Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем і комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Автоматизація процесу розгортання клієнт-серверної системи для керування цифровими активами DSpace 7»

Виконав студент 4 курсу, групи ІСТ-41
спеціальності: 126
„Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Гадупяк Ю.Ю.
(прізвище, ініціали)

Керівники: Івасюк Р.В.
(прізвище, ініціали)

Павлюк У.В.
(прізвище, ініціали)

Рецензент: Процик Ю.С.
(прізвище, ініціали)

Львів-2023

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІСКМ
Сторожук О.Л.
"21" 11 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гадупяк Юрій Юрійович
(прізвище, ім'я, по батькові)

1. Тема роботи *«Автоматизація процесу розгортання клієнт-серверної системи для керування цифровими активами DSpace 7»*
керівники роботи *Івасюк Р.В., Павлюк У.В., к.е.н., доц.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"21"11 2022 року № С-521*

2. Термін подання студентом роботи *10.06.2023р.*

3. Вихідні дані до роботи: *Постановка завдання та його формалізація. Технічне завдання до дипломної роботи. Документація для роботи з системою DSpace.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1) Стан проблемної області;
2) Інформаційне та математичне забезпечення;
3) Програмне та технічне забезпечення;
4)Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалів до доповіді

6. Дата видачі завдання 23 листопада 2022 року

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Етапи бакалаврської дипломної роботи	Термін виконання етапів роботи	Примітка
1.	1. Огляд сучасного стану задачі, методів і засобів її вирішення. 2. Формування функціональних вимог та постановка задачі проекту. 3. Оформлення першого розділу пояснювальної записки.	23.11.2022 10.12.2022	Виконано
2.	1. Огляд інформаційного забезпечення. 2. Огляд існуючих систем. 3. Оформлення другого розділу пояснювальної записки.	11.12.2022 26.12.2022	Виконано
3.	1. Вибір програмних засобів для реалізації системи.	03.01.2023 10.02.2023	Виконано
4.	Програмна реалізація інформаційної системи. Оформлення третього розділу пояснювальної записки.	11.02.2023 21.04.2023	Виконано
5.	Технічна та апаратна організація проекту.	22.04.2023	Виконано
6.	Здача пояснювальної записки на кафедру.	07.06.2023	Виконано

Студент _____ ***Гадупяк Ю.Ю.***
(підпис) (прізвище та ініціали)

Керівник роботи _____ ***Івасюк Р.В.***
(підпис) (прізвище та ініціали)

Керівник роботи _____ ***Павлюк Ю.В.***
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 44 сторінок пояснювальної записки, 3 рисунків, 1 додаток, 15 джерел.

Об'єктом розгляду в даній роботі є клієнт-серверна система Dspace 7.4 та інфраструктура для розгортання коду. Предмет роботи створення системи та інфраструктури для автоматизації процесів розгортання програмного продукту. Методом дипломної роботи створення системи контейнеризації та інфраструктури розгортання забезпечення інструменту для швидкої та ефективної системи розгортання програмного продукту Важливою частиною роботи є налаштування інфраструктури та забезпечення автоматизації процесів. Для досягнення мети були вирішені наступні завдання: 1. Розглянуто та описано роботу клієнт-серверної системи Dspace. 2. Проведено аналіз рішень. 3. Спроектовано інфраструктуру на основі хронологічного сервісу SSR. 4. Впроваджено систему контейнеризації та автоматичного розгортання за допомогою систем Docker та Docker-Compose. 5. Розроблено та протестовано систему. Ключові слова: клієнт-сервер, хронологічні технології, розгортання, GCP, Docker, Docker-compose.

Ключові слова: клієнт-сервер, хронологічні технології, розгортання, GCP, Docker, Docker-compose.

ABSTRACT

The thesis contains 44 pages of explanatory notes, 3 figures, 1 appendix, 15 sources.

The object of consideration in this work is the Dspace 7.4 client-server system and the infrastructure for code deployment. The subject of the work is the creation of a system and infrastructure for automating software product deployment processes. The thesis method is the creation of a containerization system and deployment infrastructure to provide a tool for a fast and effective software product deployment system. An important part of the work is the configuration of the infrastructure and ensuring process automation. To achieve the goal, the following tasks were solved: 1. The operation of the Dspace client-server system was considered and described. 2. A

solution analysis was performed. 3. The infrastructure was designed based on the SSR chronological service. 4. A containerization and automatic deployment system using the Docker and Docker-Compose systems was implemented. 5. The system was developed and tested. Keywords: client-server, chronological technologies, deployment, GCP, Docker, Docker-compose.

Keywords: client-server, chronological technologies, deployment, GCP, Docker, Docker-compose.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити систему для втоматизованого розгортання програмного продукту для інстутиційних репозиторіїв DSpace.

Зробити огляд програмного продукту, створити інфраструтуру на хмарному сервісі для розгортання програмного продукту.

Для вирішення завдання використати наступні сервіси і технології:

- для хмарного сервісу Google Cloud;
- інституційний репозиторій DSpace7;
- для контейнеризації програмного продукту Docker;
- для розгортання і запуску аплікації Docker Compose.

Перевірити роботу програмного продукту. Оформити пояснювальну записку відповідно до методичних вказівок.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface, інтерфейс програмування застосунків.

AJAX - Asynchronous JavaScript and XML, технологія асинхронного оновлення веб-сторінок.

BOAI - Budapest Open Access Initiative, Будапештська ініціатива відкритого доступу.

CSV - Comma-Separated Values, формат табличних даних.

DDF - DSpace Development Fund, фонд розвитку DSpace.

HAL - Hypertext Application Language, формат представлення гіпермедійних

HTTP - HyperText Transfer Protocol, протокол передачі гіпертексту.

IR - Institutional Repository, інституційний репозиторій.

OAI-PMH - Open Archives Initiative Protocol for Metadata Harvesting, протокол збору метаданих.

ORM - Object Relational Mapping, технологія відображення об’єктів у реляційні бази даних.

P2P - Peer-to-Peer, технологія обміну даними між рівноправними вузлами.

REST - Representational State Transfer, архітектурний стиль побудови веб-сервісів.

RSS - Really Simple Syndication, формат веб-каналів для розповсюдження новин.

SCOSS - Global Sustainability Coalition for Open Science Services, коаліція підтримки сервісів відкритої науки.

SPARQL - SPARQL Protocol and RDF Query Language, мова запитів до RDF-даних.

ВСТУП

Побудований на стеку сучасних фреймворків і залежностей, DSpace 7 є найбезпечнішим випуском DSpace за всю історію. Спільнота розробників підняла планку тестування та критеріїв прийняття запитів на виправленн програмного продукту. Повністю автоматизоване сканування та автоматичне оновлення застарілих залежностей забезпечує ефективний та дієвий спосіб реагування на нові повідомлення про загрози безпеці.

Вся бізнес-логіка забезпечується і захищається на рівні API функцій. Розділення на back end і front end DSpace 7 забезпечує більший розподіл та реалізацію стратегій кластеризації, які були неможливими в попередніх версіях.

DSpace 7 має сучасний інтерфейс користувача Angular. Динамічний і швидкий, DSpace 7 пропонує найкращий досвід в розробці інтерфейсу як для аудиторії, що отримує доступ до контенту інституційного репозитарію, так і для адміністраторів, відповідальних за управління та розвиток колекцій.

Завдяки Bootstrap 4 тепер легше налаштовувати і розширювати теми і брендинг, ніж у попередніх інтерфейсах. Angular UI веде спільноту DSpace в еру сучасного Інтернету.

DSpace, дозволяє дослідникам і науковцям публікувати документи і дані. Хоча DSpace має деякі спільні риси з системами управління контентом і системами управління документами, програмне забезпечення репозиторію DSpace слугує конкретній потребі як система цифрових архівів, орієнтована на довгострокове зберігання, доступ і збереження цифрового контенту, що робить DSpace програмним забезпеченням вибору для академічних, некомерційних і комерційних організацій, які створюють відкриті цифрові репозитарії. Це безкоштовне і просте у встановленні програмне забезпечення "з коробки", яке можна повністю налаштувати відповідно до потреб будь-якої організації.

РОЗДІЛ 1. СУЧАСНИЙ СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Перспективи використання інститутських репозиторіїв

На сьогоднішній день у світі йде безперервне зростання кількості споживачів, які користуються досягненнями в галузі технологій пошуку та доставки наукової інформації. Це зумовлено розповсюдженням комп'ютерних мереж, збільшенням обсягу інформаційного потоку й усвідомленням світовою спільнотою цінності інформації. Усе це призвело до створення спеціалізованих центрів, інформаційних систем і ресурсів, що підвищують швидкість і зручність отримання необхідної інформації. Інституційні сховища зараз створюються для того, щоб управляти, зберігати та підтримувати цифрові активи, інтелектуальну продукцію, та історію установ, які раніше були недоступні для громадськості та пошукових систем.

Наукова комунікація здійснюється двома способами: публікації та конференції, завдяки чому результати наукових, досліджень стають доступними науковій спільноті, соціальних та економічних чинників на наукову комунікацію, засновану на традиційних каналах поширення результатів наукових досліджень, зумовлюють перехід від парадигми наукових досліджень, зумовлюють перехід від парадигми традиційної публікації до створення відкритих архівів для наукової спільноти.

Відкритий доступ до результатів досліджень (Open Access to Research) - це спосіб наукового спілкування шляхом реалізації права автора твору на доведення до загального відома таким чином, що будь-яка особа може отримати доступ до твору з будь-якого місця і в будь-який час за власним вибором. Відкритий доступ посилює науковий вплив автора і не суперечить авторському праву.

Архіви відкритого доступу надають нові можливості, такі як: індексування таблиць, діаграм тощо, додавання новин через систему RSS, часткове оновлення (AJAX) веб-сторінки, обмін інформацією за технологією peer-2-peer (P2P). Тому електронні архіви відкритого доступу легше супроводжувати, вони зручніші в управлінні, ніж традиційні.

14 лютого 2002 року в Будапешті на зборах інституту "Відкрите суспільство суспільство" було ухвалено документ, у якому вперше сформульовано основні принципи відкритого доступу: Будапештська ініціатива "Відкритий доступ" (Budapest Open Access Initiate - BOAI): "Це право користувача читати, вивантажувати, копіювати, поширювати, друкувати, здійснюdпошук або проставляти гіперзв'язки до повного тексту статей".

Другий документ, який визначає основні принципи відкритого доступу, ухвалений у жовтні 2003 року, - це Берлінська декларація про відкритий доступ до наукових і гуманітарних знань (Open Access to Knowledge in the Sciences and Humanities).

