

EU-FarmBook

Deliverable 1.10

System Infrastructure and API

Report | Public



Funded by
the European Union

Summary

Call	HORIZON-CL6-2021-GOVERNANCE-01
Topic	HORIZON-CL6-2021-GOVERNANCE-01-24
Project	EU-FarmBook: Supporting knowledge exchange between all AKIS actors in the European Union
Acronym	EU-FarmBook
Project No.	101060382
Management	Universiteit Gent
Duration	84 Months
Start & end date	01/08/2022-31/07/2029
Deliverable	D1.10. System Infrastructure and API
Type	R (Document/Report)
Dissemination level	PU (Public)
Due Date	31/07/2023 (Original) 30/09/2023 (Revised)
Submission Date	30/09/2023
Work Package No.	WP1
Lead Beneficiary	Maastricht University
Authors	Louis Powell Maastricht University Christopher Brewster Maastricht University Pranav Bapat Maastricht University Xu Wang Maastricht University
Contributors	Hercules Panoutsopoulos Agricultural University of Athens Panagiotis Stamatelopoulos Agricultural University of Athens Jonas De Moor University of Ghent Saskia Vandamme University of Ghent Montse Cuadros VICOMTEC Aitor Garcia VICOMTEC Wouter Vandersyppe LeapForward Group Clio Pype LeapForward Group Wout Vandesompele LeapForward Group Sarah Van Den Berghe LeapForward Group
Version	Version 1

History of Changes

Version 0.1	13/06/2023	Louis Powell	Setup and initial drafting
Version 0.2	01/07/2023	Louis Powell	Further drafting and developments
Version 0.3	03/09/2023	Louis Powell	Drafting of main body and sections
Version 0.4	04/09/2023	Jonas De moor	Addition of section 8 for the UGENT platform
Version 0.5	05/09/2023	Louis Powell	Final changes before 1 st draft version submission for review to Pranav
Version 0.6	05/09/2023	Pranav Bapat	1 st review and changes
Version 0.7	11/09/2023	Christopher Brewster	1 st review and changes
Version 0.8	11/09/2023	Louis Powell	Rework following reviews
Version 0.9	26/09/2023	Christopher Brewster	Edits, copyediting, formatting
Version 0.91	29/09/2023	Louis Powell	Final edits and formatting

Index

Summary	1
History of Changes	3
Index.....	4
Figure index.....	5
Abbreviations.....	6
Executive Summary	7
1. Introduction	7
1.1. Relationship to work package.....	8
1.2 Structure of the Deliverable	9
2. Purpose	9
3. Scope	9
3.1 System Infrastructure	9
3.2 API	10
3.3 Out of scope	10
4. System Infrastructure	10
4.1 Architecture	10
5 System Infrastructure	16
5.1 Micro-services	16
5.2 User Interfaces	17
5.3 Databases	18
5.4 Search.....	20
5.5 Natural Language Processing	20
6 API Software	21
7 Development Tools	22
7.1 Collaboration and Integration.....	22
8 Technical Deployment.....	24
8.1 Non-Functional Requirements	24
8.2 Orchestration.....	25
8.3 Service Discovery.....	25
8.4 Passwords and Access Keys.....	25
8.5 Continuous Integration	26
8.6 Load Balancing.....	26

9	References.....	26
10	Annexes.....	27
10.1	GitLab code repository	27
10.2	Technical Deployment.....	28

Figure index

Figure 1 – C4 Level 1 – System Context.....	11
Figure 2 – C4 Level 2 – Container	12
Figure 3 – C4 Level - 3 – Services Container.....	13
Figure 4 – C4 Level - 3 – API Container.....	14
Figure 5 – C4 Level - 3 – Security and Administration Components.....	15
Figure 6 – Next.js implementation.....	18
Figure 7 – FastAPI documentation.....	21
Figure 8 – FastAPI documentation – endpoint testing example.....	22
Figure 9 – Docker Desktop User Interface	23
Figure 10 - A view of EU-FarmBook GitLab repository registry	27
Figure 11 - A representation of Nomad workload orchestraton	30
Figure 12 - An example “Job Spec” for Nomad	31
Figure 13 - The Nomad User Interface.....	32
Figure 14 - The Consul User Interface	33
Figure 15 - An overview of Consul, Nomad and Vault.....	34
Figure 16 -The Vault User Interface	34
Figure 17 -The Jenkins CI/CD pipeline and User Interface	36

Abbreviations

AGROVOC	AGROVOC is a multilingual controlled vocabulary covering all areas of interest of the Food and Agriculture Organization of the United Nations, including food, nutrition, agriculture, fisheries, forestry and the environment.
AI	Artificial Intelligence
AKIS	Agricultural Knowledge and Innovation Systems
API	Application Programming Interface
C4	Software visualisation architecture model (Context, Container, Component, Code)
CORDIS	Community Research and Development Information Service
CRUD	Create, Read, Update, Delete
DEM	Demonstrator/Pilot/Prototype
DL	Deep Learning
DMP	Data Management Plan
DOI	Digital Object Identifier
EIP	European Innovation Partnership
ETL	Extract, Transform, Load
EU	European Union
EURAKNOS, EUREKA	Predecessor projects for EU-FarmBook
FAIR	Findability, Accessibility, Interoperability, and Reusability
FAQ	Frequently Asked Questions
FoodOn	A broadly scoped ontology representing entities which bear a "food role"
GDPR	General Data Protection Regulation
GET	GET is used to request data from a specified resource
GUI	Graphical User Interface
ID	Identification
JSON	JavaScript Object Notation
JSON-LD	JSON Linked Data
KG	Knowledge Graph
KO	Knowledge Object
M[0-9]	Month (month number starting from August 2022) M1 = August 2022, M6 = January 2023, etc.
MA	Multi-Actor
MS	Member States

MVP	Minimum Viable Product
NLP	Natural Language Processing
OG	Operational Group
PBIS	Product Backlog Items
PPT	PowerPoint
PU	Public
RDF	Resource Description Framework
RIA	Research and Innovation Action
SE	Sensitive
SPARQL	SPARQL Protocol and RDF Query Language
SSO	Single Sign-On
UI	User Interface
UX	User Experience
WP1/2/3	Work Package 1/2/3

Executive Summary

The following document provides the initial release of the System Infrastructure and API deliverable. This document is the outcome of extensive consultation between the technical partners in the project, analysis and discussion of the technical lessons learnt from the preceding EURAKNOS and EUREKA projects, and testing of available open-source software tools and design approaches. This document focuses on the infrastructure and API design, including software tools that support the 1st version of EU-FarmBook to be released in 2024. A series of Annexes provide technical details concerning technology choices, design decisions and technical infrastructure. This deliverable will be revised and extended every 18 months throughout the project's life.

1. Introduction

This document describes the system infrastructure and Application Programming Interface (API) at the core of the EU-FarmBook platform. It includes details of the functionality of various tools and modules that enable the overall platform to operate.

The purpose and objective of the EU-FarmBook project are to support knowledge exchange between all EU and national AKIS actors by further developing, expanding, exploiting and maintaining an easily accessible and user-friendly, EU-wide digital platform for practitioners in the agriculture, forestry and other rural sectors (including farmers, foresters, advisors, educators and trainers). The EU-FarmBook project will deliver a cross-media platform, focussing on ease of use and enabling user-friendly multilingual access to practice-oriented materials ('knowledge objects') generated by EU-funded and national research and innovation (R&I) projects.

The platform, on the one hand, will make it easy for existing and upcoming EU-funded and national agriculture and forestry research and innovation projects to share their knowledge outputs, and on the other hand, for a large variety of users to easily find access and reuse these knowledge objects for their agronomic practice, for their agricultural advisory services and wider stakeholder benefit.

End user ease of access across multiple platforms (web, mobile, tablet) using different modalities (simple text, chat, voice) will further stimulate research and innovation, enabling the outputs of EU-funded Multi-Actor (MA) projects, national projects and EIP Operational Groups (OGs) to have real impacts on agronomic and forestry practice across the EU and beyond.

The overall concept of the EU-FarmBook project is deeply embedded in the ambition of the European Commission to make European agriculture, forestry and rural enterprise more sustainable by strengthening the so-called Agricultural Knowledge and Innovation Systems (AKISs) that exist at regional, national and EU levels. Further, the project is complementary to the overall EU objectives in Open Science in enabling access to and reuse of all results from EU-funded projects.

The platform being developed in the EU-FarmBook project builds upon the work done in the EURAKNOS and EUREKA projects, especially regarding lessons learnt and the experience of previous technological choices. Nonetheless, the rapidly evolving domain of digital technologies and the necessity to adequately respond to unprecedented changes in knowledge- and data-related needs requires revisiting the choices already made and continuing to their refining per the expectations of the EU-FarmBook users and the AKIS actors at regional, national and EU levels. This further exploration and fine-tuning of technical specifications will be an ongoing exercise based on feedback loops for the timely capture of shifts in end-user needs. Furthermore, the continuous evolution of platform expectations and preferences, driven mainly by the design paradigms adopted by big corporate players (e.g., social media service providers), requires user research to be a continuous, iterative task, allowing the project to keep pace with the changes in user needs that will occur during the project's lifetime.

The EU-FarmBook is an RIA project that is designed for interactive action research. The project implements an approach to plan and coordinate project activities and the operation of a sustainable digital knowledge platform to respond to the 'evolving AKIS ecosystem' in all Member States (MSs) and sectors. This iterative and incremental approach facilitates flexibility, adaptation, and responsiveness to the dynamic needs and capacity of the EU (sectoral) and national-level AKISs.

1.1. Relationship to work package

This deliverable provides the core system infrastructure and API design and implementation under development by WP1 and WP2. Further versions of this deliverable will be provided at regular intervals (i.e., every 18 months), reflecting revised requirements as the project progresses. These revisions will reflect a combination of a) the experience of the technical team, b) requirements resulting from front-end user testing, and c) improvements in available software and security.

There is two-way communication between WPs 1+2. Other work packages have influenced and continue to influence the overall platform primarily through discussions

undertaken as part of T1.1 and feedback processes under development for delivery together with the first release. This deliverable has been written in tandem with D1.13 Data Standards and Knowledge Graph v1 and D1.6 Data Ingestion Pipeline and Upload Interface v1, and the three deliverables should be read as an integrated whole.

The current document is the version for M12.

