде внесена в список для вибору.

The screenshot shows the 'AWS Service Role' creation page in the AWS IAM console. On the left, a sidebar lists steps from 'Step 1: Name' to 'Step 6: Review', with 'Step 5: Service Role' highlighted. The main panel is titled 'AWS Service Role' and contains the following elements:

- Role name***: A text input field with a dropdown menu showing 'AWS-CodePipeline-Service' selected.
- Create role**: A button to create the role.

At the bottom, there are buttons for 'Cancel', 'Previous', and 'Next step' (which is highlighted with a mouse cursor). A legend indicates that fields with an asterisk are required.

Рисунок 2.25 – Вибір Service Role служби

Перевіряємо конфігураційні налаштування і натискаємо на «Create pipeline».

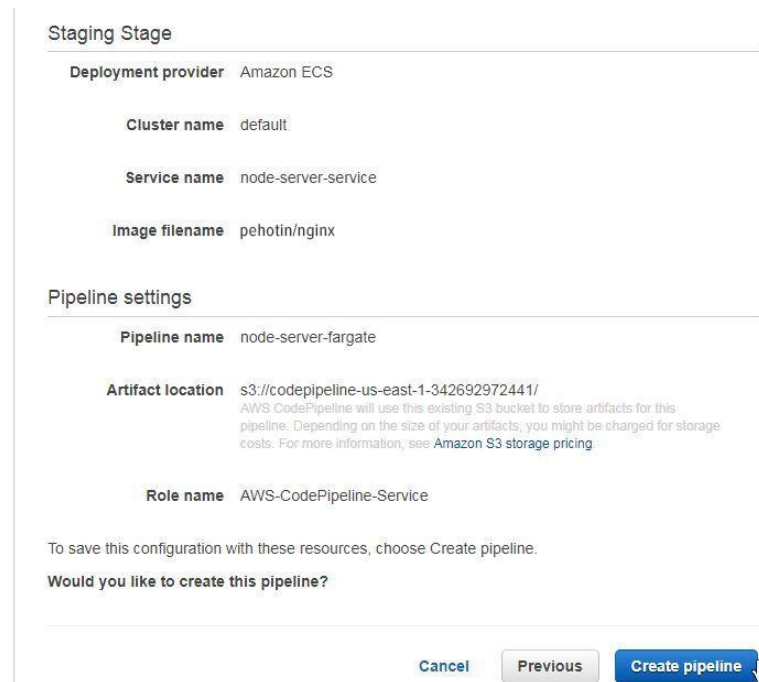


Рисунок 2.26 – Останній етап створення CodePipeline

Починається процес створення CodePipeline. Після створення CodePipeline запускається автоматично, що буде повідомлено статусом «In Progress» для етапу «Source».

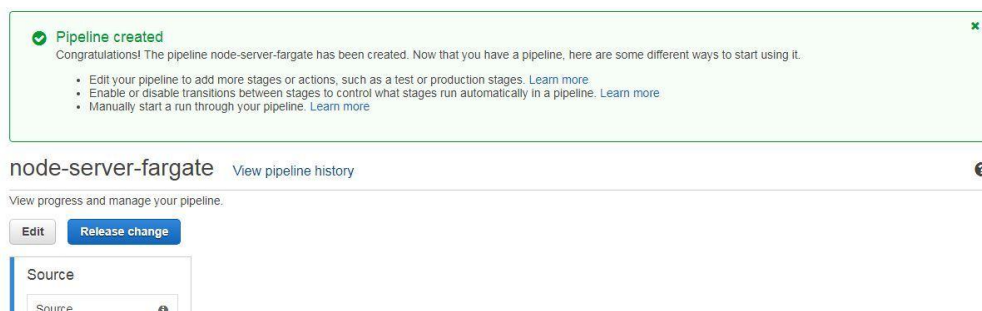


Рисунок 2.27 – Завершення створення CodePipeline, етап «Source» в процесі обробки

Коли етап «Source» завершується, його статус стає «Succeeded» і етап «Build» починає виконуватися, що позначається статусом «In Progress».

Після завершення етапу збірки починається останній етап інсталяції додатку в Staging середовище.

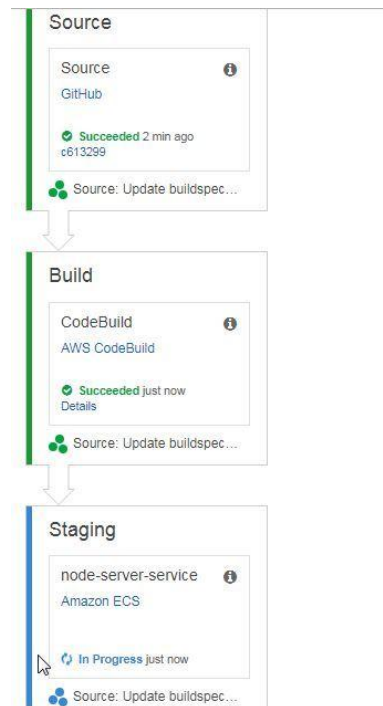


Рисунок 2.28 - Етап збірки виконаний успішно, етап Staging в процесі.

Після налаштування всіх етапів CodePipeline, додатково треба змінити налаштування для артефактів вводу/виводу (Input/Output Artifacts). Оскільки параметри артефактів вводу/виводу за замовчуванням не є такими, які потрібні для запуску CodePipeline. За замовчуванням вхідні артефакти для стадії «Staging» є вихідними артефактами зі стадії «Build.» Для CodeBuild, який створює тільки образ Docker і завантажує образ Docker в Docker Hub або Amazon ECR, етап CodeBuild не створює вихідних артефактів. Вхідні артефакти на етапі «Staging» повинні бути встановлені на вихідні артефакти стадії «Source».