Правовласники та автори, які підписали Берлінську декларацію, надають користувачам необмежене право доступу, право на копіювання, розповсюдження, використання, передачу та публічну демонстрацію. Користувачі також мають право на тиражування з відповідним зазначенням авторства, а також на виготовлення друкованих копій для особистого користування.

Надалі на Берлінських конференціях освітнім і науковим установам було запропоновано такі рекомендації:

1. рекомендувати вченим, науковим співробітникам і дослідникам депонувати свої опубліковані роботи в репозиторії відкритого доступу;

2. підтримувати вчених і дослідників у їхньому прагненні опублікувати результати наукових робіт в існуючих журналах відкритого доступу з даної тематики;

3. надавати підтримку для створення та розвитку таких журналів.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Опис системи репозиторію Dspace 7

Програмне забезпечення репозиторію з відкритим вихідним кодом DSpace було розроблено для задоволення потреб академічних, некомерційних і комерційних організацій, що створюють відкриті цифрові репозиторії, такі як інституційні репозиторії в університетських бібліотеках (IR).

DSpace - дозволяє організаціям:

1) збирати та описувати цифрові матеріали за допомогою модуля робочого процесу подання або різноманітних програмних опцій введення;
2) поширювати цифрові активи організації в Інтернеті за допомогою пошукової системи;
3) зберігати цифрові активи в довгостроковій перспективі.

DSpace складається з фронтенду (інтерфейс користувача) та бекенду (REST API). Нижче наведено короткий огляд технологій, що використовуються для кожної з них.

Фронтенд DSpace надає користувачам інтерфейс, який дозволяє людям взаємодіяти з DSpace. Він вимагає наявності серверної частини DSpace і не може бути запущений автономно.

Фронтенд DSpace побудований на платформі Angular і написаний на мові Typescript. Він використовує Bootstrap і HTML5 для оформлення/стилізації. Фронтенд також використовує Angular Universal для "рендерингу на стороні сервера", що дозволяє йому функціонувати навіть тоді, коли Javascript недоступний у браузері.

DSpace Backend надає REST API, який необхідний для DSpace Frontend. Він також надає додаткові машинні інтерфейси для взаємодії з даними в DSpace, такі як OAI-PMH, SWORDv2 Server, SWORDv1 Server і різні інструменти командного рядка (CLI). Бекенд DSpace можна запускати

автономно, але він не забезпечує дружнього веб-інтерфейсу (саме тому рекомендується використовувати фронтенд DSpace).

Бекенд DSpace побудований на Spring Boot, написаний на Java. Частина REST API бекенда побудована на Spring Technologies, включаючи Spring REST, Spring HATEOAS, і узгоджується з Spring Data REST. REST API використовує Spring Data REST Hal Browser як базовий веб-інтерфейс для вивчення REST API.

Програмне забезпечення DSpace фінансово підтримується спільнотою: членськими внесками, сертифікованими партнерами, а також з різних заходів зі збору коштів. Нинішніми донорами та кампаніями є Фонд розвитку DSpace (DDF) та Глобальна коаліція зі сталого розвитку відкритих наукових сервісів (SCOSS).

Фонд розвитку DSpace (DDF)

Починаючи з квітня 2022 року, DSpace радий повідомити про запуск Фонду розвитку DSpace (DDF).

DSpace розпочав успішну кампанію зі збору коштів для прискорення розробки версії 7.0, яка стала важливою віхою в історії DSpace. Значною мірою завдяки щедрим внескам установ-членів ми змогли інвестувати понад 300 000 доларів США в цільову розробку, щоб уможливити прискорений випуск DSpace 7 у серпні 2021 року та кількох наступних точкових випусків.

Протягом майже 20 років розробники покладалися виключно на волонтерів, які створювали код для DSpace. Щоб забезпечити досягнення наших амбітних цілей версії 7.0, керівництво DSpace затвердило план прямого фінансування робіт з розробки DSpace 7. Фінансуючи роботу з розробки, керівництво DSpace може більш передбачувано планувати випуски і краще підтримувати нагляд спільноти за результатами розробки. Хоча внески волонтерів дуже вітаються і надзвичайно допомагають у розвитку DSpace, фінансована робота з розробки вивела DSpace на абсолютно новий рівень.

Глобальна коаліція зі сталого розвитку сервісів відкритої науки (SCOSS) обрала DSpace для участі у третьому раунді оголошень. SCOSS, створена у 2017 році, - це мережа впливових організацій, які прагнуть допомогти

забезпечити безпеку інфраструктури відкритого доступу та відкритого коду в майбутньому, визначаючи некомерційні сервіси, що мають важливе значення для відкритої науки, та надаючи кваліфіковані рекомендації щодо того, які з цих сервісів слід розглянути для фінансової підтримки. Щорічно SCOSS закликає спільноту відкритої науки фінансово підтримати протягом трьох років рекомендовану інфраструктуру, яку SCOSS вважає важливою.

Станом на вересень 2021 року SCOSS успішно завершила два раунди збору коштів і нещодавно оголосила про третій раунд та активно залучає фінансування для 3 програм з відкритим вихідним кодом, одна з яких Dspace

SCOSS розуміє важливість DSpace і зусиль світової спільноти, спрямованих на підтримку, подальший розвиток і підтримку DSpace.

2.2. Архітектура *DSpace*

Система DSpace організована в три рівні, кожен з яких складається з ряду компонентів зображених на рисунок 2.1.

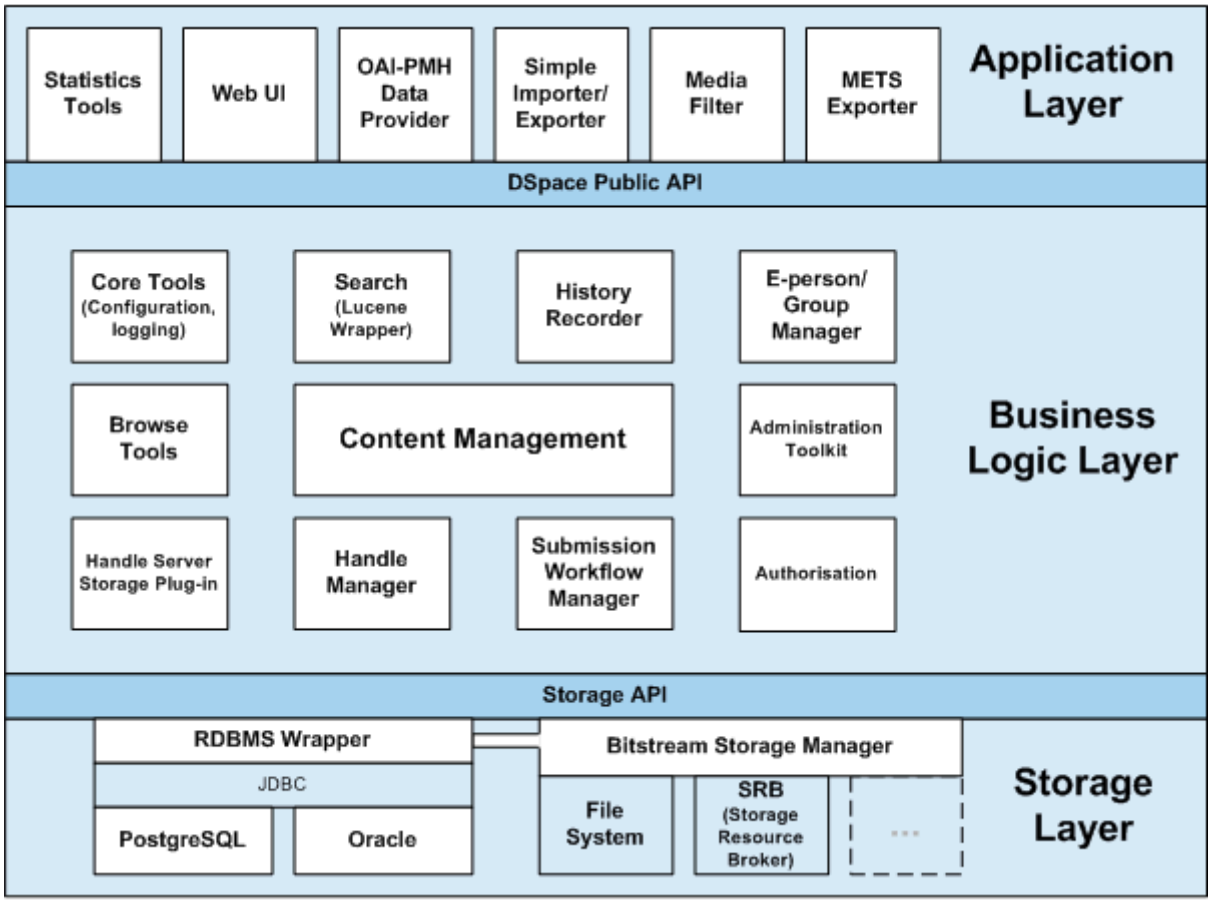


Рисунок 2.1. Архітектура DSpace.

Application Layer (рівень додатків) - включає всі зовнішні/публічні інтерфейси. До них відносяться веб-інтерфейс користувача, REST API, OAI-PMH, RDF та інтерфейси SWORD (v1 і v2). Також включає інтерфейс командного рядка і різні інструменти, які можна використовувати для імпорту/експорту даних DSpace.

Business Logic Layer (рівень бізнес-логіки) - це насамперед рівень Java API ([dspace-source]/dspace-api та dspace-services), який забезпечує основну бізнес-логіку для всіх різних інтерфейсів додатків.

Storage Layer (рівень зберігання) - підмножина dspace-api (класи org.dspace.storage.*), роль якої полягає в управлінні всім вмістом сховища (метаданими, зв'язками, бітовими потоками) для всіх об'єктів бізнес-рівня. Цей рівень забезпечує доступ до реляційної бази даних (зазвичай Postgres або Oracle) через Hibernate ORM та використання FlywayDB для міграції/оновлення. Він також визначає власний BitStoreService для зберігання файлів (бітпотоків) за допомогою плагінів зберігання (наразі підтримується зберігання у файловій системі або Amazon S3).

Storage Layer відповідає за фізичне зберігання метаданих і контенту. Рівень бізнес-логіки займається управлінням контентом архіву, користувачами архіву, авторизацією та робочим процесом. Прикладний рівень містить компоненти, які взаємодіють зі світом поза межами окремої інсталяції DSpace, наприклад, веб-інтерфейс користувача і протокол Ініціативи відкритих архівів для сервісу збору метаданих.