1.2 Structure of the Deliverable

This deliverable is structured as follows: Section 2 explains the intention and purpose of this document and how it should be used. Section 3 describes the high-level scope for the Infrastructure and API in relationship to the EU-FarmBook platform. Section 4 introduces the technical drawings of system infrastructure and API components. Section 5 gives information on technical tools used for developing and deploying the platform. Section 6 highlights the specific software implementations that will power the EU-FarmBook infrastructure, followed by the API-specific software in section 7. Section 8 details the physical infrastructure on which the EU-FarmBook will be hosted and the tools that will support it to remain available and secure. The final section of the document consists of a series of Annexes detailing the requirements and design decisions with the requisite detail for actual implementation. Taken as a whole, the Annexes form the technical details of our upcoming release.

2. Purpose

The purpose of this document is to provide technical specifications behind the EU-FarmBook platform, due to be released internally in Autumn 2023 and officially launched in early 2024. These technical specifications are driven by various requirements-gathering exercises as detailed in deliverable D1.1.

While the content in the main body of this document is deliberately non-technical and aimed at all stakeholders, the content included in the annexes is deliberately more technical and gives specific information on the implementation and tools involved.

3. Scope

The scope of the specifications in this document relates to the technical details of the implementation of EU-FarmBook System Infrastructure, including the Application Programming Interface (API).

3.1 System Infrastructure

System infrastructure encompasses the foundational components required to support the technical operation of the EU-FarmBook platform. Example components include the dependencies for core EU-FarmBook website functionalities and features and the back-end software applications, for example, databases, code and software tools, which enable users, via interfaces including the website and upload form, to access EU-FarmBook services and maintain metadata and digital objects, namely Knowledge

Objects. As explained in Deliverable 1.1, the EU-FarmBook platform is built using a micro-services architecture.

3.2 API

The Application Programming Interface, or API, acts as a technical interface between the different components of the EU-FarmBook platform's overall system infrastructure and enables controlled and secure access to the platform's services for EU projects, Operational Groups (OGs) and national/regional repositories. These stakeholders can access knowledge objects and FAIR metadata through the API, facilitating data exchange, integration, and collaboration. As this document will evidence, the API adheres to industry-standard protocols, ensuring compatibility and interoperability with external systems.

3.3 Out of scope

To avoid doubt, the specific requirements and features of the EU-FarmBook platform, for example, user stories, are out of the scope of this document. This information can be found in deliverable D1.1 and its future versions.

4. System Infrastructure

4.1 Architecture

The research nature of this project gives the technical partners freedom from constraint of specific technologies and technical stacks. During the requirements defining stages, focus was given first to what the platform needs to do i.e., features, and then to how it will do it i.e., software implementations, based on testing different open-source tools, and considering proprietary tools already available at one of our institutions.

To make this distinction clear, this chapter on System Infrastructure will focus on what the platform is doing by introducing a series of technical drawings and describing the purpose of each component of the platform. The following chapter will focus on how these components have been built by describing the software which has been chosen to meet the necessary requirements.

4.1.1 The C4 model

First introduced in D1.1 - Platform Requirements and Design the C4 model (<https://c4model.com/>) has been used by the technical partners to describe the platform. It promotes simplicity by setting the context for the EU-FarmBook platform and system infrastructure at a high level and provides ever-increasing detail into the underlying software components, down to the lowest level, code.

The following sub sections of this document introduce the current versions of the first three levels of the model, and level 4, the code can be found in the private EU-FarmBook GitLab repository (<https://ci.tno.nl/gitlab/groups/eu-farmbook/>).

4.1.2 C4 Level 1 - Context

The 1st level of the C4 model gives a high-level overview of the scope of the architecture, its users and external dependencies. Although perhaps a simple and obvious representation of the EU-FarmBook, this level makes clear the primary users of the platform and their role concerning the high-level representation of requirements for the platform itself. This level is one of the first figures included in core technical documentation. It is set at a deliberately high level of abstraction to give a simple overview of the platform and its primary users and interactions with developers and stakeholders with little or no background knowledge of the EU-FarmBook project.

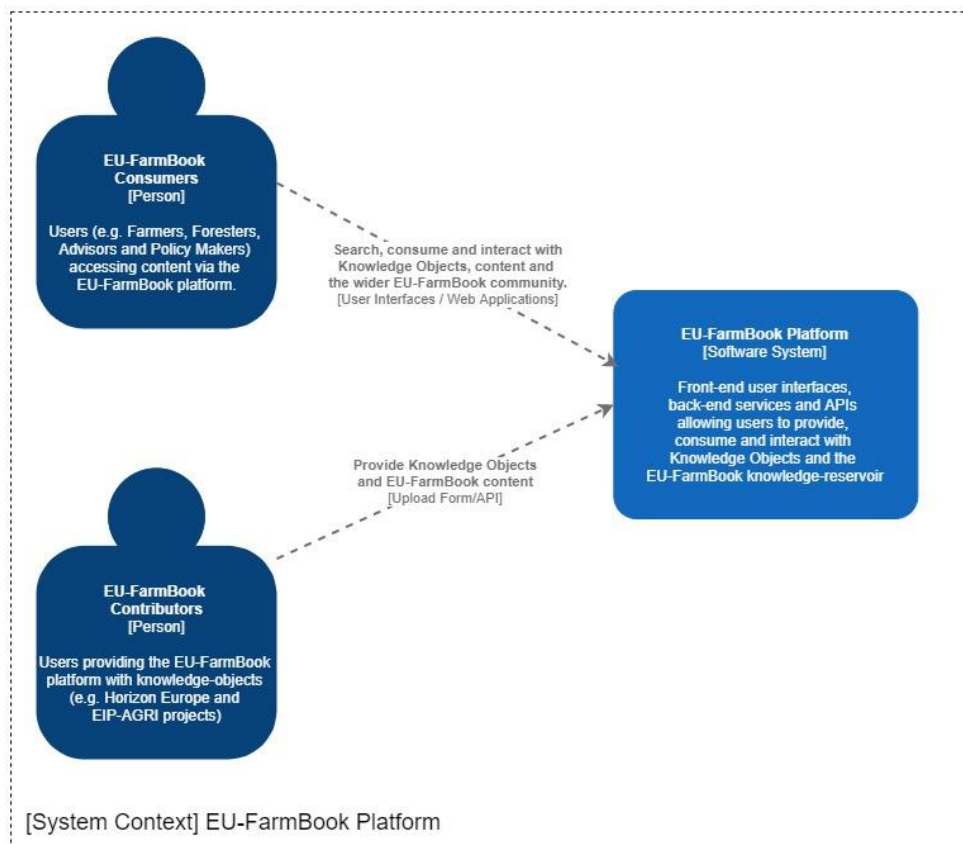


Figure 1 – C4 Level 1 – System Context

4.1.3 C4 Level 2 – Container

The framework's high-level architecture is depicted in Figure 2, providing an overview of conceptual layers (e.g., Databases and Interfaces) and their interactions. These layers are named "containers" in the C4 model, not to be confused with Docker containers (see section 5.2.2). The diagram illustrates the various layers or "containers" that constitute the system infrastructure and API.

Interfaces represent the external points of interaction with the system. These interfaces facilitate data input, system configuration, and output visualisation. The diagram

illustrates the connection between users, contributors, and the system through these interfaces. For example, the upload form interface allows contributors to upload Knowledge Objects and their metadata.

At the system's core, the API layer serves as a bridge between different components. It enables communication and data exchange via web addresses or "endpoints" within the EU-FarmBook platform, each of which perform a different task, for example, store a Knowledge Object or access metadata.

The databases store the actual or "physical" data the platform receives or extracts from each source/input in a "physical" layer, and transformed or "logical" data, for example, for enrichment with references ontologies and vocabularies, is stored in a "logical layer".

Different database implementations (e.g., SQL and NoSQL) accommodate structured and unstructured data, allowing for efficient storage and retrieval.

The services container hosts essential functionalities, and each service is responsible for one or more tasks, for example, Natural Language Processing (NLP).

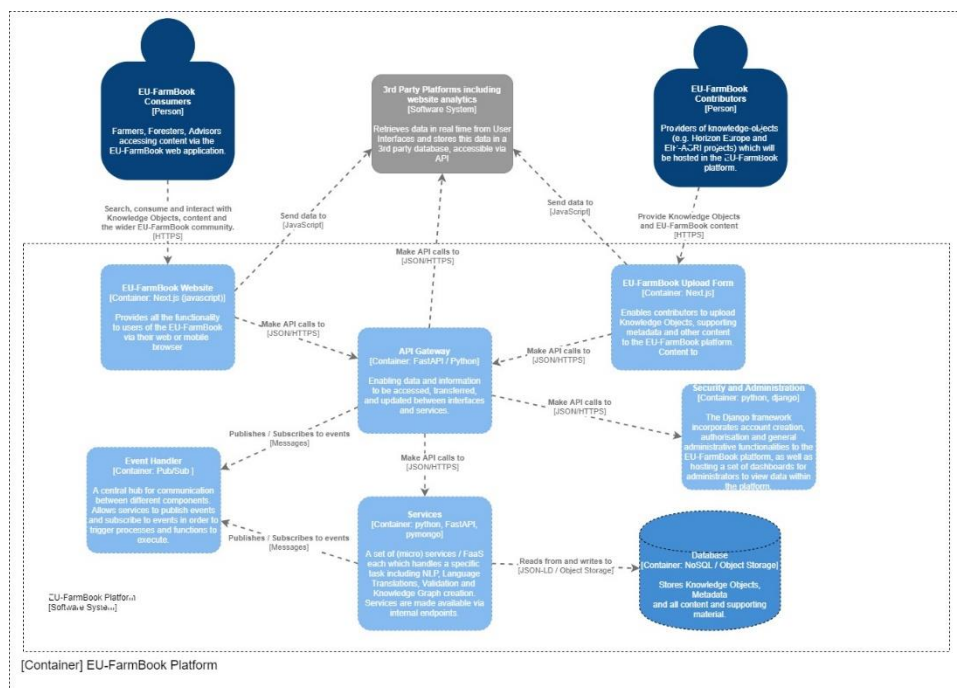


Figure 2 – C4 Level 2 – Container

4.1.4 C4 Level 3 – Component

At Level 3, we move away from the more conceptual view of the platform and into each container's functional components. This section focuses on the primary back-end containers, namely Security and Administration, Services (including database interactions) and API. Further information about the front-end components can be found in chapter 5.2.