Без зміни налаштувань для артефактів вводу/виводу етап підготовки завершиться помилкою. CodePipeline, який був запущений автоматично після створення, дає збій.

Щоб змінити артефакти вводу/виводу, натискаємо «Edit» на сторінці налаштувань CodePipeline. Після модифікації вхідних/вихідних артефактів натискаємо на «Save pipeline changes».

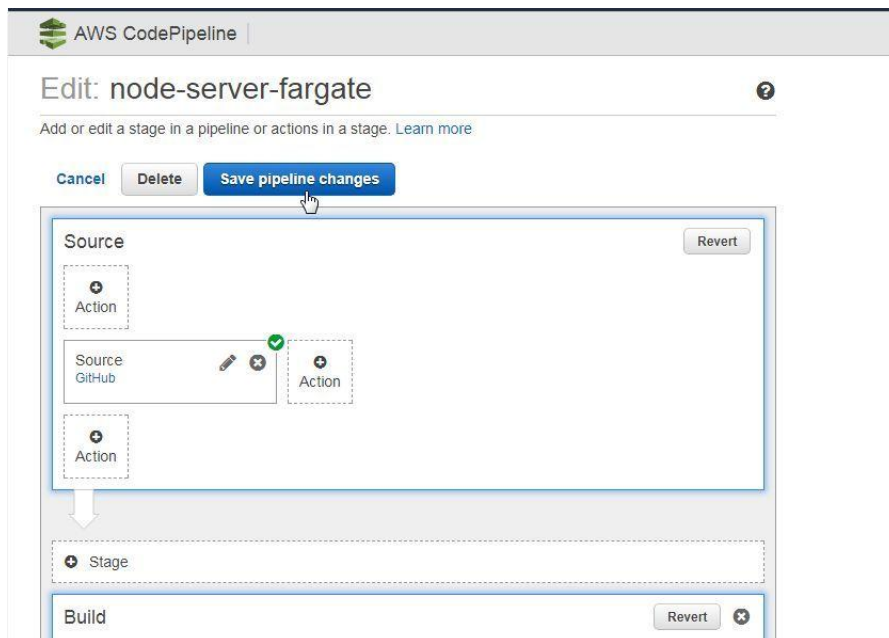


Рисунок 2.29 - Збереження змін конвеєра для Input/Output Artifacts

Щоб запустити CodePipeline після модифікації, треба натиснути «Release change». У діалоговому вікні «Release change» натискаємо «Release». Зміни, внесені до CodePipeline, застосовуються, і CodePipeline починає запускатися з самого початку, це показується статусом «In Progress» для етапу «Source». Статуси етапів «Build» та «Staging» відносяться до попереднього запуску і не вказані в поточному стані етапів.

Через деякий час, завдяки модифікаціям артефактів вводу/виводу, всі етапи CodePipeline завершуються успішно.

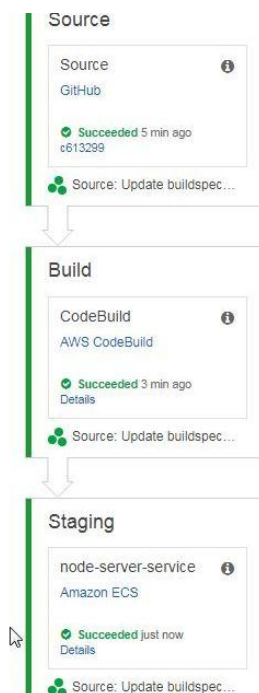


Рисунок 2.30 - Всі етапи CodePipeline успішно завершені

Попередня службова задача зупиняється, і задача, заснована на змінених конфігураціях, починає виконуватися, це показано статусом завдання «RUNNING».

Cluster : default Delete Cluster

Get a detailed view of the resources on your cluster.

Status **ACTIVE**

Registered container instances 0

Pending tasks count 0 Fargate, 0 EC2

Running tasks count 1 Fargate, 0 EC2

Active service count 1 Fargate, 0 EC2

Draining service count 0 Fargate, 0 EC2

Services Tasks **ECS Instances** Metrics Scheduled Tasks

Run new Task Stop Stop All Last updated on February 7, 2018 8:47:58 PM (0m ago) ↺ ?

Desired task status: **Running** Stopped

Filter in this page Launch type ALL < 1-1 > Page size 50

☐	Task	Task defi...	Container...	Last status	Desired s...	Started By	Group	Launch t...	Platform ...
☐	75484f91-...	node-serv...	--	RUNNING	RUNNING	ecs-svc/92...	service:no...	FARGATE	1.0.0

Рисунок 2.31 - Виконується нова задача

За таким же принципом, як в минулий раз, щоб викликати нову задачу використовуємо загальнодоступний IP-адрес з інтерфейсу «Elastic Network».

Ми отримаємо таку ж відповідь на виклик програми, як представлено на зображенні 2.16.

Може виникнути питання, в чому перевага запуску CodePipeline, якщо відповідь служби ECS така ж, як при прямому виклику служби без CodePipeline. Зазвичай вихідний код ECS, розгорнутого у робочому середовищі, необхідно періодично оновлювати, а це означає, що образ Docker в репозиторії GitHub потрібно буде перебудовувати. Також образ, який розгорнуто в службі ECS, необхідно оновити без припинення її безпосередньої роботи.

За допомогою CodePipeline процес запускається автоматично кожного разу, коли вносяться зміни до вихідного коду, без будь-якого втручання системного адміністратора.