Кожен рівень викликає лише рівень, що знаходиться під ним; наприклад, прикладний рівень не може безпосередньо використовувати рівень сховища. Кожен компонент на рівнях зберігання та бізнес-логіки має визначений загальнодоступний API. Об'єднання API цих компонентів називається Storage API (у випадку рівня зберігання) і DSpace Java API (у випадку рівня бізнес-логіки), а також DSpace REST API (у випадку прикладного рівня). У випадку прикладного рівня варто зазначити, що веб-інтерфейс користувача отримує доступ до DSpace лише через REST API.

Важливо зазначити, що кожний рівень є довіреним. Хоча логіка авторизації знаходиться на рівні бізнес-логіки, система покладається на окремі додатки на рівні додатків для правильної та безпечної аутентифікації. Якби "ворожій" або незахищеній програмі було дозволено безпосередньо викликати Java API, вона могла б дуже легко виконувати дії як будь-яка в системі.

Причиною такого вибору дизайну є те, що методи аутентифікації будуть сильно відрізнятися між різними додатками, тому має сенс залишити логіку і відповідальність за це в цих додатках.

Вихідний код організовано таким чином, щоб дуже строго відповідати цій трирівневій архітектурі.

API рівня сховища та бізнес-логіки детально задокументовані з коментарями у стилі Javadoc. Згенеруйте їх HTML-версію, увійшовши до каталогу [dspace-source]/dspace і запустивши програму:

```
mvn javadoc:javadoc
```

Результуючу документацію буде розміщено за адресою [dspace-source]dspace-api/target/site/apidocs/index.html. Документація на рівні пакунків кожного пакунка зазвичай містить огляд пакунка та деякі приклади використання

2.2.1. Application Layer

Рівень додатків складається з наступних компонентів

- Web User Interface
- REST API
- OAI-PMH Data Provider
- RDF / Linked Data Provider
- SWORD v1 Service / Server
- SWORD v2 Service / Server
- DSpace Command Line Launcher

Веб інтерфейс (Web User Interface) DSpace є найбільшим і найбільш використовуваним компонентом прикладного рівня. Починаючи з DSpace 7, він був перебудований на Angular.io, який взаємодіє через REST API з іншими компонентами DSpace.

Код веб-інтерфейсу користувача знаходиться в окремому проекті GitHub:

https://github.com/DSpace/dspace-angular/

REST API - компонент визначає основний публічний API прикладного рівня

REST API для DSpace проронується як частина "серверного" веб-додатку ([dspace-source]/dspace-server-webapp/). Він доступний в підпункті `/api/` цього веб-додатку (${dspace.server.url}/api/), хоча зручний інтерфейс з можливістю перегляду/пошуку (за допомогою HAL Browser) також доступний за відповідним шляхом - (${dspace.server.url}).

На даний момент REST API відповідає тільки у форматі JSON.

Інтерфейс OAI-PMH може використовуватися іншими системами для збору записів метаданих з вашого DSpace.

OAI-PMH Server Activation

DSpace's OAI-PMH за замовчуванням увімкнено. Однак ви можете ввімкнути/вимкнути його у вашому local.cfg за допомогою цих налаштувань:

```
# Enable (true) or disable (false) OAI-PMH server
oai.enabled = true

# When enabled, OAI-PMH server is available at this path
oai.path = oai
```

Якщо ви змінюєте будь-яку з цих конфігурацій, ви повинні перезапустити ваш контейнер сервлетів (зазвичай Tomcat).

Ви можете перевірити, чи все працює, надіславши запит на адресу [dspace.server.url]/[oai.path]/request?verb=Identify (наприклад, http://localhost:8080/server/oai/request?verb=Identify)

Відповідь має бути схожою на відповідь демонстраційного сервера DSpace 7: https://api7.dspace.org/server/oai/request?verb=Identify.

Якщо ви використовуєте сучасний браузер, ви побачите HTML-сторінку з описом вашого сховища. Те, що ви отримуєте від сервера, насправді є XML-файлом з посиланням на таблицю стилів XSLT, яка відображає цей HTML у вашому браузері (на стороні клієнта). Будь-який браузер, який не може інтерпретувати XSLT, відобразить чистий XML. Таблиця стилів за замовчуванням знаходиться в [dspace-source]/dspace-oai/src/main/resources/static/style.xsl, і її можна змінити, налаштувавши атрибут stylesheet елемента Configuration в [dspace]/config/crosswalks/oai/xoai.xml.

RDF / Linked Data Provider

Обмін вмістом сховища

Більшість сайтів в Інтернеті орієнтовані на використання людиною. Хоча HTML може бути зручним форматом для представлення інформації для людей, він не є зручним форматом для експорту даних у спосіб, зручний для роботи з ними на комп'ютері. Як і більшість програм для створення сховищ, DSpace підтримує OAI-PMH як інтерфейс для представлення збережених метаданих. Хоча OAI-PMH добре відомий у сфері репозитаріїв, він рідко зустрічається в інших сферах (наприклад, Google припинив підтримку OAI-PMH у 2008 році). Семантична павутина - це загальний підхід до публікації даних в Інтернеті разом з інформацією про їхню семантику. Його застосування не обмежується репозиторіями чи бібліотеками, і він має зростаючу базу користувачів. RDF і SPARQL - це стандарти, розроблені W3C для публікації структурованих даних в Інтернеті в машинозчитуваному вигляді. Дані, що зберігаються в репозиторіях, особливо добре підходять для використання в Семантичній павутині, оскільки метадані вже доступні. Їх не потрібно генерувати або вводити вручну для публікації у вигляді пов'язаних даних. Для більшості репозитаріїв, принаймні для репозитаріїв з відкритим доступом, дуже важливо

ділитися своїм вмістом. Пов'язані дані - це досить великий шанс для репозитаріїв представити свій контент таким чином, щоб до нього можна було легко отримати доступ і (повторно) використати.

Починаючи з DSpace 5.0, DSpace надає підтримку публікації збереженого вмісту у вигляді пов'язаних (відкритих) даних.

Щоб опублікувати контент, що зберігається в DSpace, як пов'язані (відкриті) дані, дані мають бути перетворені в RDF. Перетворення в RDF має сконфігуровано, оскільки різні екземпляри DSpace можуть використовувати різні схеми метаданих, різні постійні ідентифікатори (DOI, Handle, ...) і так далі. Залежно від контенту, який потрібно конвертувати, конфігурації та інших параметрів, конвертація може зайняти багато часу і вплинути на продуктивність. Вміст репозитаріїв набагато частіше читають, ніж створюють, видаляють або змінюють, оскільки основною метою репозитаріїв є безпечне зберігання їхнього вмісту. З цієї причини вміст, що зберігається в DSpace, конвертується і зберігається в потрійному сховищі одразу після його створення або оновлення. Потрійне сховище слугує кешем і надає кінцеву точку SPARQL, щоб зробити перетворені дані доступними за допомогою SPARQL. Перетворення запускається автоматично системою подій DSpace і може бути запущено вручну за допомогою інтерфейсу командного рядка. Немає необхідності створювати резервну копію потрійного сховища, оскільки всі дані, що зберігаються в потрійному сховищі, можуть бути відтворені з вмісту, що зберігається деінде в DSpace (в сховищі(ах) активів і в базі даних). Окрім кінцевої точки SPARQL, дані також слід публікувати у вигляді серіалізації RDF. З dspace-rdf DSpace пропонує модуль, який завантажує перетворені дані з потрійного сховища і надає їх у вигляді RDF-серіалізації. Наразі він підтримує RDF/XML, Turtle і N-Triples.

Репозитарії використовують постійні ідентифікатори, щоб зробити контент цитованим та адресувати його. Дотримуючись Принципів зв'язаних даних, DSpace використовує постійний ідентифікатор у вигляді HTTP(S) URI, перетворюючи дескриптор на http://hdl.handle.net/<handle>, а DOI на http://dx.doi.org/<doi>. Загалом, підтримка DSpace Linked Data охоплює всі три

рівні: рівень зберігання з потрійним сховищем, бізнес-логіку з класами для перетворення збереженого вмісту в RDF і прикладний рівень з модулем для публікації RDF-серіалізацій. Подібно до того, як DSpace дозволяє вибрати Oracle або Postgresql в якості реляційної бази даних, ви можете вибирати між різними потрійними сховищами. Єдиною вимогою є те, що потрійне сховище повинно підтримувати мову запитів SPARQL 1.1 та протокол HTTP SPARQL 1.1 Graph Store, які DSpace використовує для зберігання, оновлення, видалення та завантаження перетворених даних в/з потрійного сховища, а також використовує потрійне сховище для надання даних через кінцеву точку SPARQL.

Пакет org.dspace.rdf.conversion містить класи, що використовуються для перетворення вмісту сховища у формат RDF. Саме перетворення виконується за допомогою плагінів. Інтерфейс org.dspace.rdf. conversion. ConverterPlugin дуже простий, за допомогою якого можна розширити можливості перетворення. Єдине, що важливо - плагіни повинні створювати тільки RDF, який можна зробити загальнодоступним, оскільки потрійне сховище надає його за допомогою кінцевої точки sparql, на яку не поширюються обмеження доступу DSpace. Плагіни, що перетворюють метадані, повинні перевіряти, чи потрібно захищати певне поле метаданих (див. org.dspace.app.util.MetadataExposure про те, як це перевірити). Плагін MetadataConverterPlugin має широкі можливості налаштування (див. нижче) і використовується для перетворення метаданих елементів. Плагін StaticDSOConverterPlugin можна використовувати для додавання статичних RDF. Плагін SimpleDSORelationsConverterPlugin створює зв'язки між елементами та колекціями, колекціями та спільнотами, підспільнотами та їхніми батьками, а також між спільнотами верхнього рівня та інформацією, що представляє саме сховище.

Оскільки різні репозитарії використовують різні постійні ідентифікатори для звернення до свого вмісту, можуть бути реалізовані різні алгоритми створення URI, що використовуються в перетворених даних. Наразі можна використовувати HTTP(S) URI репозитарію (так звані локальні URI),

дескриптори та DOI. Якщо ви хочете додати інший алгоритм, зверніться до інтерфейсу org.dspace.rdf.storage.URIGenerator.

SWORD v1 Service / Server

SWORD (Simple Web-service Offering Repository Deposit) - це протокол, що дозволяє клієнтам і серверам спілкуватися навколо складних цифрових об'єктів, особливо щодо підтримки зберігання цих об'єктів у такому сервісі, як цифровий репозитарій. Складні цифрові об'єкти складаються як з метаданих, так і з вмісту файлів, причому файли можуть бути різних форматів, їх може бути багато, а деякі з них можуть бути дуже великими. Протокол визначає семантику для створення, додавання, заміни, видалення та пошуку інформації про ці складні ресурси. Він також дозволяє серверам обмінюватися інформацією про стан обробки доданого контенту, наприклад, розкриваючи інформацію про робочий процес прийому.