Although not always an accurate distinction, these components can generally be considered a microservice or part of a microservice, communicating with other components via the API.

In this level of the C4 model, each component contains an overview of the features and tasks it performs to enable EU-FarmBook to store Knowledge Objects and ensure the website and interfaces keep up to date and online. The software involved is mentioned for each service, and the lines also indicate the dependencies, protocols and procedures for transferring data and information between each microservice.

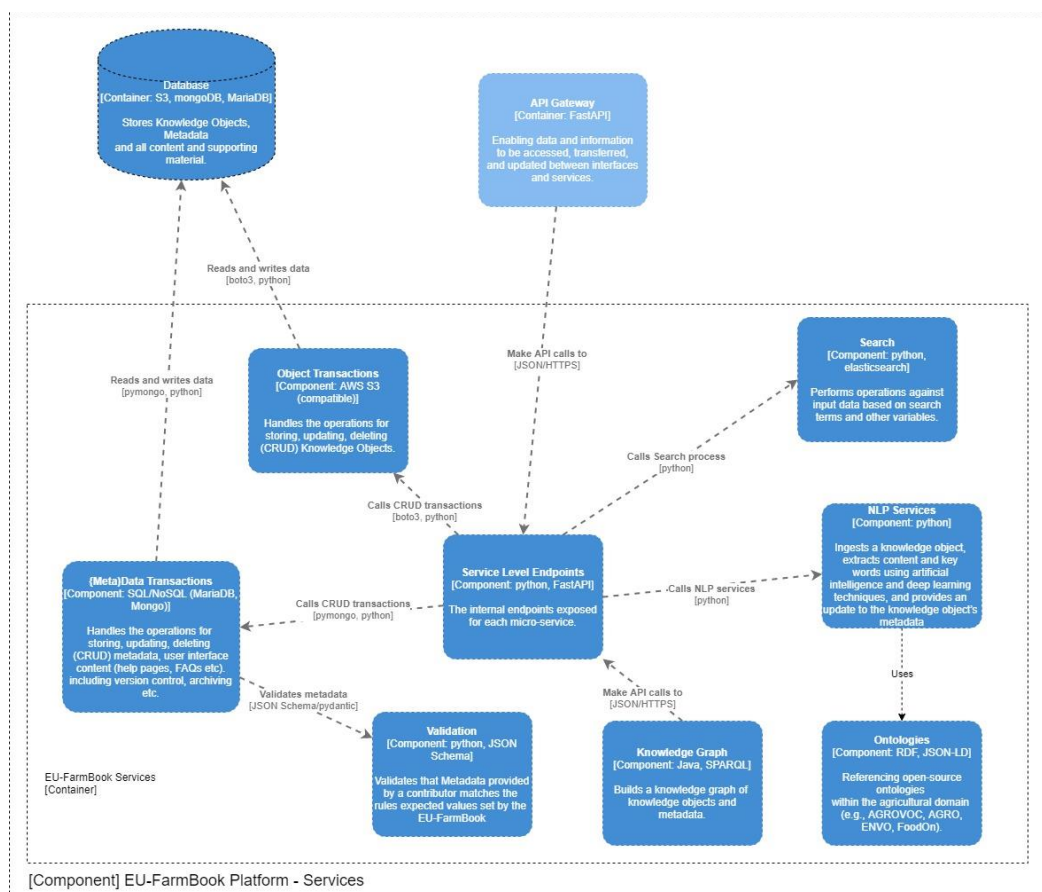


Figure 3 – C4 Level - 3 – Services Container

- **Service Level Endpoints** receive API requests from the main API Gateway and perform the necessary actions to return an output based on a set of given inputs.
- **Metadata transactions** responsible for performing CRUD (Create, Read, Update, Delete) transactions or operations on Mongo DB databases. This has a dependency on open-source tools and libraries, most significantly pymongo (<https://pymongo.readthedocs.io/en/stable/>)
- **Search** responsible for taking keywords and inputs from users and returning relevant Knowledge Objects. The EU-FarmBook first release supports both

Elasticsearch (<https://www.elastic.co/elasticsearch/>) and Mongo DB native search endpoints.

- **Object transactions** – responsible for performing CRUD (Create, Read, Update, Delete) transactions on Knowledge objects stored in an S3-compatible (<https://aws.amazon.com/s3/>) object service database.
- **Natural Language Processing (NLP)** – responsible for finding and connecting open-source ontologies and vocabularies and extracting metadata from Knowledge objects.
- **Ontologies** – responsible for connecting the EU-FarmBook to external ontologies (e.g., via AGROVOC's SPARQL endpoints) which can be queried, for example, by returning unique identifiers (URIs) for keyword matches.
- **Validation** – responsible for using JSON-Schema (<https://json-schema.org/>) and pydantic (<https://docs.pydantic.dev/>) to validate metadata and other data against a specific schema to promote interoperability and completeness/FAIRness.
- **Publisher/Subscriber** – responsible for managing and triggering events within the platform. Redis (<https://redis.io/docs/>) is being considered for the initial release version of the EU-FarmBook.
- **Knowledge Graph** – responsible for hosting a SPARQL endpoint (<https://www.w3.org/TR/rdf-sparql-query/>), which will allow students and researchers to query the EU-Farmbook knowledge base and databases directly in an RDF based Knowledge Graph and provide a key resource for the eventual integration of the chatbot.

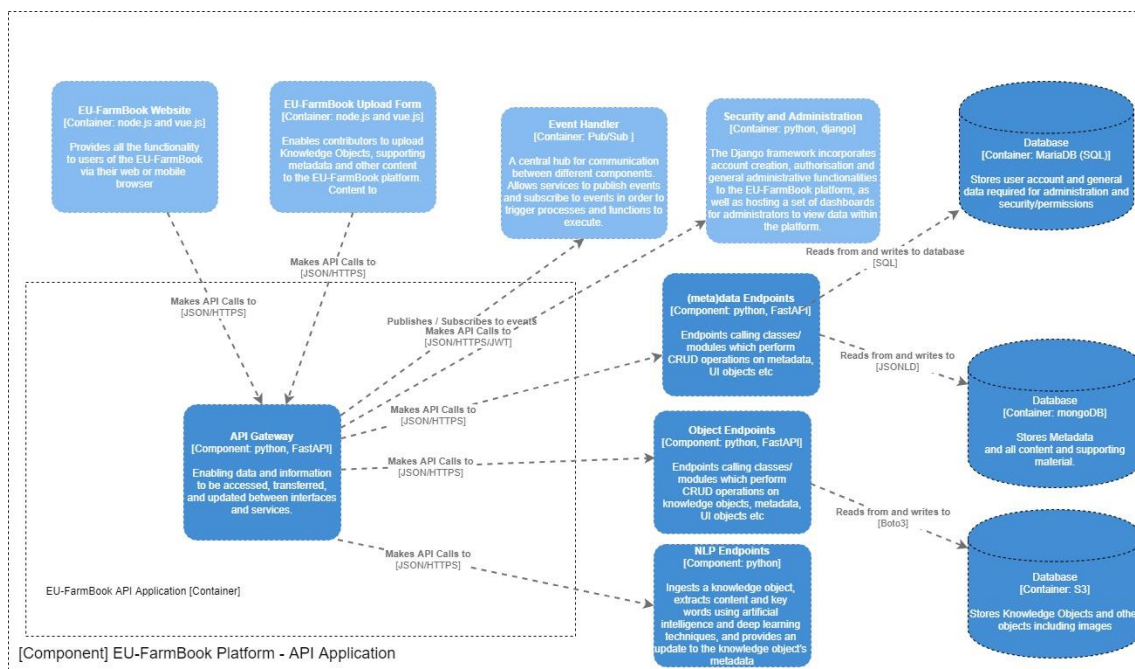


Figure 4 – C4 Level - 3 – API Container

Figure 4 highlights the API at the core of the EU-FarmBook, responsible for enabling communication between the different components.

The API hosts a set of web addresses or endpoints which can receive information and different inputs criteria in the form of an HTTP request.

The main gateway is available via the internet and directs requests into the service level endpoints which cannot be reached publicly. This adds an extra layer of security on top of the platform, as we will not expose the methods and software being used to perform actual database operations.

The API gateway receives a request, for example, to upload or POST a Knowledge Object. It directs the request to the Object Endpoints, the operation is performed against the database, and a message is returned providing a unique identifier for the Knowledge Object. This now enables metadata to be gathered and stored against this identifier via another API request which would now use the Metadata Endpoints, transactions and corresponding database.