Для демонстрації вносимо зміни в вихідний код в репозиторії GitHub. Натискаємо «Commit changes» в репозиторії. CodePipeline починає автоматично перезавантажуватися, це показано успішним завершенням етапу «Source» і виконанням етапу «Build». Після успішного завершення етапу «Build» запускається етап Staging, про що свідчить повідомлення про стан.

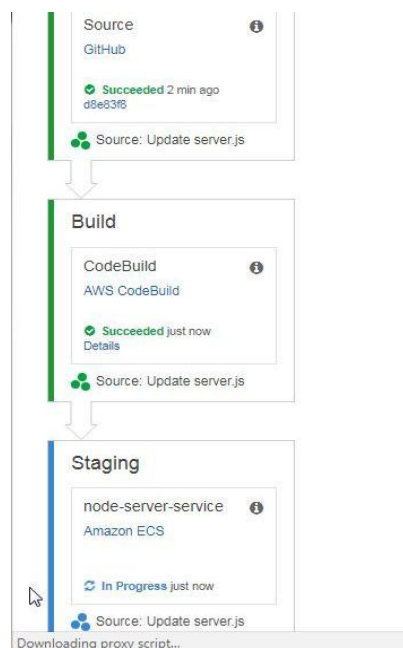


Рисунок 2.32 - Етап «Build» завершено і розпочато етап «Staging»

Після успішного завершення етапу інсталяції, запускаємо нову задачу і повторно перевіряємо працездатність сайту і зміни, які повинні бути внесені.

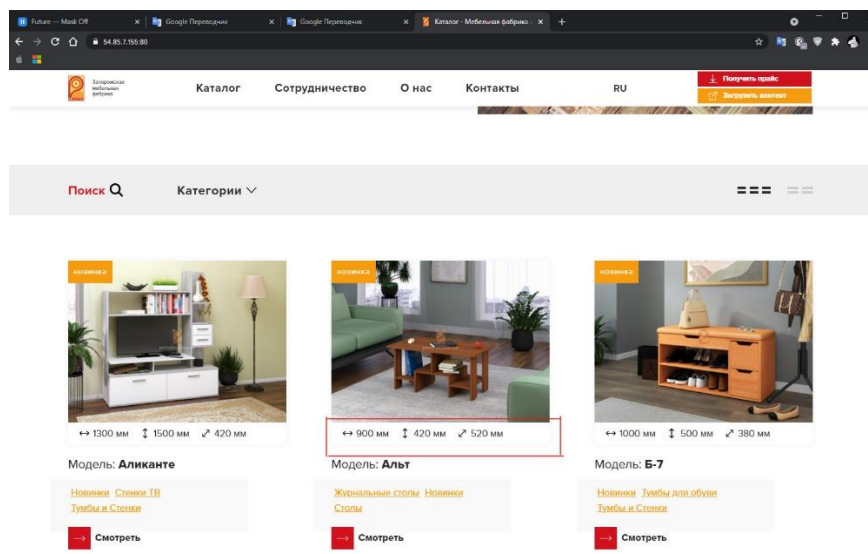


Рисунок 2.33 – Успішне завантаження додатку і відображення внесених змін

Як бачимо, зміна вихідного коду пройшла успішно та удосконалення у вигляді додаткового меню габаритів виробів з'явилися на сайті. Етап перенесення сайту на хмарний хостинг на базі Amazon ECS завершено успішно. Тепер має сенс зробити порівняння швидкості роботи сайту, до, і після переходу на новий хостинг.

3. АНАЛІЗ ШВИДКОСТІ РОБОТИ САЙТУ

3.1 Порівняння швидкості роботи сайту після переходу на AWS

Після завершення роботи над перенесення корпоративного сайту в нове середовище роботи, треба зробити тестування швидкості завантаження. Найпростіше це зробити за допомогою вбудованого інструменту Google Chrome для розробників Lighthouse.

Для цього заходимо на сторінку сайту і натискаємо клавішу F12. Повинно відкритися вікно консолі, в якому треба вибрати Lighthouse зі списку.

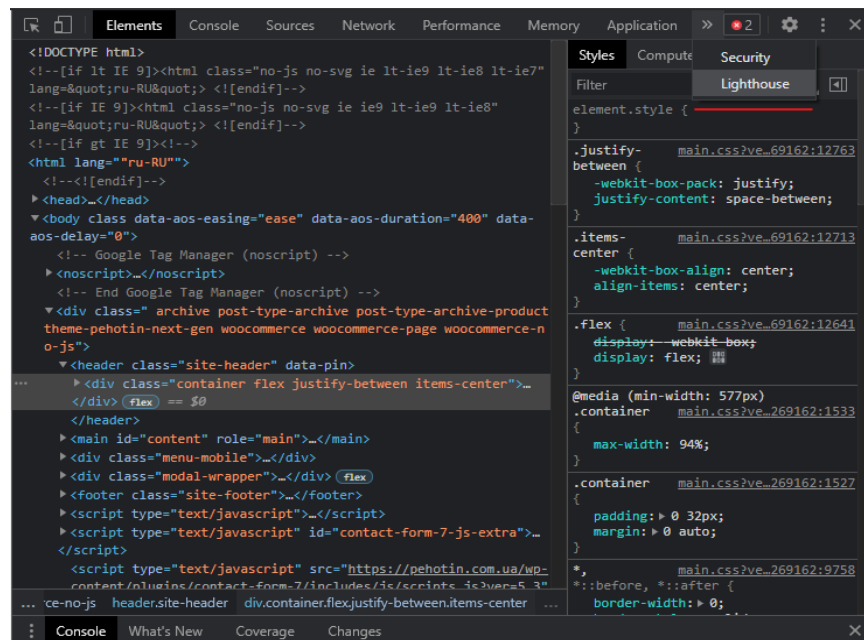


Рисунок 3.1 - Lighthouse - автоматизований інструмент для вимірювання якості веб-сторінок

Натискаємо кнопку «Generate report», починається процес тестування веб-сторінки.