Перша основна версія SWORD [SWORD 1.3] була побудована на аспектах створення ресурсів у AtomPub [AtomPub], щоб надати можливість додавання пакунків на сервері за принципом "fire-and-forget".

Такий підхід, коли вкладник не має подальшої взаємодії з сервером, має значну цінність у певних випадках використання, але є й інші, де цього недостатньо. Уявімо, наприклад, що депонент бажає створити цифровий артефакт файл за файлом протягом певного періоду часу, перш ніж вирішити, що настав час його архівувати. У цих випадках потрібен вищий рівень інтерактивності між системами і саме для виконання цієї ролі згодом було розроблено SWORD 2.0 [SWORD 2.0]

DSpace реалізує протокол SWORD через веб-додаток "sword". Наразі DSpace підтримує версію SWORD v1 - 1.3. Специфікацію та додаткову інформацію можна знайти на сайті http://swordapp.org.

DSpace Command Line Launcher

Засіб запуску команд DSpace або інтерфейс CLI дозволяє виконувати різні операції з обслуговування.

Інтерфейс CLI знаходиться за адресою [dspace]/bin/dspace. Запустіть його без аргументів або з опцією -h, щоб побачити всі доступні операції. Виконайте dspace op -h, щоб переглянути подробиці про операцію.

bin/dspace -h
bin/dspace cleanup -h
bin/dspace cleanup
bin/dspace cleanup –verbose

2.2.2. Storage Layer

DSpace використовує реляційну базу даних для зберігання всієї інформації про організацію контенту, метаданих про контент, інформації про користувачів та авторизацію, а також про стан поточних робочих процесів.

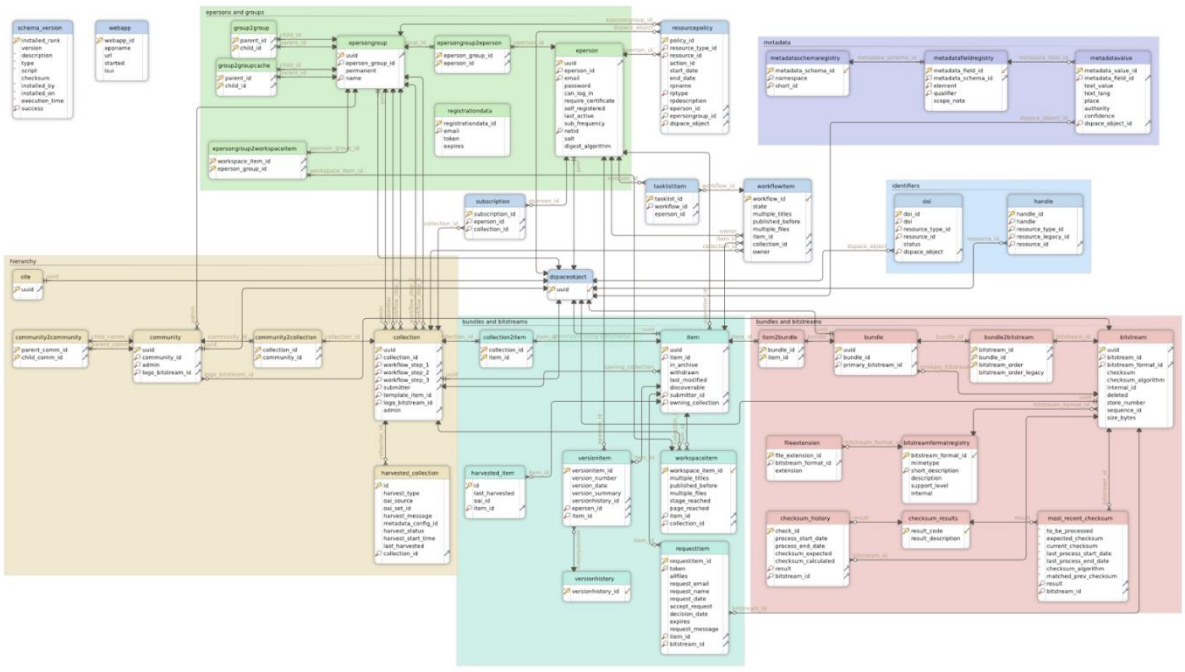


Рисунок. 2.2. Схема бази даних

DSpace використовує FlywayDB для автоматичної ініціалізації та оновлення баз даних. Роль Flyway полягає в ініціалізації таблиць бази даних перед ініціалізацією Hibernate.

Клас org.dspace.storage.rdbms.DatabaseUtils керує всіма викликами API Flyway і виконує міграції SQL у пакеті org.dspace.storage. rdbms.sqlmigration та міграції Java у пакеті org.dspace.storage. rdbms.migration.

Після запуску всіх міграцій баз даних запускається серія зворотних викликів Flyway для ініціалізації бази даних з необхідним вмістом за замовчуванням. Наприклад, існують зворотні виклики для додавання стандартних груп DSpace (GroupServiceInitializer), стандартних реєстрів метаданих і форматів (DatabaseRegistryUpdater) і стандартного об'єкта Site (SiteServiceInitializer). Усі функції зворотного виклику знаходяться у пакунку org.dspace.storage.rdbms.

Хоча Flyway автоматично ініціалізується і виконується під час запуску, різні утиліти для роботи з базами даних також доступні для запуску в командному рядку. Ці утиліти дозволяють вручну запускати оновлення бази даних або перевіряти стан бази даних.

DSpace використовує Hibernate ORM як об'єктно-реляційний рівень відображення між базою даних DSpace та кодом DSpace.

Основну конфігурацію Hibernate можна знайти у [dspace]/config/hibernate.cfg.xml

Ініціалізація Hibernate запускається за допомогою Spring, визначених у [dspace]/config/spring/api/core-hibernate.xml. Ця конфігурація Spring підтягує деякі налаштування з конфігурації DSpace, а саме всі налаштування бази даних (db.*), визначені там.

Всі об'єктні класи DSpace надають клас реалізації DAO (Data Access Object), який розширює інтерфейс GenericDAO, визначений у класі org.dspace.core.GenericDAO. Реалізація за замовчуванням (абстрактна) знаходиться у класі org.dspace.core.Abstract Hibernate DAO.

Об'єкт DSpace Context (org.dspace.core.Context) надає доступ до налаштованого org.dspace.core.DBConnection (з'єднання з базою даних), яке за

замовчуванням є Hibernate DB Connection. Клас org.dspace.core.Hibernate DB Connection надає доступ до інтерфейсу Hibernate Session (org.hibernate.Session) та його транзакцій.

Кожен сеанс Hibernate відкриває одне з'єднання з базою даних, коли він створюється, і утримує його, поки сеанс не буде закрито. Сеанс може складатися з однієї або декількох транзакцій. Сесії не потоко безпечні (тобто окремі об'єкти не можуть бути спільними для різних потоків).

Hibernate інтелектуально кешує об'єкти у поточному сеансі Hibernate (при доступі до об'єктів), що дозволяє оптимізувати продуктивність.

DSpace надає методи на об'єкті Context для спеціального видалення (Context.uncacheEntity()) або перезавантаження (Context.reloadEntity()) об'єктів у кеші сеансу Hibernate.

DSpace також надає спеціальні "режими" об'єктів Context для оптимізації продуктивності Hibernate для доступу тільки для читання (Mode.READ_ONLY) або пакетної обробки (Mode.BATCH_EDIT). Ці режими можна вказати при створенні нового контекстного об'єкта.

Більшість функцій, які використовує DSpace, може запропонувати будь-яка стандартна база даних SQL, що підтримує транзакції. Однак, наразі DSpace надає скрипти міграції Flyway лише для PostgreSQL та Oracle. Використання додаткових серверів баз даних теоретично є можливим, але мінімально вимагатиме створення спеціальних сценаріїв міграції Flyway для цих серверів баз даних.

2.3. Пошукова система Solr

Solr - це пошукова платформа з відкритим вихідним кодом, яка використовується для створення пошукових додатків. Він був побудований на основі програмного продукту Lucene (повнотекстова пошукова система). Solr готовий до роботи, швидкий і масштабований. Додатки, створені з використанням Solr, є складними та забезпечують високу продуктивність.

Саме Йонік Сілі створив Solr у 2004 році, щоб додати можливості пошуку на сайті компанії. У 2006 році було зроблено проект із відкритим вихідним

кодом у рамках Apache Software Foundation. Його останню версію, було випущено 2016 року з підтримкою виконання паралельних SQL-запитів.

Solr можна використовувати разом із Hadoop. Оскільки Hadoop обробляє великий обсяг даних, Solr допомагає знайти необхідну інформацію з великого джерела інформації. Solr може використовуватися не тільки для пошуку, а й для зберігання. Як і інші бази даних NoSQL, це нереляційна технологія зберігання та обробки даних.

2.3.1. Особливості Apache Solr

Solr - це оболонка навколо Java API від Lucene. Тому, використовуючи Solr, ви можете використовувати всі можливості Lucene. Деякі з найбільш важливих особливостей Solr:

Restful APIs - Щоб обмінюватися інформацією з Solr, не обов'язково мати навички програмування на Java. Замість цього ви можете використовувати API функції для взаємодії з ним. Ми вводимо дані в Solr у таких форматах, як XML, JSON і .CSV, і отримуємо результати в тих самих форматах.

Повнотекстовий пошук - Solr надає всі можливості, необхідні для повнотекстового пошуку, такі як токени, фрази, перевірка орфографії, підставні знаки та автозаповнення.

Готовність для підприємства - залежно від потреб організації Solr може бути розгорнутий у будь-яких системах (великих або малих), таких як автономні, розподілені, хмарні тощо.

Гнучкість і розширюваність. Розширюючи класи Java і налаштовуючи їх відповідним чином, ми можемо легко налаштовувати компоненти Solr.

База даних NoSQL - Solr також може використовуватися як база даних NOSQL з великими обсягами даних, де ми можемо розподіляти завдання пошуку по кластеру.

Інтерфейс адміністратора - Solr надає простий у використанні, зручний, функціональний інтерфейс користувача, за допомогою якого ми можемо виконувати всі можливі завдання, як-от керування журналами, додавання, видалення, оновлення та пошук документів.

Висока масштабованість. Використовуючи Solr з Hadoop, ми можемо масштабувати його вмістимість, додаючи репліки.