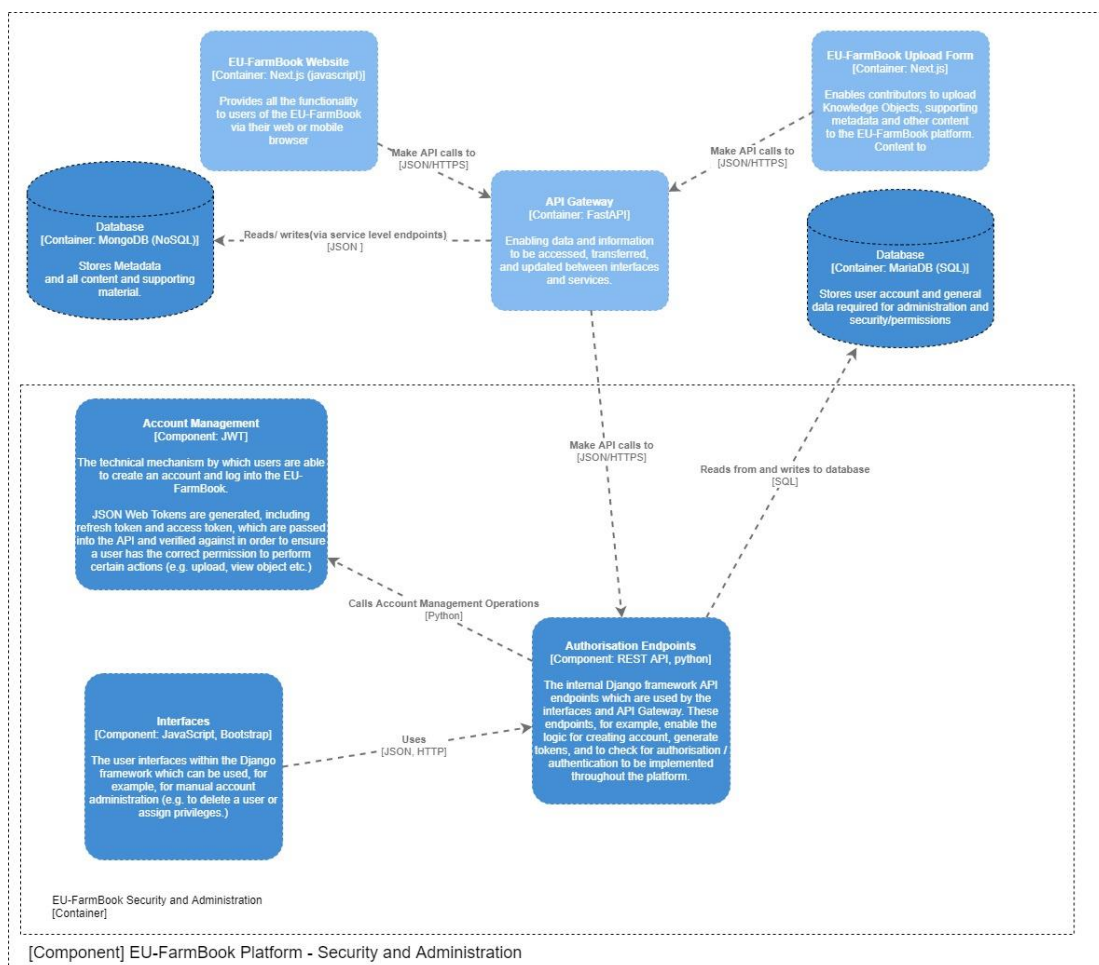


Figure 5 shows the primary components within the security and administration layer of the platform. For the first version, this is largely built using the Django framework, and an SQL implementation of MariaDB is used to store user details. The platform's FastAPI gateway interacts with the Django admin panel, by sending requests and receiving information in return. FastAPI accepts the username and password from the client and forwards them to the Django-based service for authentication. Using RESTful API architecture, the Django service queries the MariaDB database to validate these credentials.

Upon successful validation, the Django service issues access and refresh "tokens" (JSON Web Tokens or JWT) back to the FastAPI layer, further ensuring secure, stateless interactions. This REST API is also the gateway for any administrative functions, providing a programmable interface for typical tasks executed through the Django admin panel, built with a combination of JavaScript libraries and Bootstrap for a streamlined user experience.

Once a user is authenticated, they can create, upload, or modify Knowledge Objects and metadata via distinct API endpoints, provided they have the necessary user rights to do so.

By layering these different components, we create a highly modular system and add multiple levels of security without revealing the underlying database operations and software methodologies.

5 System Infrastructure

5.1 Micro-services

During the design, development and documentation of the EU-FarmBook platform, a core consideration of the consortium partners has been how to promote clear communication of technical architecture and infrastructure decisions both internally and for external stakeholders, given a vast array of domain-specific technical concepts and terminology.

One of the key ideas adopted to achieve this is the concept of a **micro-services** architecture or approach. This is at the core of the design and implementation of the System Infrastructure and API in EU-FarmBook. Traditional software architecture generally follows a "monolithic" application approach. The Monolithic application is software in which different components (such as authorisation, business logic, notification module, etc.) are combined into a single tightly integrated program [1]. Typically, examples of such monolithic systems are those used by banks and insurance companies. While this approach still has merit in software development, some significant considerations exist for the EU-FarmBook.

The project runs for seven years, during which time and beyond, the EU-FarmBook must continuously adapt to users' requirements while integrating relevant developments in data, software, and technologies such as AI (including NLP, DL and LLMs). Given such factors, as well as the wide range of technical contributors to the EU-FarmBook, the design and development of the platform's System Infrastructure and API is more suited to a microservices approach. "Microservices are small, autonomous services that work

together" [2]. Interoperability with other platforms (e.g., national repositories) and external services (e.g., translation software) is facilitated by developing and deploying each component of the EU-FarmBook (e.g., databases, interfaces, user management) as a separate service. This service can be updated or replaced without impacting other components or services significantly. This allows development to be agile and for the project to benefit from advancements in open-source tools and technologies (e.g., Large Language Models (LLM)) while continuously building and integrating new features and facilitating interoperability with other platforms.

Although not expected to be a future requirement, the approach to building each component also supports deployment as a monolith, albeit with some technical overhead.

5.2 User Interfaces

Most users of the EU-FarmBook will interact with the platform via the main user interface, specifically the eufambook.eu website. For the 1st version of the EU-FarmBook, the chosen framework for the interfaces is a Next.js implementation of React.

“Used by some of the world's largest companies, Next.js enables you to create full-stack Web applications by extending the latest React features and integrating powerful Rust-based JavaScript tooling for the fastest builds.” (<https://nextjs.org/>).

A graphical representation of the Next.js implementation and its core dependencies is available in figure 6.

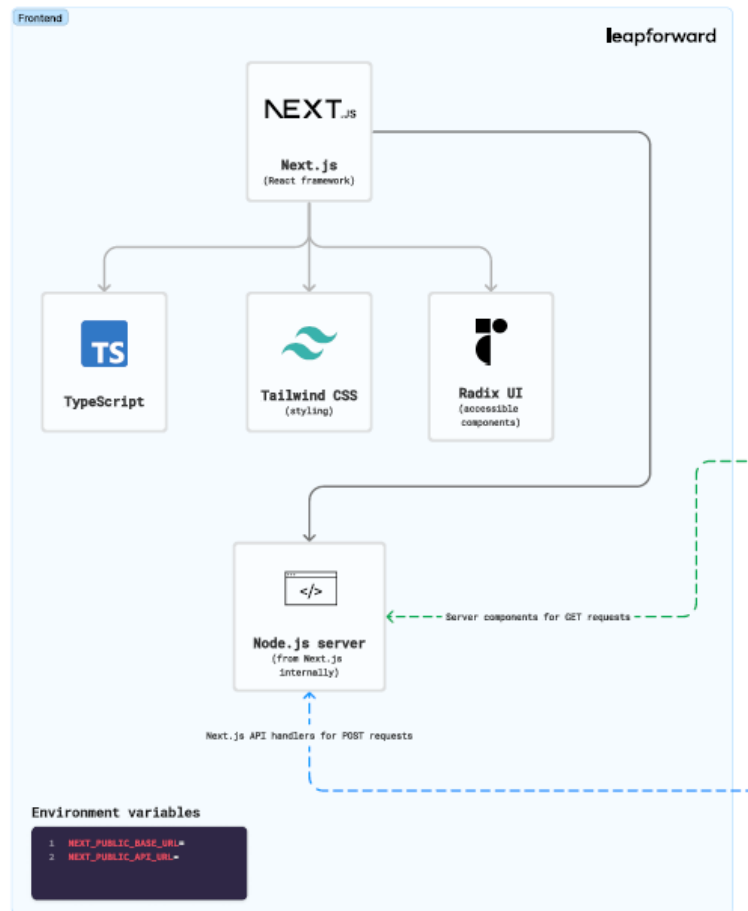


Figure 6 – Next.js implementation

5.3 Databases

Databases support the physical storage of digital objects in the form of structured or unstructured data, which can be stored, organised, updated, retrieved and, where necessary, archived and deleted. There are two primary conceptual groups of data in consideration for this project: objects (i.e., Knowledge Objects including PDF, DOC(X), etc.) and their metadata which is stored in the structure of a JSON (JavaScript Object Notation) document.

5.3.1 Objects

One of the key value propositions of the EU-FarmBook is the long-term storage of Knowledge Objects.

To ensure scalability as well as secure and efficient lifecycle management (e.g., storing, updating, versioning, translating, archiving, deleting, etc.), the EU-FarmBook platform will store objects, including Knowledge Objects, in an AWS S3 (Amazon Web Services Simple Storage Solution) compatible platform. This will be hosted by the University of Gent in Belgium and storage is on-premises. Regular snapshots and backups will be taken and made available as necessary.

Databases like S3 use the concept of a bucket for storing objects. Buckets are analogous to folders; in that they allow data to be stored and maintained under a pre-defined structure. This feature enables systematic structuring of the content within the EU-FarmBook as well as the ability to archive and back up data in line with data retention policies as defined in the Data Transfer Agreement.

5.3.2 (Meta)data

There are two families of storage and structure available in open-source database management. These are Structured Query Language (SQL), primarily used for relational databases, and its alternative, NoSQL, which is for non-relational databases.

A conventional SQL database (MariaDB) will be used to handle configurations and user management. the primary database of the EU-FarmBook, in particular Knowledge Object metadata, is a MongoDB implementation using an interoperable format called JSON (JavaScript Object Notation) documents. MongoDB is a popular open-source NoSQL database.

The decision to use MongoDB and NoSQL rests on three primary considerations:

- Contributors to the EU-FarmBook are National, Regional, and Operational Groups and Projects whose platforms cover various domains, countries and languages. Therefore, we expect data of a heterogenous nature in terms of the ontologies and structures they can provide to better represent their Knowledge Objects in the EU-FarmBook beyond the standard metadata we collect during the upload process.
- The overall project must abide by the FAIR data principles (i.e., making data findable, accessible, interoperable, and reusable through standardised descriptions, locations (e.g., URIs), and semantic annotations) per the Open Science demands of the EC for all funded projects. Adherence to the FAIR principles is at the core of the EU-FarmBook data management strategy. Using the FAIR data principles also accords with the overall needs and priorities of the project (see Eureka Deliverable D3.2). A natural consequence of following the FAIR data principles is using semantic technologies, including RDF/RDFS. The Resource Description Framework or RDF is a structure for representing data on the web in a way that computers can understand (i.e., machine-readable). It enables data to be effectively shared, integrated and utilised across diverse research contexts. MongoDB supports data stored in JSON-LD format (JSON for Linked Data). JSON-LD () can be used to express data in a way compatible with RDF. In short, this approach enables us to ensure EU-FarmBook metadata is Findable, Accessible, Interoperable and Reusable.
- The flexibility of a NoSQL database allows WP1 and WP2 to quickly adapt and modify data models as the EU-FarmBook application evolves. This helps avoid a slow change management process when implementing new features and updates.
- Again, regular snapshots and metadata backups will be taken and made available as necessary.