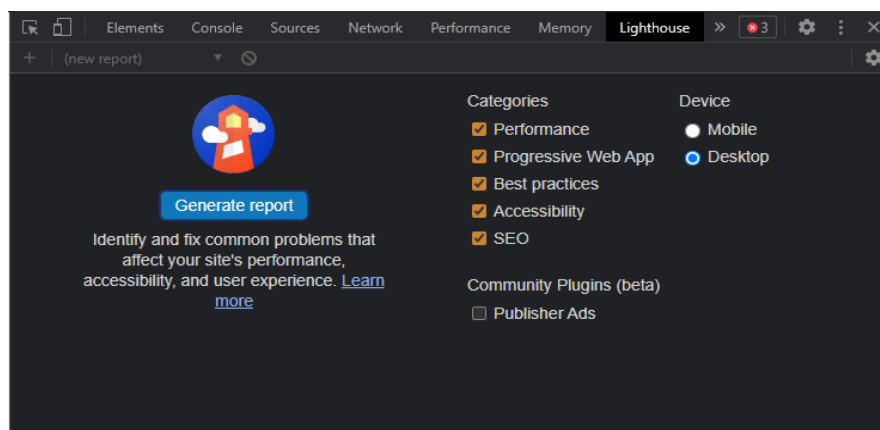


Рисунок 3.2 – Створення звіту якості роботи веб-сторінки

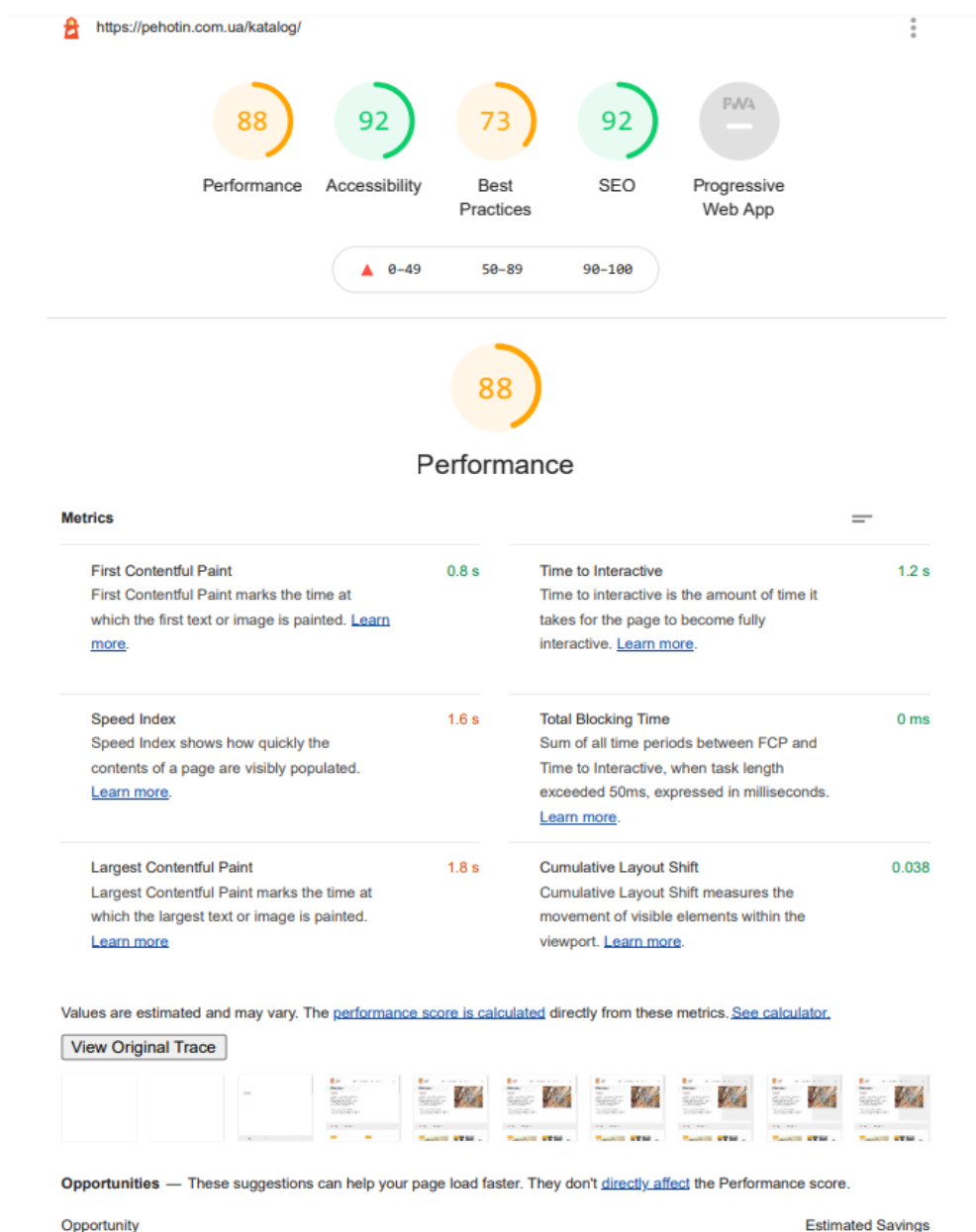


Рисунок 3.3 – Звіт якості роботи веб-сторінки створений Lighthouse

Як ми бачимо, показник Performance показує 88 балів зі 100 що є хорошим результатом у порівнянні з показниками сайту працюючому на старому хостингу, тести продуктивності якого були зроблені до переходу на хмарний хостинг.