Текст-орієнтований і відсортований за релевантністю - Solr здебільшого використовується для пошуку текстових документів, а результати надаються відповідно до запиту користувача.

Apache Solr – основні команди

Після встановлення Solr перейдіть у папку bin у домашньому каталозі Solr і запустіть Solr, використовуючи таку команду.

[Hadoop@localhost ~]$ cd

[Hadoop@localhost ~]$ cd Solr/

[Hadoop@localhost Solr]$ cd bin/

[Hadoop@localhost bin]$ ./Solr start

Ця команда запускає Solr у фоновому режимі, прослуховуючи порт 8983, відображаючи таке повідомлення.

Waiting up to 30 seconds to see Solr running on port 8983 [\]

Started Solr server on port 8983 (pid = 6035). Happy searching!

Якщо ви запустите Solr за допомогою команди запуску , Solr запуститься у фоновому режимі. Замість цього ви можете запустити Solr на передньому плані, використовуючи опцію -f .

Використовуючи опцію -p команди start , ми можемо запустити Solr на іншому порту, як показано в наступному блоці коду.

[Hadoop@localhost bin]$ ./Solr start -p 8984

Waiting up to 30 seconds to see Solr running on port 8984 [-]

Ви можете зупинити Solr, використовуючи команду stop.

$ ./Solr stop

2.3.2. Архітектура Apache Solr

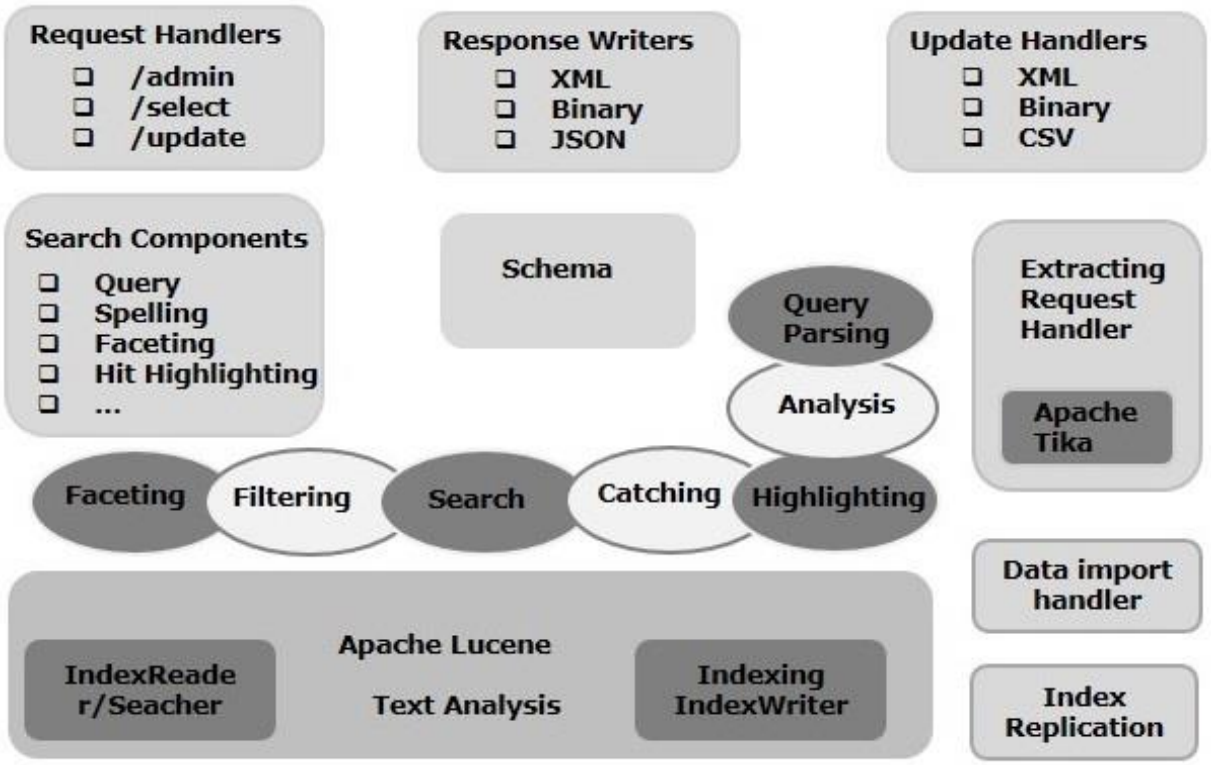


Рис. 2.3. Блок схема архітектури Apache Solr

Нижче наведено основні компоненти Apache Solr:

Обробник запитів - запити, які ми відправляємо в Apache Solr, обробляються цими обробниками запитів. Запити можуть бути запитами запиту або запитами на оновлення індексу. Виходячи з нашої вимоги, нам потрібно вибрати обробник запиту. Щоб передати запит до Solr, ми зазвичай зіставляємо обробник з певною кінцевою точкою URI, і вказаний запит буде обслуговуватися ним.

Компонент пошуку - це тип (функція) пошуку, що надається в Apache Solr. Це може бути перевірка орфографії, запит, огранювання, виділення збігів тощо. Ці компоненти пошуку зареєстровані як обробники пошуку. Кілька компонентів можуть бути зареєстровані в обробнику пошуку.

Парсер запитів - Парсер запитів Apache Solr аналізує запити, які ми передаємо Solr, і перевіряє запити на наявність синтаксичних помилок. Після аналізу запитів він переводить їх у формат, зрозумілий Lucene.

Засіб запису відповідей - Засіб запису відповідей в Apache Solr - це компонент, який генерує форматований висновок для користувацьких запитів. Solr підтримує формати відповідей, такі як XML, JSON, CSV тощо.

Аналізатор / токенізатор - Lucene розпізнає дані у вигляді токенів. Apache Solr аналізує вміст, розділяє його на токени і передає ці токени в Lucene. Аналізатор в Apache Solr аналізує текст полів і генерує потік токенів. Токенізатор розбиває потік токенів, підготовлений аналізатором, на токени.

2.4. HAL та HAL-браузер

Гіпертекстова мова додатків SON, або HAL, - це простий формат, який забезпечує послідовний і простий спосіб створення гіперпосилань між ресурсами в нашому API. Включення HAL в наш REST API робить його набагато зручнішим для користувачів, а також дозволяє самодокументувати його.

Він повертає дані у форматі JSON, які містять відповідну інформацію про API.

Перш за все, якщо ви використовуєте Spring Data REST для надання доступу до вашого репозиторію через API, включення та використання HAL-браузера є дуже простим кроком. Єдине, що вам потрібно зробити, це додати наступну залежність до вашого файлу збірки Gradle.

Після цього ви можете відкрити HAL-браузер за адресою http://<host>:<port>/browser/index.html і перейти до API. Більше того, якщо ви використовуєте curle, HAL-браузер також включає легкодоступні посилання на вашу документацію та інтегрує її в браузер.

API, які використовують HAL, можна легко обслуговувати та використовувати за допомогою бібліотек з відкритим кодом, доступних для більшості основних мов програмування. Він також досить простий, тому ви можете працювати з ним так само, як і з будь-яким іншим JSON.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Docker Compose

Compose - це інструмент для визначення та запуску багато контейнерних Docker-додатків. За допомогою Compose ви використовуєте YAML-файл для налаштування сервісів вашого додатку. Потім, за допомогою однієї команди, ви створюєте і запускаєте всі сервіси з вашої конфігурації.

3.1.1. Основні функції та варіанти використання Docker Compose

Compose використовує назву проекту для ізоляції середовищ одне від одного. Ви можете використовувати цю назву проекту у кількох різних контекстах:

на хості розробника, щоб створити кілька копій одного середовища, наприклад, коли ви хочете запустити стабільну копію для кожної функціональної гілки проекту;

на CI-сервері, щоб збірки не заважали одна одній, ви можете встановити назву проекту як унікальний номер збірки;

на спільному хості або хості розробника, щоб різні проекти, які можуть використовувати однакові імена служб, не заважали один одному.

Назвою проекту за замовчуванням є базове ім'я каталогу проекту. Ви можете задати власну назву проекту за допомогою параметра командного рядка -p або змінної середовища COMPOSE_PROJECT_NAME.

За замовчуванням директорія проекту є базовою директорією файлу Compose. Користувацьке значення для нього можна вказати за допомогою параметра командного рядка --project-directory.

Compose зберігає всі томи, які використовуються вашими службами. Під час повторного запуску докера, якщо він знаходить контейнери з попередніх запусків, він копіює томи зі старого контейнера до нового контейнера. Цей процес гарантує, що всі дані, які ви створили у томах, не буде втрачено.

Compose кешує конфігурацію, використану для створення контейнера. Коли ви перезапустите службу, яка не зазнала змін, Compose повторно використовує існуючі контейнери. Повторне використання контейнерів означає, що ви можете вносити зміни до вашого середовища дуже швидко.

Compose можна використовувати в різних ситуаціях.

Середовища розробки

Коли ви розробляєте програмне забезпечення, можливість запускати програму в ізольованому середовищі та взаємодіяти з нею має вирішальне значення. Інструмент командного рядка Compose можна використовувати для створення середовища і взаємодії з ним.

Файл Compose надає можливість документувати і налаштовувати всі сервісні залежності програми (бази даних, черги, кеш, API веб-сервісів тощо). За допомогою інструменту командного рядка Compose ви можете створити і запустити один або декілька контейнерів для кожної залежності за допомогою однієї команди (docker compose up).

Разом ці можливості надають розробникам зручний спосіб розпочати роботу над проектом. Compose може скоротити багатосторінковий "посібник для початківців" до одного машинозчитуваного Compose-файлу та кількох команд.

Важливою частиною будь-якого процесу безперервного розгортання або безперервної інтеграції є автоматизований набір тестів. Автоматизоване наскрізне тестування вимагає середовища для виконання тестів. Compose надає зручний спосіб створення і знищення ізольованих тестових середовищ для вашого набору тестів. Визначивши повне середовище у файлі Compose, ви можете створювати і знищувати ці середовища лише за допомогою кількох команд.

docker compose up -d

./run_tests

docker compose down

3.1.2. Особливості налаштування Docker Compose

Робота Compose базується на конфігураційному файлі docker-compose.yml. У ньому ми визначаємо всі наші сервіси. Уявіть собі сервіс як частину вашого додатку, наприклад, базу даних або API. Всі наші сервіси покладаються на образ, за допомогою якого ми створюємо контейнер. Запуск контейнера може мати багато опцій; те, як ці опції налаштовані, буде збережено в yml-файл.