5.4 Search

To increase the likelihood that a target user of the EU-FarmBook (e.g., a farmer or forester) can find and access relevant material (e.g., Knowledge Objects and their metadata), we must continually research and develop the logic that determines how their "search" terms are converted into results and recommendations.

There are many ways in which this can be done, and the platform's ability to interpret what users are looking for based on their search terms, analyse website analytics and optimise the results they see will ultimately play a significant role in the success of EU-FarmBook from both a usability and applicability perspective.

This approach is similar to how popular search engines like Google display articles and links to websites and other content after you type in your interest.

Elasticsearch is a search and analytics engine designed to explore, analyse, and visualise data quickly and in near real-time. It excels at full-text search, making it ideal for applications that efficiently search through large volumes of text-based data. Indexing and querying capabilities allow fast and accurate retrieval of relevant information from diverse sources. For this to function, all knowledge objects will have to be converted to plain text for the purpose of search indexing (cf. NLP below).

As data is collected by the platform in the early stages, the search algorithms will be continuously tested, developed and released to connect users with as appropriate recommendations and search results as possible. D2.4 - Search and Recommender System provides further information on this component of the platform.

5.5 Natural Language Processing

The NLP service ingests knowledge objects to automatically interpret them and output metadata and other relevant information. This metadata, as noted above, has many purposes, including enhancing the effectiveness of the search functionality of the EU-FarmBook and reducing the burden on individuals providing manual annotations of knowledge objects. The technology behind this service is based on Artificial Intelligence and Deep Learning (Transformers-based models). Despite some initial developments using more classic approaches, due to the emergence of Large Language Models (like ChatGPT and subsequent models) we are incorporating them to the process. The precise processing pipeline may evolve as we develop the technology and the metadata-extraction requirements further, but as it is devised now, it consists of a first step of extracting the raw content of a given Knowledge Object (for instance, a PDF document). Then, the raw text content is passed to a Large Language Model to be interpreted. The Large Language Model outputs metadata such as a suitable title, relevant keywords, topic classification or a short summary/description. The technological landscape is quickly evolving in this regard, and we will keep adapting, improving, and extending the NLP service capabilities during the project's lifetime.

6 API Software

An API, short for Application Programming Interface, allows different software components to communicate and interact with each other via pre-defined addresses or "endpoints", which each perform a specific task or set of tasks.

Specifically, within the EU-FarmBook platform, an API plays a crucial role in connecting the website with back-end databases and other services, enabling contributors to upload large numbers of knowledge objects directly when the upload form is not appropriate.

Following its first release in 2018, the open-source framework Fast API has gained popularity in API implementations due to its speed, simplicity and wide range of features, including automatic documentation generation (see Figure 7). This allows users to view and test each of the API endpoint definitions directly from their browser, and whenever a change is made, the documentation is updated automatically. This has been chosen as the framework for the first version of the EU-FarmBook API.

FastAPI 0.1.0 OpenAPI

API Status ^

GET	/api/status/api_status	Get Api Status
GET	/api/status/db_status	Get Db Status

Knowledge Object ^

POST	/api/knowledge_object/upload_knowledge_object/	Upload Knowledge Object
GET	/api/knowledge_object/get_knowledge_object	Get Object

Metadata ^

GET	/api/logical_layer/metadata_schema	Get Metadata Schema
GET	/api/logical_layer/data_model/{property_name}	Get Data Model Properties
GET	/api/logical_layer/interface_model/{object_name}	Get Interface Model Objects
GET	/api/logical_layer/document/{document_id}	Get Document
GET	/api/logical_layer/documents	Get Documents
POST	/api/logical_layer/documents_filtered	Get Documents Filtered
POST	/api/logical_layer/search	Search Documents
POST	/api/logical_layer/search_new	Search Documents New
GET	/api/logical_layer/count_documents/{property_name}	Count Documents By Property Name

Figure 7 – FastAPI documentation

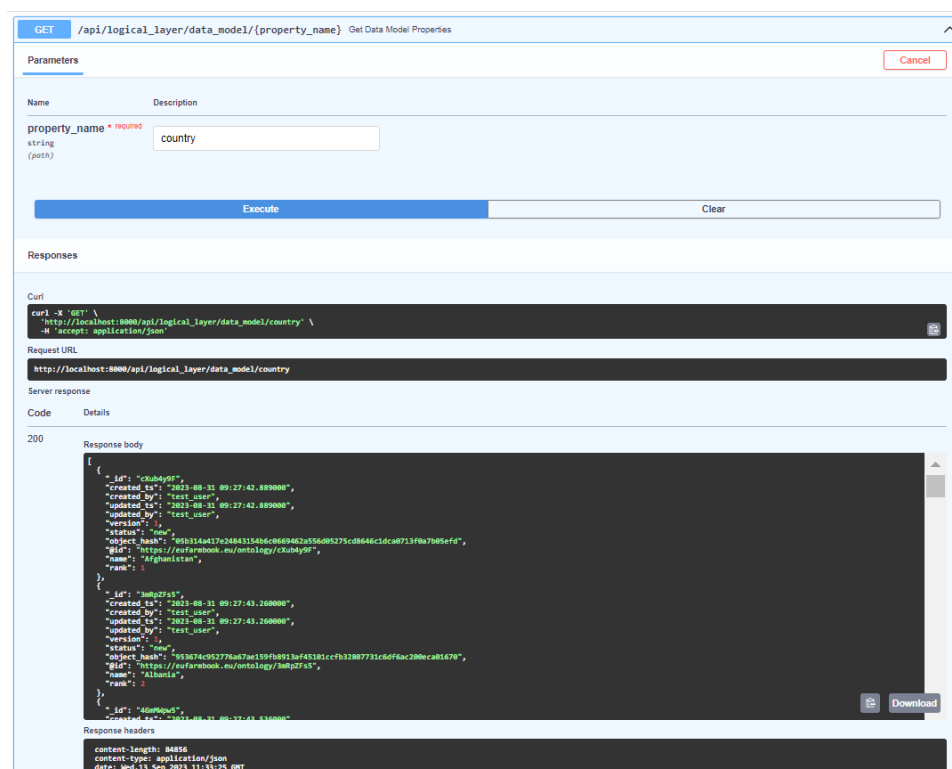


Figure 8 – FastAPI documentation – endpoint testing example

In simple terms, Fast API acts as the main gateway of the EU-FarmBook, enabling the website to request and manipulate data stored in the databases without directly connecting to the data. It promotes interoperability with other platforms by enabling (bulk) upload of knowledge objects, metadata, and other relevant information.

7 Development Tools

7.1 Collaboration and Integration

Over 20 partners across 12 organisations are involved in WP1 and WP2, many of whom directly contribute code and software. These partners use a wide variety of tools, particularly Integrated Development Environments (IDEs), to support code editing, compiling, debugging and other challenges. What is most suitable for each software task and each partner differs, but the tangible output from each development must fit two key criteria.

- Collaboration - writing code and building software as a team.
- Deployment - ensuring whatever is built can be deployed anywhere and not dependent on specific development environments.

There are two popular and widely used open-source tools, Git and Docker, that simplify adhering to these criteria.

7.1.1 Collaboration

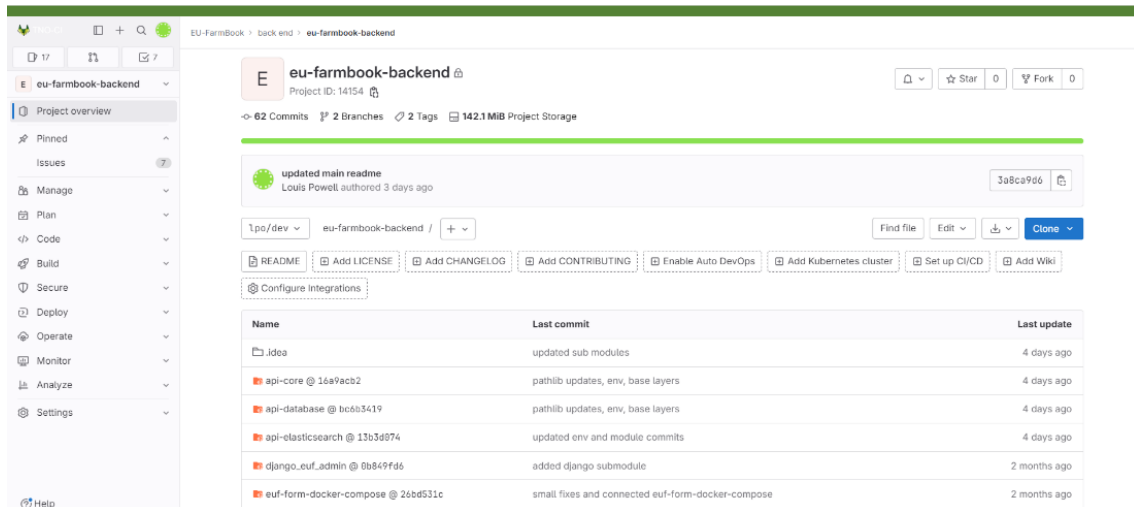


Figure 9 - GitLab User Interface

The tangible deliverable at the base of the EU-FarmBook platform is the actual code produced by WP1 and WP2, which sits behind each micro-service. A version control system, or VCS, tracks the history of changes (to code) as developers collaborate on projects together. Earlier versions can be recovered anytime. This open-source VCS is one of the most commonly used tools in software development, with over 90% of developers surveyed by the popular site Stack Overflow responding that they use Git [3].

GitLab, hosted by our partner, The Netherlands Organization for Applied Scientific Research (TNO), simplifies the EU-FarmBook development by offering a solution for version control (using Git) and project management tools that help organise development tasks and link code to specific requirements.