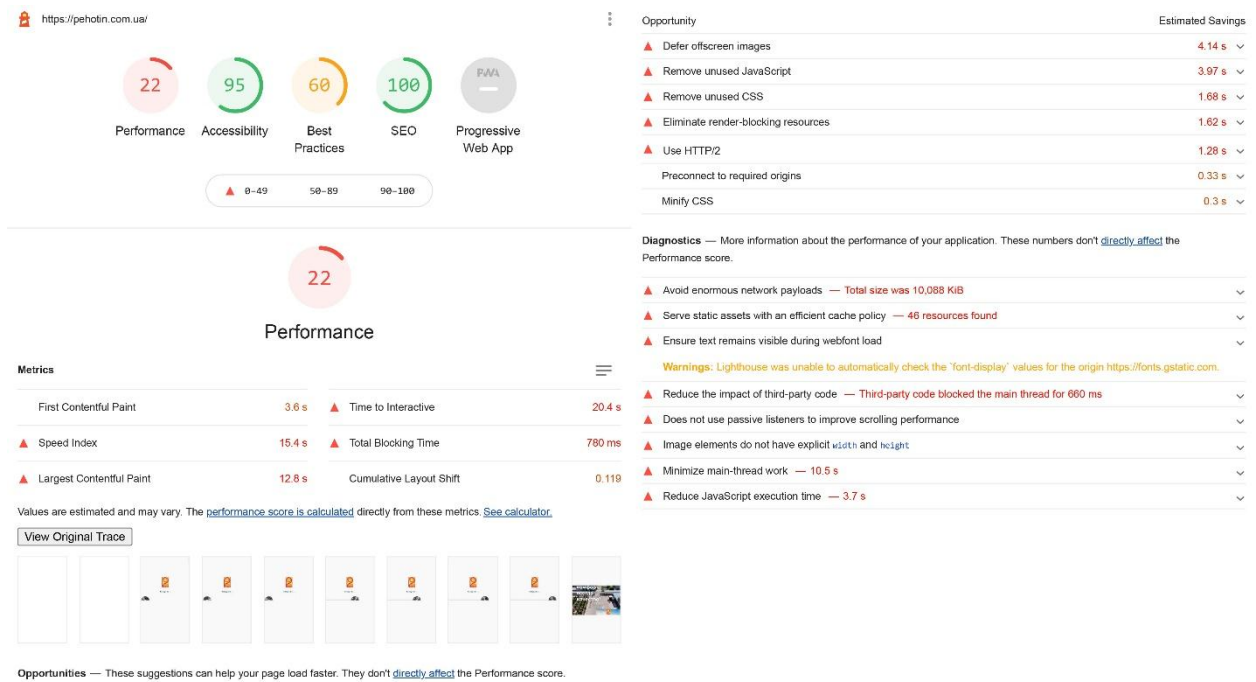


Рисунок 3.4 - Звіт якості роботи веб-сторінки до переходу на хмарний ХОСТИНГ

Можна сказати, що перехід був виправданий і швидкість завантаження веб-сторінок сайту в середньому виросла в 4 рази, виходячи з отриманих раніше результатів тестування швидкості роботи сайту на старому веб-хостингу, де індекс продуктивності дорівнював 22 балам.

3.2 Економічні аспекти переходу на нову платформу

Швидкість роботи показує хороший результат, але що стосується вартості, тут є деякі мінуси хмарний хостингів. Після місяця роботи AWS виставляє рахунок на 49\$ за користування своїми сервісами, що може здаватися великою сумою в порівнянні з дешевими хостинг провайдерами.

VAT Invoice		aws	
Email or talk to us about your AWS account or bill, visit console.aws.amazon.com/support . More information regarding your service charges is available by accessing your Billing Management Console.			
Account number:	746518461870	Invoice Summary	
Address:	pehotin ATTN: Dmitriy Pehotin	VAT Invoice Number:	EUINUA21-53735
		VAT Invoice Date:	March 3, 2021
		TOTAL AMOUNT	USD 48.80
		TOTAL VAT	EUR 0.00

This VAT Invoice is for the billing period February 1 - February 28, 2021

Invoice Summary	
AWS Service Charges	USD 48.80
Charges	USD 48.80
Net Charges (After Credits/Discounts, excl. Tax)	USD 48.80
VAT - 0%	USD 0.00
VAT in EUR (1 USD = 0.83139 EUR)	EUR 0.00

This supply is not subject to VAT. It is your liability to account for VAT under the reverse-charge procedure if required by your local VAT regulations

Рисунок 3.5 – Рахунок за користування службами AWS

Що стосується попереднього хостингу, його послуги коштували 560 грн. в місяць, що приблизно 20\$ і різниця з хмарний хостингом складає, приблизно 30\$, що в 2,5 разів більше. Використовувався не най дешевший тариф і це часто не враховується користувачами при вибори хостингу. Дешеві тарифи мають малу доступну ємкість дискового простору і обмежений процесорний час, ця проблема може негативно вплинути на доступності сайту в майбутньому, і потребувати перехід на більш дорогі тарифні плани.