Ще однією перевагою наявності всіх наших сервісів з усіма відповідними параметрами збірки в одному файлі є те, що ми можемо збирати і запускати всі наші сервіси одночасно, викликаючи docker-compose up!

Використання змінних середовища

Запуск наших контейнерів можна зробити гнучкішим за допомогою змінних оточення. Ми можемо надати команді docker-compose up файл, який містить деякі змінні оточення. Таким чином, можна безпечно надавати паролі без їх запису їх у файлі конфігурації.

Приклад: ми створимо файл з назвою .env з DBPASSWORD=secretpass. Потім ми можемо використовувати DBPASSWORD як змінну у файлі docker-compose.yml.

Для цього ми вказуємо наш env-файл у виклику docker-compose: docker-compose --env-file .env up. Це убезпечить наші паролі від потрапляння до наших репозиторіїв та docker-compose.yml. Також це забезпечує більшу гнучкість і охайність проекту, оскільки наші паролі більше не є жорстко закодованими.

Спільна мережа контейнерів

Коли ми створюємо наші контейнери за допомогою Compose, він визначає мережу, до якої всі наші сервіси мають спільний доступ. Це означає, що всі сервіси можуть спілкуватися між собою.

Наприклад, ми розмістили наш додаток на test.com. Коли нашому API потрібно зв'язатися з нашою базою даних, йому не потрібно підключатися через test.com:5432, а замість цього він може просто доступитися до бази даних через внутрішню мережу. Це дуже важливо для безпеки, тому що це означає, що

тільки наші сервіси мають доступ до нашої бази даних, клієнти не можуть отримати доступ до сервісу ззовні.

Переносимість та контроль версій

Оскільки ми зберігаємо всю нашу конфігурацію в одному файлі, ми можемо легко поділитися цим файлом через систему контролю версій, наприклад таку як Git. Користувачі можуть просто витягнути docker-compose.yml і вихідний код, і мати змогу запустити всі контейнери. Додатковою перевагою є те, що ми зберігаємо всю історію змін, які ми вносимо в нашу конфігурацію, так що ми можемо при необхідності відновити попередню версію. Це також може полегшити налаштування конвеєра CI/CD для цього додатка.

Гнучкість

Оскільки всі наші сервіси повністю ізольовані один від одного, ми можемо легко додати новий сервіс. Можливо, в майбутньому нашому додатку знадобиться кешування → просто створіть контейнер Redis! Інші сервіси, такі як наш API, можуть легко підключатися до нового сервісу Redis через внутрішню мережу.

Pulling an Image.

Іноді Image, необхідний для нашого сервісу, вже опубліковано (нами або іншими) на Docker Hub або в іншому реєстрі Docker.

У такому випадку ми посилаємося на нього за допомогою атрибута image, вказуючи назву image та тег:

```
services:
  my-service:
    image: ubuntu:latest
    ...
```

Побудова іміджу

Крім того, нам може знадобитися зібрати образ з вихідного коду, використавши його Docker-файл.

Цього разу ми скористаємося ключовим словом build, передавши шлях до докер файлу як значення:

```
services:
  my-custom-app:
    build: /path/to/dockerfile/
    ...
```

Ми також можемо використовувати URL замість шляху:

```
services:
  my-custom-app:
    build: https://github.com/my-company/my-project.git
    ...
```

Крім того, ми можемо вказати ім'я зображення у поєднанні з атрибутом build, яке дасть ім'я створеному зображенню, що зробить його доступним для використання іншими сервісами:

```
services:
  my-custom-app:
    build: https://github.com/my-company/my-project.git
    image: my-project-image
    ...
```

Налаштування мережі

Docker контейнери взаємодіють між собою у мережах, створених, неявно або через конфігурацію, за допомогою Docker Compose. Сервіс може взаємодіяти з іншим сервісом у тій самій мережі, просто посилаючись на нього за назвою контейнера та портом (наприклад, network-example-service:80), за умови, що ми зробили порт доступним за допомогою ключового слова expose:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    expose:
      - "80"
```

У цьому випадку він також працюватиме без експонування, оскільки директива expose вже міститься у докер-файлі зображення.

Щоб отримати доступ до контейнера з хоста, порти повинні бути відкриті декларативно за допомогою ключового слова ports, яке також дозволяє нам вибрати, чи ми відкриваємо порт інакшим чином на хості:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    ports:
      - "80:80"
    ...
  my-custom-app:
    image: myapp:latest
    ports:
      - "8080:3000"
    ...
  my-custom-app-replica:
    image: myapp:latest
    ports:
      - "8081:3000"
    ...
```

Порт 80 тепер буде видно з хоста, а порт 3000 двох інших контейнерів буде доступний на портах 8080 і 8081 хоста. Цей механізм дозволяє нам запускати різні контейнери, що відкривають одні й ті ж порти, без колізій.

Тепер, ми можемо визначити додаткові віртуальні мережі для розділення наших контейнерів:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    networks:
```

```
      - my-shared-network
    ...
  another-service-in-the-same-network:
    image: alpine:latest
    networks:
      - my-shared-network
    ...
  another-service-in-its-own-network:
    image: alpine:latest
    networks:
      - my-private-network
    ...
networks:
  my-shared-network: {}
  my-private-network: {}
```

Налаштування томів (Volumes)

Існує три типи томів: анонімні, іменовані та хост.

Docker керує як анонімними, так і іменованими томами, автоматично монтуючи їх у самостійно створені каталоги на хості. Хоча анонімні томи були корисними у старих версіях Docker (до 1.9), зараз рекомендується використовувати іменовані томи. Томи хоста також дозволяють вказати існуючу папку на хості.

Ми можемо налаштувати томи хоста на рівні сервісу, а іменовані томи - на зовнішньому рівні конфігурації, щоб зробити останні видимими для інших контейнерів, а не лише для того, до якого вони належать:

У цьому випадку обидва контейнери матимуть доступ на читання/запис до спільної теки my-named-global-volume, незалежно від того, до якого шляху вони її прив'язали. Натомість, обидва томи хоста будуть доступні лише для volumes-example-service.

Теку /tmp файлової системи хоста буде зіставлено з текою /my-volumes/host-volume контейнера. Ця частина файлової системи доступна для запису, що означає, що контейнер може читати, а також записувати (і видаляти) файли на хост-машині.

Ми можемо змонтувати том у режимі лише для читання, додавши до правила :ro, як для теки /home (ми не хочемо, щоб контейнер Docker помилково стер наших користувачів).

Оголошення залежностей

Часто нам потрібно створити ланцюжок залежностей між нашими сервісами, щоб одні сервіси завантажувалися раніше (і вивантажувалися пізніше) за інші. Цього можна досягти за допомогою ключового слова depends_on:

```
services:
  kafka:
    image: wurstmeister/kafka:2.11-0.11.0.3
    depends_on:
      - zookeeper
    ...
  keeper:
    image: wurstmeister/zookeeper
    ...
```

Однак слід пам'ятати, що Compose не чекатиме на завершення завантаження сервісу keeper перед запуском сервісу kafka; він просто чекатиме на його запуск. Якщо нам потрібно, щоб служба була повністю завантажена перед запуском іншої служби, нам потрібно глибше контролювати порядок запуску і завершення роботи у Compose.

Керування змінними середовища

Працювати зі змінними оточення у Compose дуже просто. Ми можемо визначати статичні змінні оточення, а також динамічні змінні за допомогою нотації ${}:

```
services:
  database:
    image: "postgres:${POSTGRES_VERSION}"
    environment:
      DB: mydb
      USER: "${USER}"
```

Існують різні способи надання цих значень для Compose.

Наприклад, один з них - задати їх у файлі .env у тому ж каталозі, структурованому як файл .properties, ключ=значення:

```
POSTGRES_VERSION=alpine
USER=foo
```

В іншому випадку ми можемо встановити їх в операційній системі перед викликом команди:

```
export POSTGRES_VERSION=alpine
export USER=foo
docker-compose up
```

Нарешті, ми можемо легко використовувати простий однорядковий рядок в оболонці shell:

```
POSTGRES_VERSION=alpine USER=foo docker-compose up
```

Ми можемо змішувати підходи, але пам'ятаймо, що Compose використовує наступний порядок пріоритетів, перезаписуючи менш важливі з вищими пріоритетами:

Compose file

Shell environment variables

Environment file

Dockerfile

Variable not defined

Масштабування та реплікації

У старих версіях Compose можна було масштабувати екземпляри контейнера за допомогою команди docker-compose scale. У нових версіях вона застаріла і замінена на опцію --scale.

Ми також можемо використовувати Docker Swarm, кластер Docker Engines, і автоматично масштабувати наші контейнери декларативно через атрибут replicas секції deploy:

```
services:
  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 6
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      ...
```

У розділі deploy ми також можемо вказати багато інших параметрів, наприклад, порогові значення ресурсів. Однак, Compose враховує весь розділ deploy лише під час розгортання до Swarm, і ігнорує його в інших випадках.

Хоча існує безліч опцій і команд, потрібно знати принаймні ті з них, які допоможуть правильно активувати і деактивувати всю систему.

Створювати і запускати контейнери, мережі і томи, визначені в конфігурації можна за допомогою команди:

```
docker-compose up
```

Після першого запуску, ми можемо просто використовувати старт для запуску сервісів:

```
docker-compose start
```

Якщо наш файл має ім'я, відмінне від стандартного (docker-compose.yml), ми можемо використати прапори -f і --file, щоб вказати альтернативне ім'я файлу:

```
docker-compose -f custom-compose-file.yml start
```

Compose також може працювати у фоновому режимі як демон, якщо запустити його з опцією -d:

```
docker-compose up -d
```

Щоб безпечно зупинити активні сервіси, ми можемо використовувати stop, який збереже контейнери, об'єми та мережі разом з усіма змінами, внесеними до них:

```
docker-compose stop
```

Щоб скинути статус нашого проекту, ми можемо просто збити його, що знищить все, за винятком зовнішніх томів:

```
docker-compose down
```

3.1.3. Обслуговування та резервне копіювання

Під час використання PostgreSQL рекомендується регулярно очищати базу даних для оптимізації продуктивності. За замовчуванням PostgreSQL виконує автоматичне очищення від вашого імені. Однак, якщо ви вимкнули цю функцію, рекомендується запускати команди vacuumdb на регулярній основі.