7.1.2 Integration

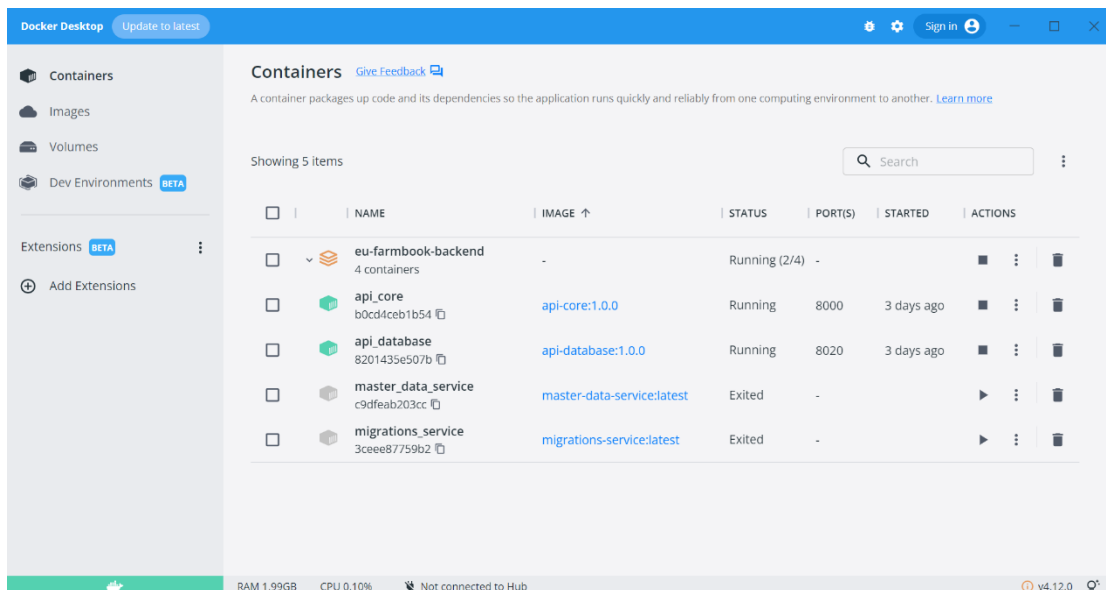


Figure 10 – Docker Desktop User Interface

Docker is a versatile platform that simplifies software deployment by packaging applications and their dependencies into portable containers. These containers can run consistently across various environments (e.g., MacOS, Windows, Linux), enhancing application reliability and efficiency. Docker abstracts away differences in operating systems and infrastructure, ensuring that applications behave predictably across development, testing, and production environments.

Through containerisation, Docker streamlines the process of building, shipping, and running applications, enabling developers to focus on coding rather than dealing with complex environment setups (<https://www.docker.com/>).

This technology has revolutionised software development, enhanced application consistency, scalability, and isolation while improving resource utilisation. Docker's container-based approach optimises the deployment pipeline, making it an integral tool for modern software development and deployment workflows.

8 Technical Deployment

8.1 Non-Functional Requirements

While many of the software introduced in this document provide functionality to the platform, a robust and successful platform has many non-functional requirements to consider. Non-functional requirements include security, performance, scalability and availability, and such requirements are significantly impacted by the technologies and tools used when the platform is deployed.

While the micro-services approach to building the EU-FarmBook, including Docker, allows it to be deployable anywhere, the production version launched to the public in February 2024 will be deployed and hosted at the University of Gent in Belgium. The University of Gent also provides on-premises physical storage of data and will host the main website and back-end services.

While the annexe section 10.3 gives technical details of the deployment framework and security mechanisms in place at UGENT, this section briefly overviews some of the important software tools supporting the release and management of the EU-FarmBook platform during development and production. These software tools are already available and in operation at the University of Gent, providing the EU-FarmBook with an existing stable and tested environment. These tools also perform key roles in supporting non-functional requirements as highlighted in the following sections.

Once the first version of the platform is released, it will also be mirrored in two other locations (the University of Maastricht in The Netherlands and the Agricultural University of Athens in Greece).

8.2 Orchestration

One of the most important considerations for the brand and reputation of the EU-FarmBook platform is that it remains available to users with little or no downtime. Websites which are unreliable or suffer regular outages can expect to lose the confidence of users and stakeholders. This could significantly impact the EU-FarmBook, for example, limiting the Ambassador's ability to encourage projects and Operational Groups to contribute their content to the platform. While it is inevitable that there will be technical issues outside of the project's control, there are several solutions which can be adopted to minimise the impact and keep the website and API accessible. These solutions support the orchestration of running a platform.

HashiCorp Nomad is a program that allows a platform like the EU-FarmBook to be distributed over multiple servers or computers. When one server becomes unavailable (i.e., it crashes), Nomad allows a different server to take over its tasks. This way, the platform remains online even though one of the servers experiences a fatal error. Because Nomad distributes applications over multiple servers or computers, it is also far less likely to overload one specific server. This can also potentially increase the performance of an application.

8.3 Service Discovery

When developing and building the EU-FarmBook and its future releases, it is important to avoid repetitive tasks that can be otherwise automated and support faster deployments and more efficient monitoring of how the platform and its services perform. Service discovery automatically detects devices and offers services over a network. Discovery, which minimises administrator configuration efforts, is commonly found in microservice architectures and containerisation platforms. [4]

HashiCorp Consul is a program that contains a catalogue of all applications or services. Other computers or applications can determine where a certain application is running by looking at the service catalogue in Consul.

Without such a service discovery tool, applications distributed over multiple servers via Nomad would require constant manual oversight and input, creating a maintenance cost of sorts and introducing human error into an otherwise automatable set of tasks. This approach promotes efficiency in terms of cost and accuracy while helping ensure the platform does not suffer from unnecessary down time.

8.4 Passwords and Access Keys

Access to individual components or services of the EU-FarmBook, for example, the databases, is managed using combinations of account information, access keys and secret keys or passwords. The platform also uses dynamic ports whereby the physical location of a service changes regularly. Although the platform will never externally expose such passwords or the logic behind the port mapping, it is imperative we protect the physical environment on which the platform will be hosted to reduce exposure to hacks and malicious traffic.

HashiCorp Overview Vault is a program that acts as a digital safe or Vault for secret data (e.g., application passwords). It is mainly intended to be used by applications or computers and not by humans. For example, applications running on Nomad can leverage Vault to access passwords securely without exposing them externally.

Without use of such a tool, there would be a significant manual overhead and unnecessary risk due to potential human-error whereby a technical partner would need to maintain passwords and manually update them across the different services and components.

8.5 Continuous Integration

A prevalent terminology in software is CI/CD or Continuous Integration and Development. This describes a practice of continuous delivery and deployment of software that enables platforms to frequently and reliably release new features. This is, in part, made possible by automating the regular steps that occur once a new feature is built but not yet released, for example, testing, version control and deployment.

Jenkins is the leading open-source automation server with some 1,600 to 1,800 plugins to support the automation of all kinds of development tasks. The available plugins span five areas: platforms, UI, administration, source code management, and, most frequently, build management. [5]

8.6 Load Balancing

Apache HTTPD (or Apache for short) is the software responsible for serving a web page when one visits a website on the Internet.

The EU-FarmBook also uses a specific functionality of Apache that can distribute a request from a user's browser (e.g., Google Chrome) over multiple Nomad servers. This allows the EU-FarmBook to be scalable as it grows over time, while ensuring that applications are available as much as possible, even when a Nomad server crashes.

9 References

- 1) Gos, Konrad & Zabierowski, Wojciech. (2020). The Comparison of Microservice and Monolithic Architecture.
- 2) Newman, S.: Building Microservices. O'Reilly Media, Inc., Sebastopol (2015)
- 3) <https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/>
- 4) <https://www.techtarget.com/searchitoperations/definition/service-discovery>
- 5) <https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html>

10 Annexes

10.1 GitLab code repository

All code developed for the platform, including version history, branches etc can be found in the GitLab instance at TNO University.

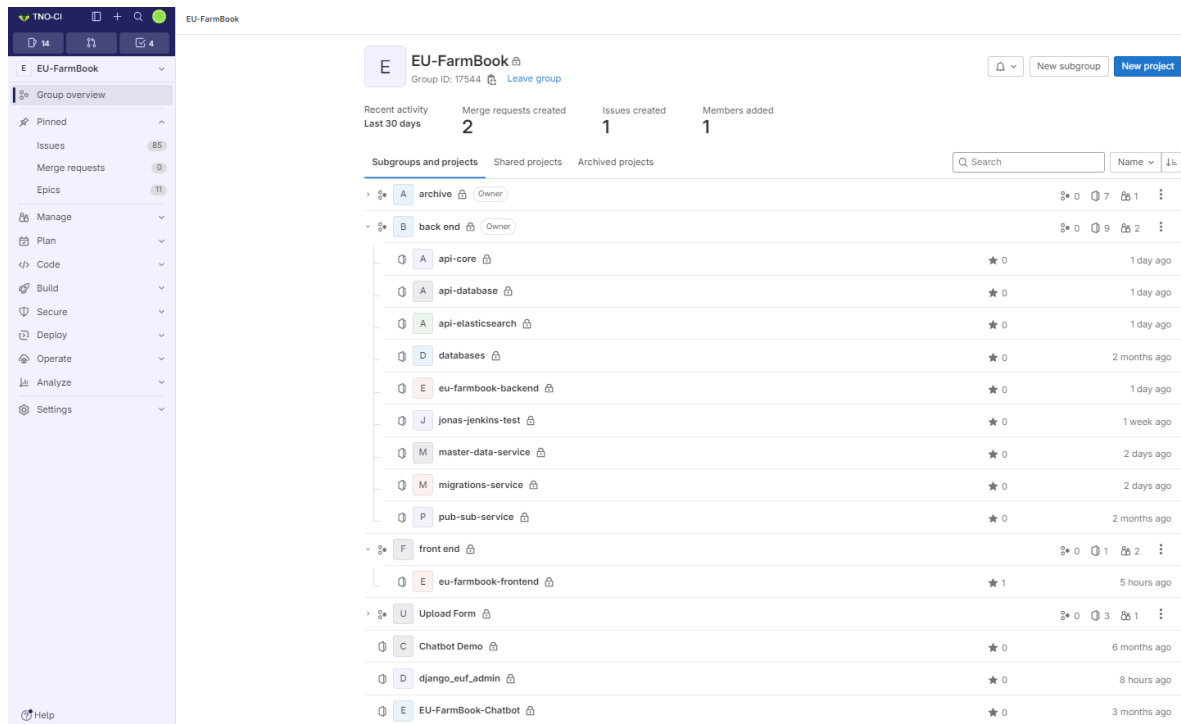


Figure 11 - A view of EU-FarmBook GitLab repository registry

Each repository contains one or more readme.md files used to document its features and explain how it can be deployed either locally or in a Docker Container.

The GitLab instance is not open source, and such access must be granted by TNO to reviewers individually, after which the code and other information including Wiki pages and Kanban boards can be found at the address <https://ci.tno.nl/gitlab/groups/eu-farmbook/>.