Рисунок 3.6 - Рахунок за користування послугами попереднього хостингу

Враховуючі усі переваги роботи сайту на базі AWS, а це:

- На багато більша швидкість роботи.
 - Власний білий ір адрес.
 - Повна свобода налаштування.
 - Значно більший базова ємність дискового простору.
 - Можливість подальшого масштабування сайту за рахунок Amazon ECS.
- Є підстави зробити висновок, що перехід є економічно обґрунтованим.

3.3 Заходи щодо оптимізації роботи сайту

Приблизна швидкість завантаження веб-сторінки дорівнює 4,5 секунд, що більше рекомендованого значення в 3 або менше секунд, яке було визначено компанією Google як най ефективніший показник для збільшення показників охоплення аудиторії сайту.

Якщо подивитися на статистику ефективності за допомогою Google Search Console - веб-сервісу від Google, який дозволяє веб-майстрам перевіряти статус індексації та оптимізувати видимість своїх веб-сайтів, аудиторія сайту збільшується після переходу на новий хостинг.

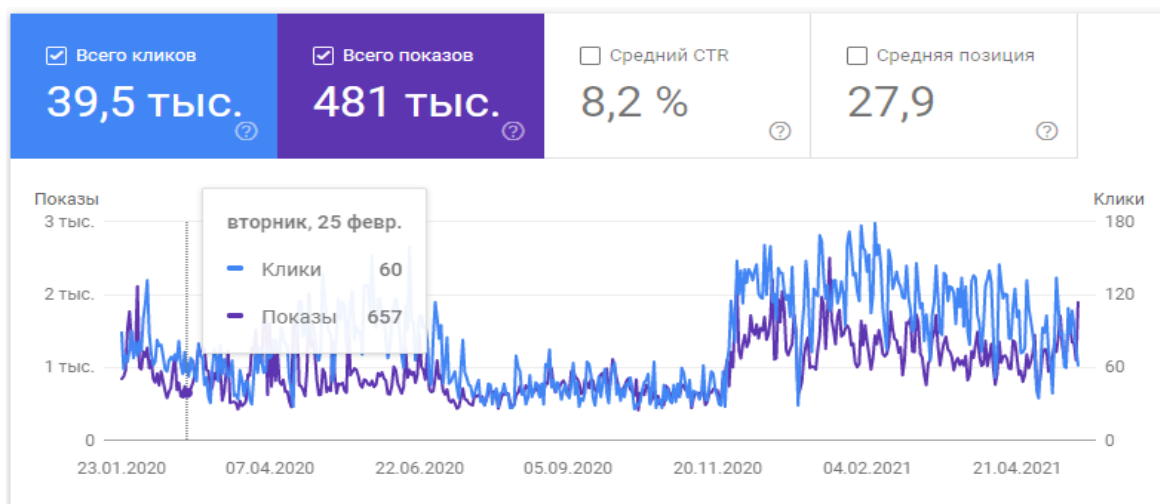


Рисунок 3.7 – Показники ефективності сайту до переходу на новий хостинг

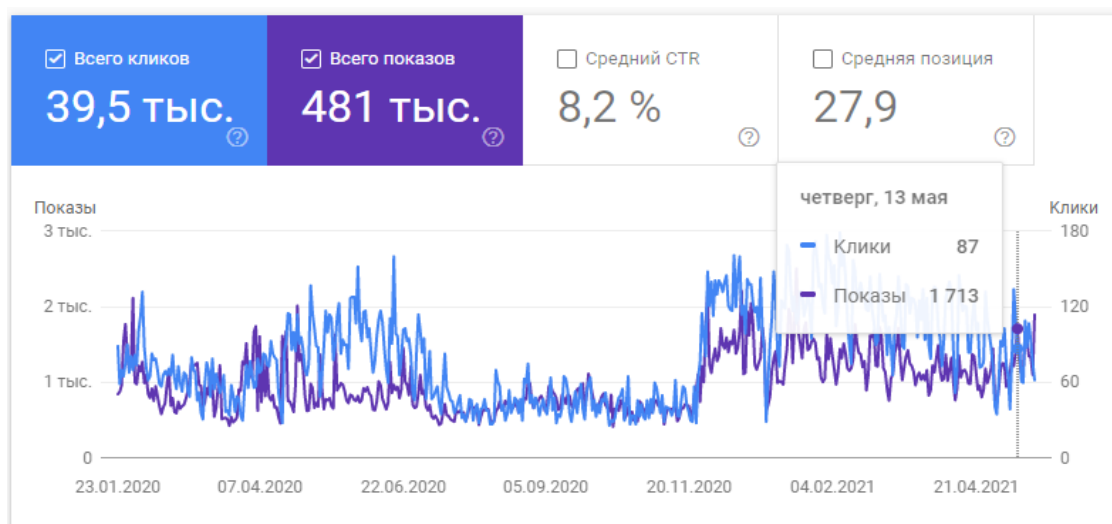


Рисунок 3.8 - Показники ефективності сайту після переходу на новий хостинг

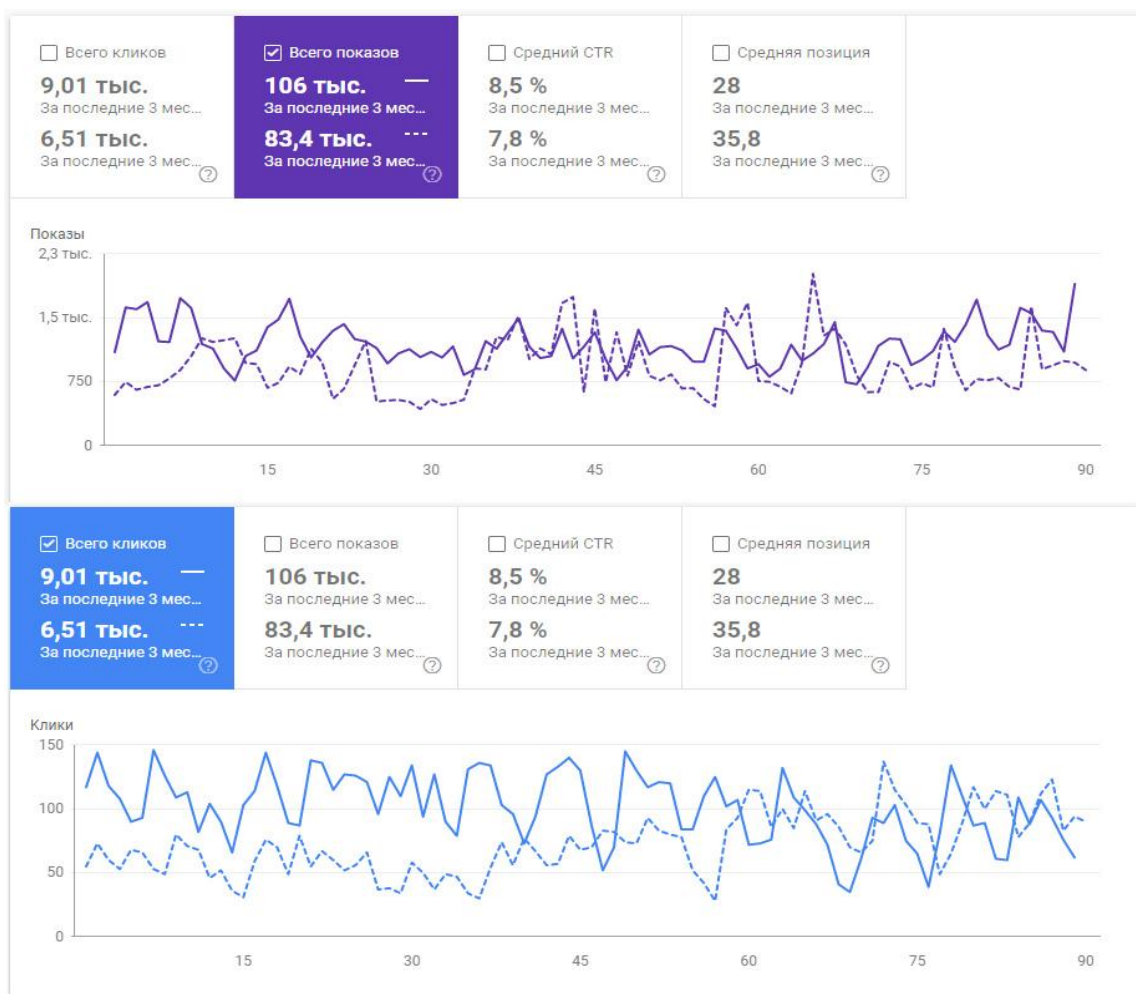


Рисунок 3.9 - Порівняння показників ефективності за 2021 і 2020 роки за 3 місячний період