```
#очищення бази даних
40 2 * * * /usr/local/pgsql/bin/vacuumdb --analyze dspace > /dev/null 2>&1
```

Резервну копію бази даних DSpace можна створювати та відновлювати за допомогою звичайних методів резервного копіювання та відновлення PostgreSQL, наприклад, за допомогою pg_dump та psql. Однак при відновленні бази даних потрібно буде виконати наступні додаткові кроки.

Після відновлення резервної копії вам потрібно буде скинути послідовності генерації первинних ключів, щоб вони не створювали вже використані первинні ключі. Зробити це можливо, виконавши SQL в [dspace]/etc/postgres/update-sequences.sql, наприклад, за допомогою:

```
psql -U dspace -f [dspace]/etc/update-sequences.sql
```

ВИСНОВОК

Під час виконання бакалаврської дипломної роботи було ознайомлено з системою інституційних репозиторіїв Dspace 7. Проведено аналіз методів деплою програмоного продукту.

Для деплою Dspace 7 було обрано хмарний сервіс Google Cloud і технлогія конейнеризації Docker, ця платформа надає своїм користувачам сервіси які допомагають клієнтам обчислювати і зберігати дані, а також допомагають розробникам створювати, тестувати і розгортати додатки.

Docker дозволяє розробникам упаковувати програми в контейнери, які є стандартизованими виконуваними компонентами, що поєднують вихідний код програми з бібліотеками та залежностями операційної системи, необхідними для запуску цього коду в будь-якому середовищі. Контейнери спрощують розповсюдження програмних продуктів.

Результатом дипломної роботи є розробка системи контейнеризації і деплою інституційного репозиторію Dspace 7, яка дозволить швидко розгортати програмне забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Google Cloud. Google Cloud Documentation. [Електронний ресурс]. – Режим доступу: https://cloud.google.com/docs (дата звернення: 05.04.2023 р.).
2) DSpace. DSpace 7 Documentation. [Електронний ресурс]. – Режим доступу: https://wiki.lyrasis.org/display/DSDOC7x (дата звернення: 11.03.2023 р.).
3) Poulton N. Docker Deep Dive, Packt Publishing, 2023. – 372 p.
4) Matthias K., Kane S. Docker: Up & Running: Shipping Reliable Containers in Production, O'Reilly Media, 2022. – 394 p.
5) Docker. Docker Compose Documentation. [Електронний ресурс]. – Режим доступу: https://docs.docker.com/compose/ (дата звернення: 06.04.2023 р.).