10.2 Technical Deployment

EU-FarmBook Hosting Infrastructure

This document describes how the server infrastructure for the EU-FarmBook project is set up at the University of Gent. This includes an overview of the tools/technologies used, as well as how deployment of application components is performed.

Base Infrastructure

Environments

As of writing, DEV is the only environment available. A separate production environment is planned and will be rolled out during the coming weeks once our Vault setup is production ready.

Operating System

The base operating system used for all servers is Debian Linux 12 ("Bookworm"). This release will be supported until the next Debian stable version is released (approximately 2-3 years), after which the Debian LTS Project takes over maintenance. We always strive to have the latest Debian stable version running on as much as servers as possible.

All machines have automatic security updates and automatic reboots enabled. For our Nomad clusters, this means that a rolling reboot will be performed when a reboot is required. It is assumed that the applications running on the cluster are designed to be highly available and thus will remain online during this process. One or more machines should also be able to be taken offline (e.g. for maintenance) without impacting applications running on the cluster. All machines also have a firewall (Linux Netfilter / iptables) enabled, which blocks all incoming traffic by default. Where possible, traffic is encrypted via (m)TLS between applications and servers.

Puppet, our configuration management system, also makes sure that every machine that we manage has the same, secure baseline configuration. Some things that are managed by Puppet:

- User Accounts
- OpenSSH server configuration
- Firewall rules
- Apache vhosts
- Docker/Podman
- Nomad (*)

(*) Puppet manages the installation and configuration of Nomad itself on the servers. Jobs and other things are managed mostly by Jenkins (see below).

Network

By default, nothing is reachable from outside the UGent network without a VPN. This is achieved via networking ACL's (Access Control Lists) on the networking equipment. Generally, the only machines that are reachable from the Internet are the Apache load balancers, which distribute incoming traffic to the correct Nomad machines.

Tools

HashiCorp Nomad

Non-technical Overview

Nomad is a program that allows to distribute an application over multiple servers or computers. When one server crashes, Nomad allows the other servers to take over the work that was running on the crashed one. This way, the application remains online, even though one of the servers experienced a fatal error.

Because Nomad distributes applications over multiple servers or computers, it is also far less likely to overload one specific server. This can also potentially increase the performance of an application.

Technical Overview

Nomad is our workload orchestrator, which is mainly used for deploying and running Docker containers at scale. It's similar to other container orchestration technologies like Kubernetes or Docker Swarm. The main advantage of Nomad is that it's much simpler to operate than e.g., a Kubernetes cluster. It also integrates well with other tools of the HashiCorp stack, which will be covered later.

A Nomad cluster consists of two types of machines:

Servers: These machines form a Raft cluster. This is where all scheduling decisions will take place (e.g., "Which *node* fulfils *all requirements* for Job X?"). This node type can be compared to Kubernetes *control plane* nodes.

Clients: On these machines, the actual workloads (Docker containers) will be scheduled. This node type can be compared to Kubernetes *worker* nodes.

On every machine, there's also a Consul Agent and Vault Agent daemon running, which are used to interact with Vault and Consul respectively (covered later in this document).

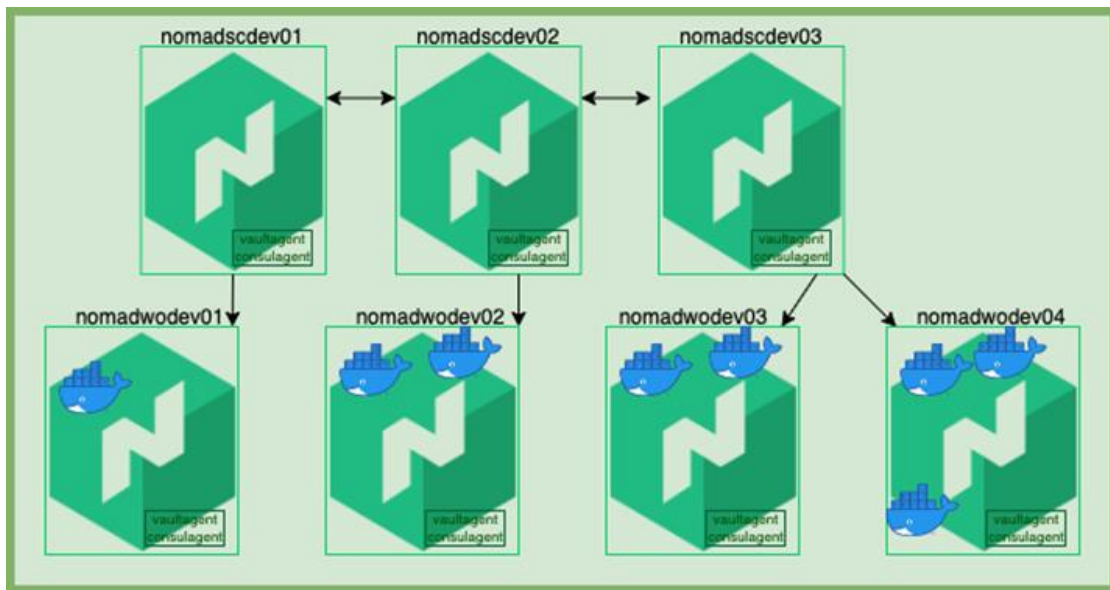


Figure 12 - A representation of Nomad workload orchestration

Access to cluster resources is controlled via Nomad ACL's. For FarmBook, there's a separate namespace provided for all FarmBook-related applications. A valid Nomad ACL token can be obtained via Vault (see below).

The main unit of work that one will interact with is a JobSpec (short for *job specification*). A Job Spec describes *how* Nomad will run an application on the cluster. An example can be found below.

```

variable "image" { (1) type = string
}

job "pearl-exporter" { type =
  "service" datacenters =
  ["S10"] namespace =
  "multimedia"

  group "web" { (2) count
    = 1

    network { (3) port
      "http" {
        to = 9115
      }
    }

    task "exporter" { (4) driver =
      "docker"

      config { (5)
        image      = var.image
        ports      = ["http"]
        force_pull = true
      }

      service { (6) provider =
        "consul"
        name = "pearl-exporter-dev" port =
        "http"

        check {
          name      = "tcp_probe"
          type      = "tcp" interval
          = "10s" timeout =
          "1s"
        }
      }

      resources { (7)
        cpu      = 500
        memory   = 256
      }
    }
  }
}

```

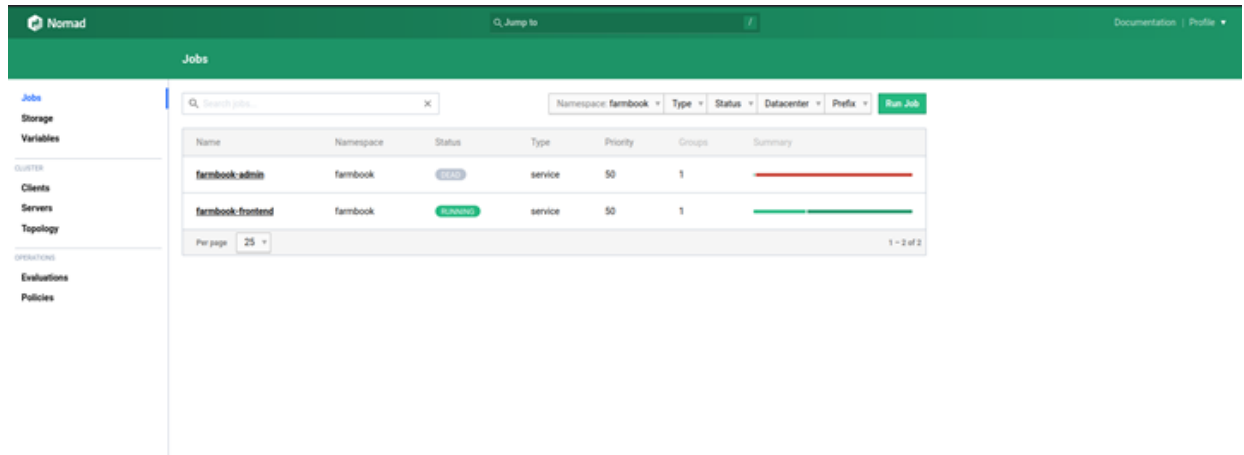
Figure 13 - An example “Job Spec” for Nomad

Some noteworthy points:

- 1 One can pass variables when submitting a job to Nomad. In this case, the *image* variable is used for dynamically setting the Docker image + tag used (e.g., after building and pushing a new image).
- 2 A *task group* can contain one or more *tasks*, which are Nomad’s main units of work. An important thing to note is that all tasks in the same group will be scheduled together on the same worker node.
- 3 The *network* stanza specifies a *port mapping* from the host to the Docker container. In this case, Nomad will dynamically allocate an unused port on the worker machine and will route all traffic on that port to port 9115 in the container.
- 4 This specifies a *task* (or workload) to be run by a Nomad worker. Nomad is a generic workload orchestrator that can run all kinds of workloads, but in our case, it will be used to run Docker containers.
- 5 The task’s *driver configuration*, which in our case will specify which Docker image etc. to use.
- 6 The *service* stanza specifies how Nomad should make this service known to Consul, which contains a catalogue of applications running on Nomad. This way, Consul

always knows on which worker node a container is running, even when it is rescheduled.

The *resources* block instructs Nomad to limit the amount of resources available to this Job.



Name	Namespace	Status	Type	Priority	Groups	Summary
farmbook:admin	farmbook	pending	service	50	1	
farmbook:frontend	farmbook	running	service	50	1	

Figure 14 - The Nomad User Interface

HashiCorp Consul

Non-technical overview

Consul is a computer program that contains a catalogue of all applications or services. Other computers or applications can determine where a certain application is running by looking at the service catalogue in Consul. This is useful when an application is distributed over multiple servers via Nomad and can be potentially running on more than one server.

Technical Overview

Consul is a service networking tool by HashiCorp. It mainly provides the following functionalities:

- Service Discovery ("Where is my application running?")
- Service Mesh ("How can I secure traffic between services?")
- Distributed Key-Value store

For Nomad, we're mainly using Consul's service discovery feature. As explained above, Nomad jobs can make themselves known to Consul's service catalogue; other services can then make use of the information in this catalogue.