Як бачимо з рисунку 3.9, графік показника «Покази» за 2021 рік (пряма лінія) знаходиться вище ніж графік за 2020 (пунктирна лінія) і становить 106 тисяч проти 83,4 тисячі показів відповідно.

Також приведено графік порівняння показника «Кліки» (відвідування), який теж показує зростання показників: 9 тисяч за 2021 рік проти 6,5 тисяч кліків за 2020 рік. В середньому ми бачимо постійне зростання крім останнього періоду в один місяць, що можна записати на сезонність представленої на сайті продукції або інші глобальні впливи.

Для того щоб продовжити впевнений рух по збільшенню показників ефективності сайту, можна скористатися рекомендаціями зі звіту про

зручність користування браузером Chrome, який можна сформувати за допомогою Google Search Console.

Проблеми для мобільних пристроїв, які можна вирішити для збільшення швидкості роботи сайту:

Низька швидкість 71 URL:

- Значення показника "LCP" занадто велике: більше 4 секунд для 41 сторінки.
- Проблема з CLS: значення показника вище порогового (0,25) для 30 сторінок.

Середня швидкість 44 URL:

- Значення показника "LCP" велике: більше 2,5 секунд для 34 сторінок.
- Проблема з CLS: значення показника вище порогового (0,1) для 51 сторінок.

LCP (отрисовка великого контенту). Час, за який виконується отрисовка найбільшого видимого елемента на сторінці.

CLS (сукупне зміщення макета). Дані про те, наскільки елементи інтерфейсу сторінки зміщуються під час її завантаження.