6) Smith M., Barton M. DSpace: Open Source Digital Repository Software, MIT Libraries, 2019. – 320 p.
7) Fyffe R., Walters T. Institutional Repositories and the Open Access Movement, Chandos Publishing, 2019. – 280 p.
8) Suber P. Open Access, MIT Press, 2019. – 256 p.
9) Giarlo M. Academic Libraries and Digital Repositories, Facet Publishing, 2020. – 310 p.
10) Coulon T., Baker D. Digital Libraries and Institutional Repositories, Springer, 2020. – 340 p.
11) Tennant J., Crane H. The State of Open Access Research, Routledge, 2020. – 298 p.
12) Fielding R. RESTful Web APIs, O'Reilly Media, 2021. – 448 p.
13) Newman S. Building Microservices, O'Reilly Media, 2021. – 620 p.
14) W3C. Linked Data and RDF Standards Documentation. [Електронний ресурс]. – Режим доступу: https://www.w3.org/RDF/ (дата звернення: 15.05.2023 р.).
15) Apache Software Foundation. Apache Lucene Documentation. [Електронний ресурс]. – Режим доступу: https://lucene.apache.org/ (дата звернення: 02.03.2023 р.).

ДОДАТКИ

Архітектура DSpace7

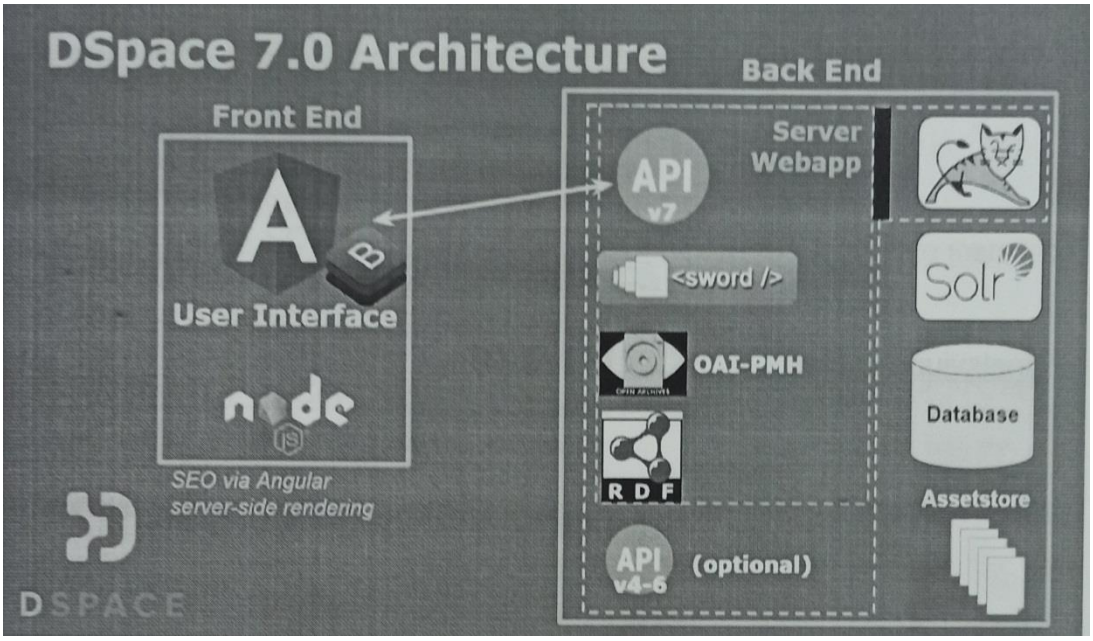
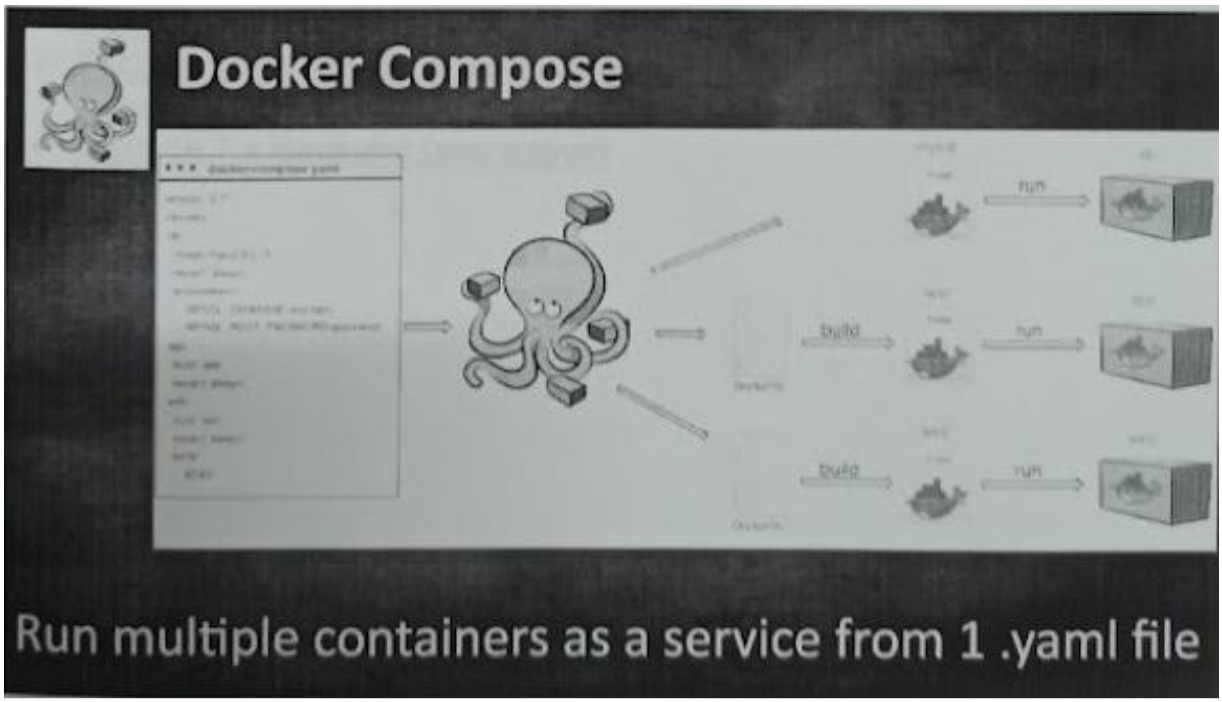


Схема роботи Docker Compose



Вимоги до серверної частини Backend і зовнішнього інтерфейсу

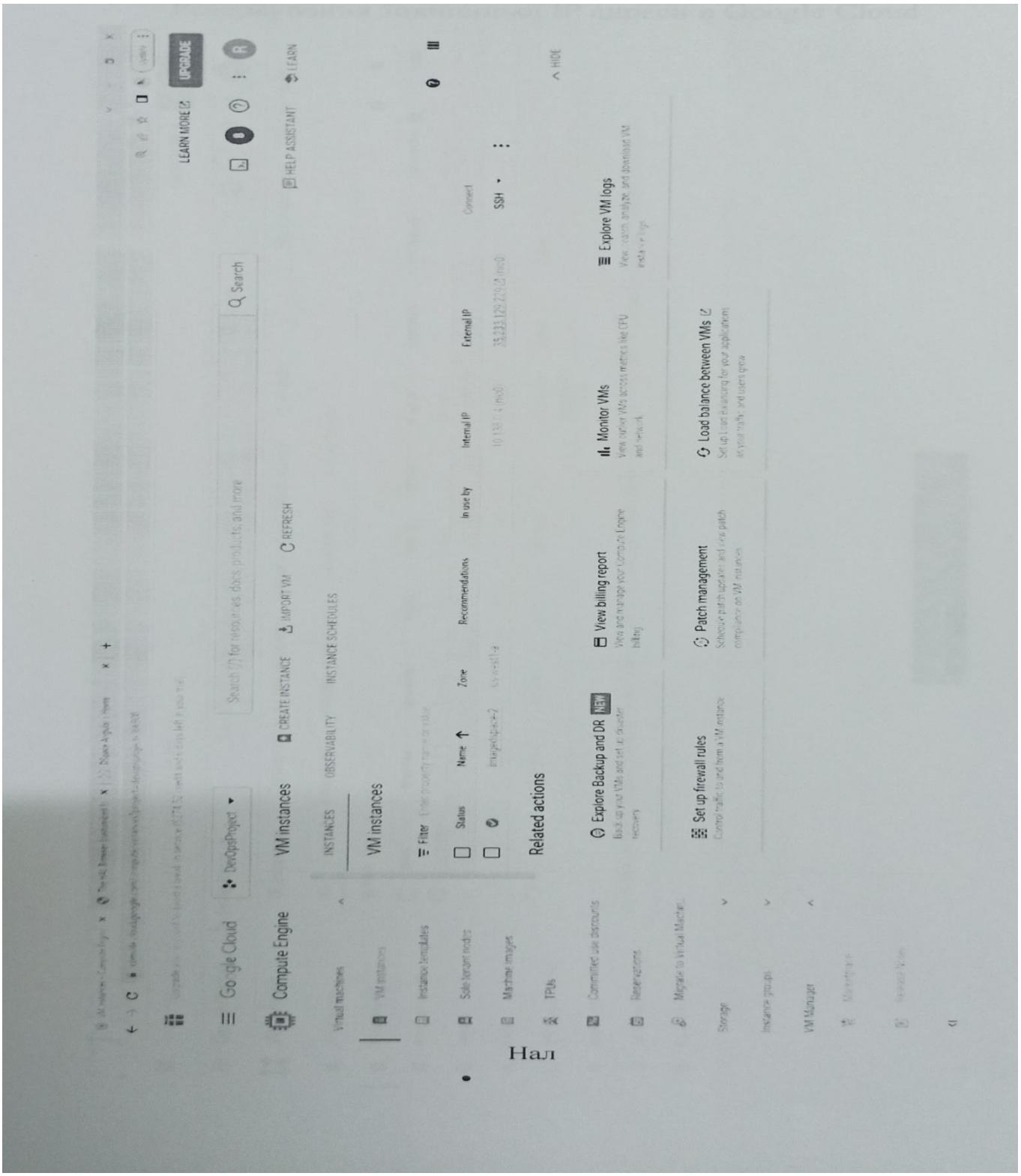
Backend Requirements

- UNIX-like OS or Microsoft Windows
- Java 8 or above (Java Development Kit, JDK)
- Apache Maven 3.3.x or above (Java build tool)
- Apache Ant 1.10.x or later (Java build tool)
- Relational database:
    - PostgreSQL 11.x–12.x, 13.x, 14.x or 15.x (with pgcrypto installed)
    - Oracle 11g/12c or later (commercial license)
- Apache Solr 8.x (full-text indexing/search service)
- Apache Kafka (or equivalent message broker)
- Optional Trip & Map (Mapbox or equivalent)
- Optional: Graph database for location-based statistics
- Git code version control

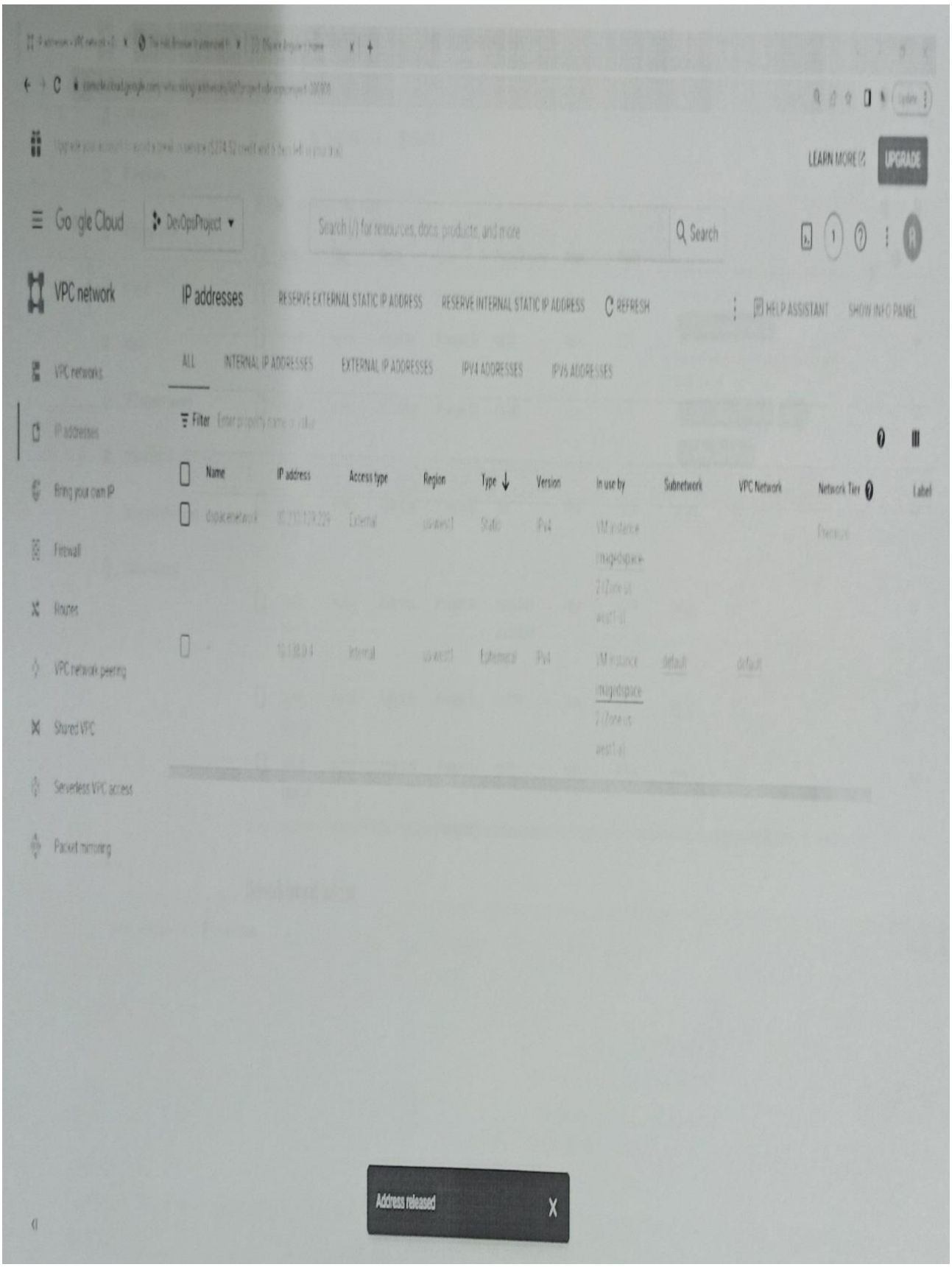
Frontend Requirements

- UNIX-like OS or Microsoft Windows
- Node.js (16.x or v18.x)
- Yarn (v1.x)
- PM2 (Process Manager for Node.js apps) (optional but recommended for production)
- Database (see Backend section above)

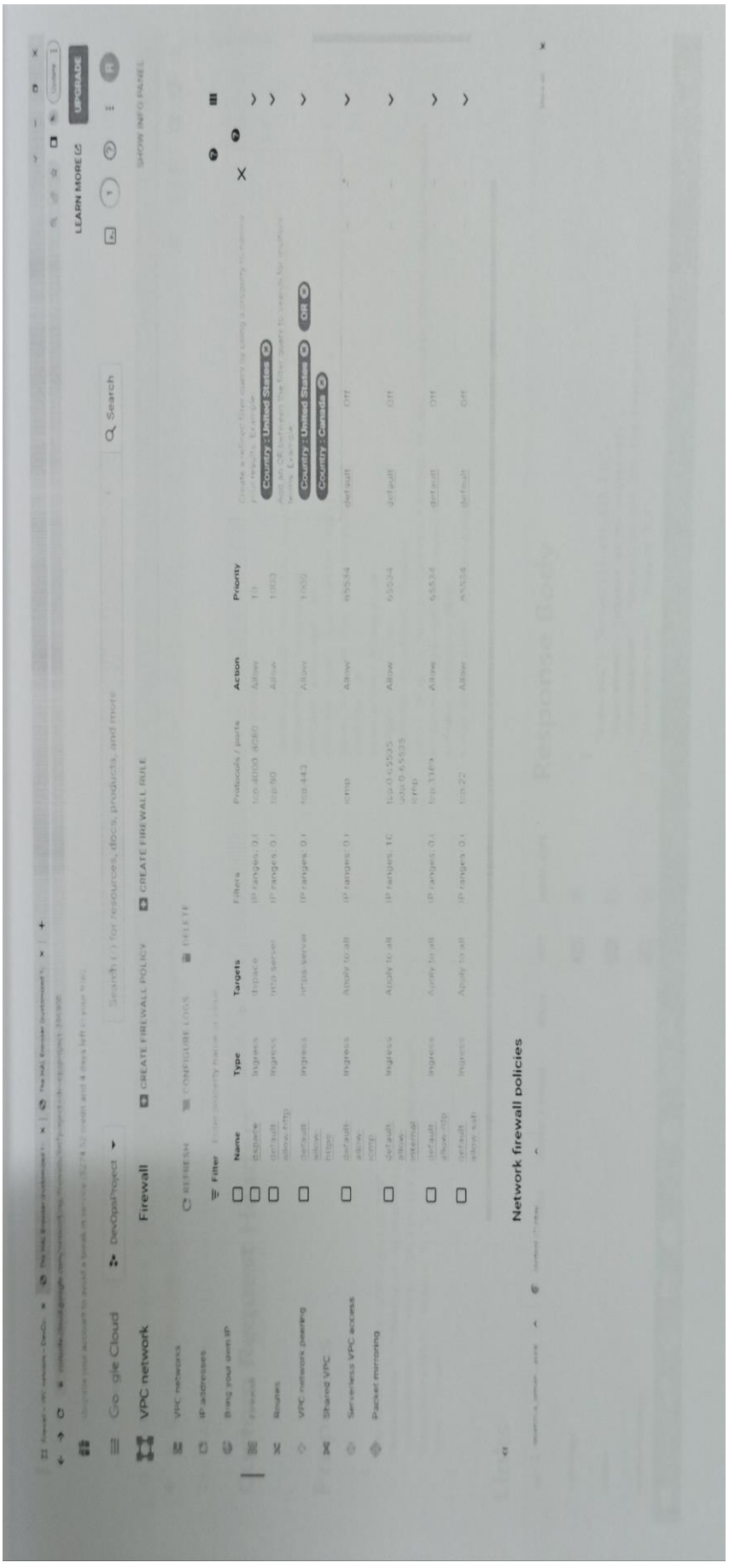
Запуск віртуальної машини в Google Cloud



Резервування зовнішньої IP адреси в Google Cloud



Налаштування firewall правил Google Cloud



Доступ до Rest функцій

Список всіх доступних Rest Endpoints

Список всіх доступних Rest Endpoints