Another interesting feature is Consul's Service Mesh. This makes it possible to secure traffic between applications running on Nomad via mutual TLS (mTLS) and provides an auditable trail of network traffic between applications. This feature is not yet implemented at the UGent setup, but it's certainly something we'll look at in the future.

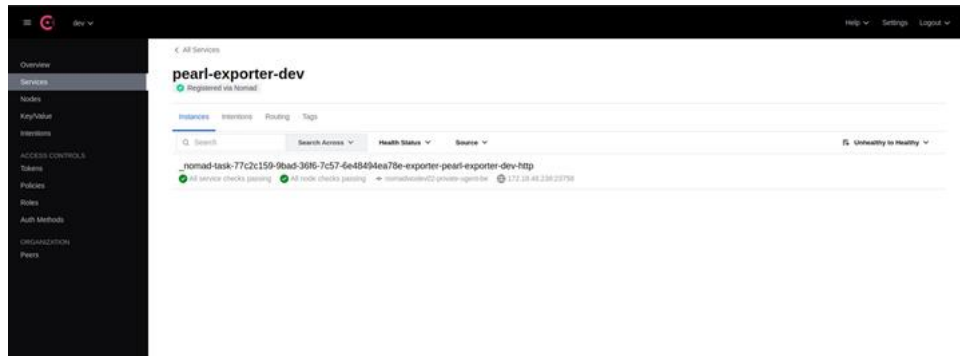


Figure 15 - The Consul User Interface

HashiCorp Vault

Non-technical Overview

Vault is a program that acts as a digital safe or vault for secrets data (e.g., application passwords). It is mainly intended to be used by applications or computers and not by humans. For example, applications running on Nomad can leverage Vault for accessing passwords in a secure way.

Technical Overview

Vault is a distributed secrets management engine that can be used for various kinds of secret data. This ranges from regular, static passwords to dynamically, on-the-fly generated credentials (e.g., database credentials).

We're using Vault at UGent for the following use cases:

- 1 Static application credentials
- 2 Nomad mTLS certificates (only API traffic for now)
- 3 Dynamic Nomad ACL Tokens for both machine and human operators
- 4 mTLS certificates for OpenSearch clusters

Both human operators and machines are able to access Vault via several authentication methods. Human operators generally authenticate via Azure AD (OpenID Connect), while machines authenticate via Puppet client certificates or AppRole.

Access to mounts is determined by ACL policies. People of the FarmBook project, for example, have access to a secrets mount for FarmBook-specific secrets and are able to request dynamic Nomad ACL tokens with Nomad ACL policies attached that allow access to the FarmBook Nomad namespace. Applications running on Nomad can be granted access to Vault secrets via their JobSpecs.

A Vault Agent daemon is running on several machines and acts as a local proxy to the Vault cluster. It is mainly used to request mTLS certificates from Vault and also renew them on a timely manner.

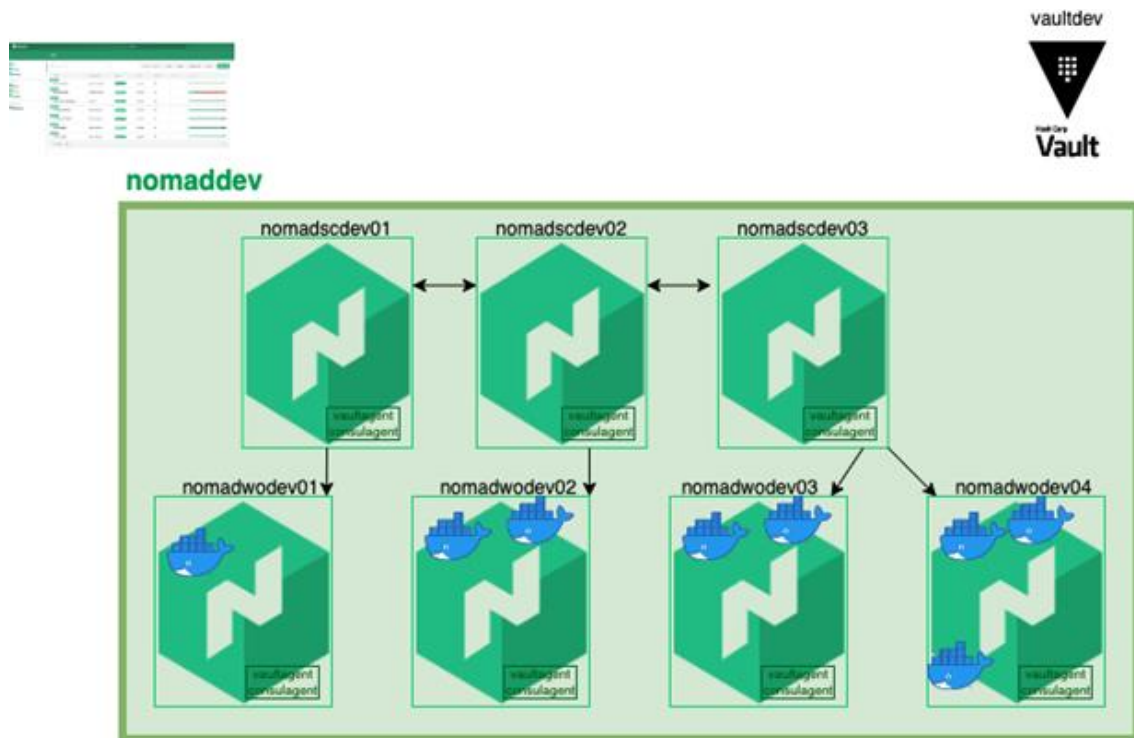


Figure 16 - An overview of Consul, Nomad and Vault



Figure 17 -The Vault User Interface

Jenkins

Non-technical Overview

Jenkins is one of our automation programs. In our case, it automates all steps that are needed to get an application running on Nomad that one would normally do manually from their laptop.

Technical Overview

Jenkins is used for Continuous Integration. It mainly automates the deployment of FarmBook applications to our Nomad platform. Deployment of an application to Nomad generally requires the following steps:

- 1 Checkout Git repository with sources
- 2 Build Docker image from source code with Git commit as image tag
- 3 Push the newly built image to our Artifactory server, which acts as a Docker registry. The image will be pushed to a specific DEV repository.
- 4 Checkout Git repository with Nomad JobSpecs
- 5 Validate and plan JobSpec with the newly built image
- 6 Apply JobSpec to Nomad DEV with the newly built image
- 7 Before continuing to PRD, Jenkins will ask for developer approval
- 8 If approved, Jenkins promotes the Docker image from the Artifactory DEV repository to the PRD repository and applies the JobSpec to Nomad PRD as well.

The above steps are also illustrated in the image below.

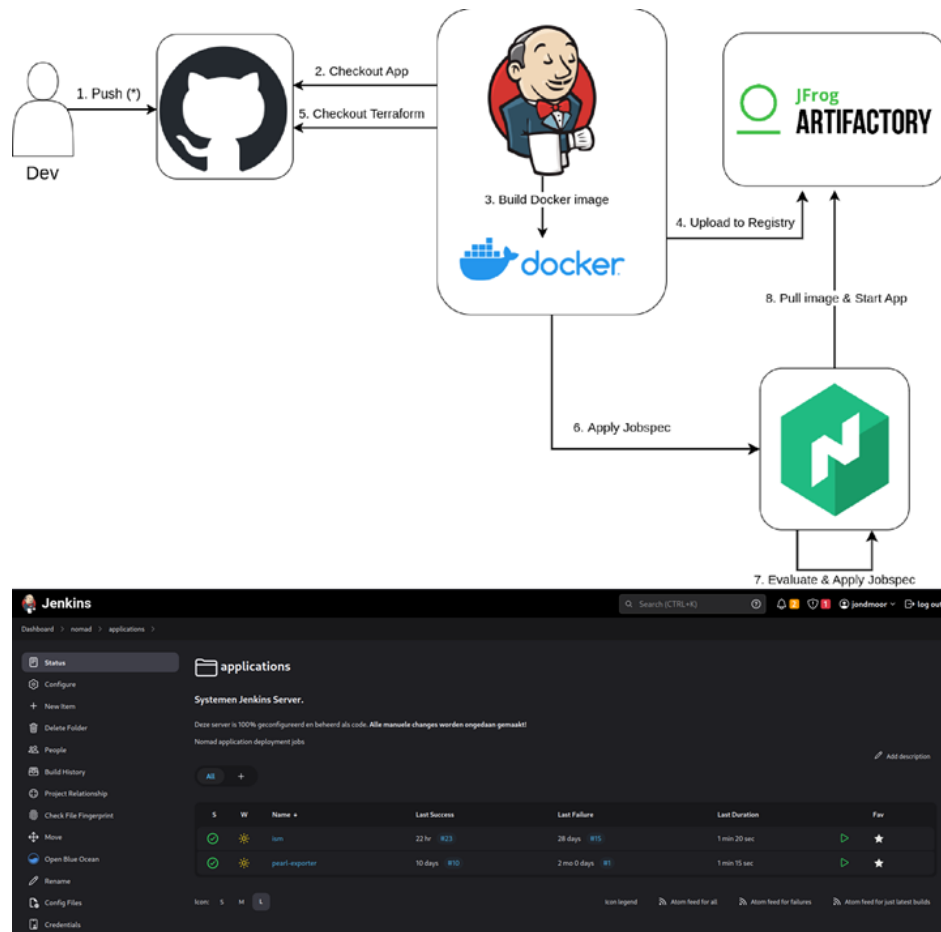


Figure 18 -The Jenkins CI/CD pipeline and User Interface

Apache Balancers

Non-technical Overview

Apache HTTPD (or Apache for short) is the software that's responsible for serving a web page when one visits a website on the Internet.

In our case, we're also using a specific functionality of Apache that allows us to distribute a request from an internet browser (e.g., Google Chrome) over multiple Nomad servers. This allows us to spread the load a bit and makes sure that applications are available as much as possible, even when a Nomad server crashes.

Technical Overview

We're using Apache HTTPD's mod_proxy_balancer functionality to load balance traffic to the Nomad API itself and to applications running on the Nomad cluster.

Since containerised applications are dynamic by nature, we combine Consul's service catalogue with Apache to correctly balance traffic. A Consul Agent daemon is running on every load balancer, which constantly monitors Consul and updates Apache's configuration when a Nomad service changes in Consul. This can happen because of various reasons, e.g. a container goes down or gets rescheduled to a different Nomad client.