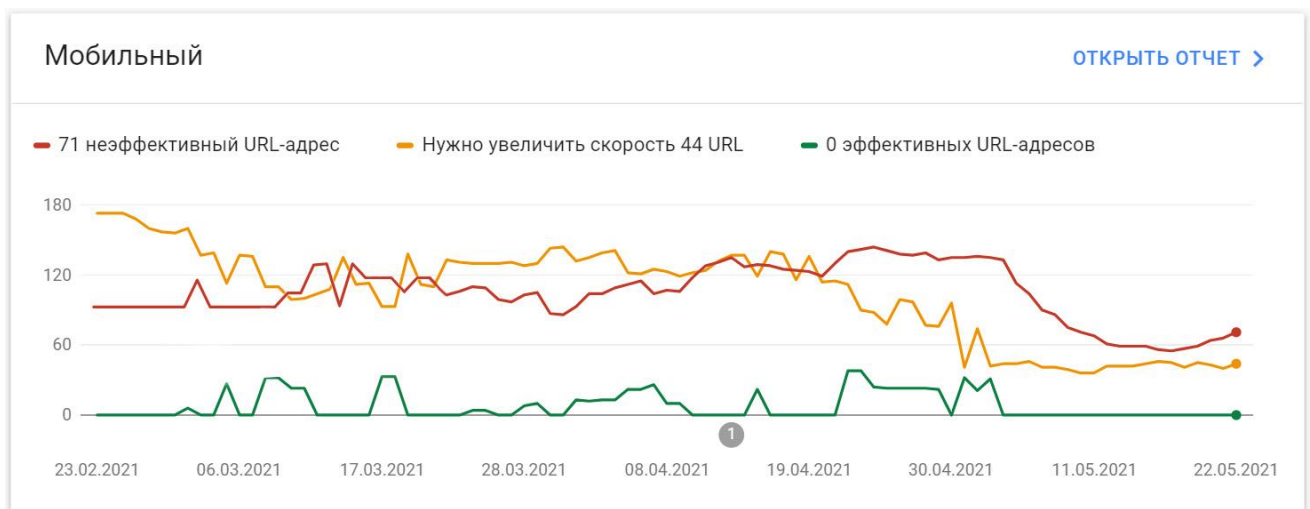


Рисунок 3.10 – Графік проблемних URL на мобільних пристроях швидкість яких треба збільшити, графік демонструє зменшення кількості неефективних URL після переходу на AWS

ВИСНОВКИ

Описано сучасні тенденції в керуванні обчислювальними ресурсами, новий принцип розробки DevOps, який значно скорочую терміни розробки і розгортання програмного забезпечення. Нові технології в керуванні інфраструктурою завдяки, всього лише, текстових файлів конфігурацій, що значно спрощує роботу системний адміністраторам і скорочує витрати компанії на ІТ сервіси.

Описано роботу контейнеризатора додатків Docker і системи контролю версій ПЗ GIT, які є основою для роботи додатків на базі середовища Amazon ECS. Docker має переваги коли розробляється додаток і потрібно надати код разом з усіма його складовими. Ця проблема вирішується через створення незалежності ПЗ від системи. Docker розділяє ядро ОС на різні контейнери, що працюють як окремі процеси, і може бути запущений в будь якому середовищі.

Наведено приклад роботи архітектури середовища Amazon ECS.

Створено віддалений репозиторій GitHub, для зберігання вихідного коду додатку, який має переваги такі як: безкоштовність і повна сумісність с сервісом Amazon ECS.

Виконано розгортання контейнерного додатку Docker в робоче середовище ECS Fargate. Продемонстровано оновлення вихідного коду для образу Docker і розгортання нового образу Docker в працюючій контейнерній службі без її зупинки з використанням AWS CodePipeline і тестів AWS CodeBuild.

Порівняно ефективність роботи сайту за допомогою вбудованого інструменту Lighthouse, який показав, що ефективність роботи сайту після переходу на хмарний хостинг в середньому виросла в 4 рази, також були представлені графіки відвідуваності сайту, показники яких, після переходу, показують збільшення аудиторії.

Наведені економічні аспекти переходу на нову платформу та запропоновані покращення для роботи сайту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Что такое Docker, и как его использовать? [Электронный ресурс]. - Режим доступа: <https://proglib.io/p/docker>.
2. Docker overview [Электронный ресурс]. - Режим доступа: <https://docs.docker.com/get-started/overview/>.
3. What are the different types of web hosting? [Электронный ресурс]. - Режим доступа: <https://www.techradar.com/web-hosting/what-are-the-different-types-of-web-hosting>.
4. Что такое DevOps? [Электронный ресурс]. - Режим доступа: <https://aws.amazon.com/devops/what-is-devops/>.
5. Continuous integration vs. continuous delivery vs. continuous deployment. [Электронный ресурс]. - Режим доступа: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>.
6. What Is Infrastructure as Code? How It Works, Best Practices, Tutorials [Электронный ресурс]. - Режим доступа: <https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/>.
7. Начало работы с Git: подробный гайд для новичков [Электронный ресурс]. - Режим доступа: <https://javarush.ru/groups/posts/2683-nachalo-rabotih-s-git-podrobnihy-gayd-dlja-novichkov>.
8. What is Amazon Elastic Container Service? [Электронный ресурс]. - Режим доступа: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>.
9. Deploying Docker Containers Using an AWS CodePipeline for DevOps [Электронный ресурс]. - Режим доступа: <https://www.infoq.com/articles/aws-codepipeline-deploy-docker/>.
10. Learning AWS - Second Edition [Текст] : інформ. журн. / видавництво Packt. – 2018, лютий -М. : Packt, 2018-, №1.

11. Ресурсы DevOps. [Электронный ресурс]. - Режим доступа: <https://aws.amazon.com/ru/devops/resources/>.
12. Amazon ECS clusters [Электронный ресурс]. - Режим доступа: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/clusters.html>.
13. Amazon ECS services. [Электронный ресурс]. - Режим доступа: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_services.html.
14. Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts. [Электронный ресурс]. - Режим доступа: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cluster-regions-zones.html>.
15. Task definition parameters. [Электронный ресурс]. - Режим доступа: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html.
16. Amazon S3. [Электронный ресурс]. - Режим доступа: https://docs.aws.amazon.com/s3/?id=docs_gateway.
17. What is Amazon EC2 Auto Scaling? [Электронный ресурс]. - Режим доступа: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html>.