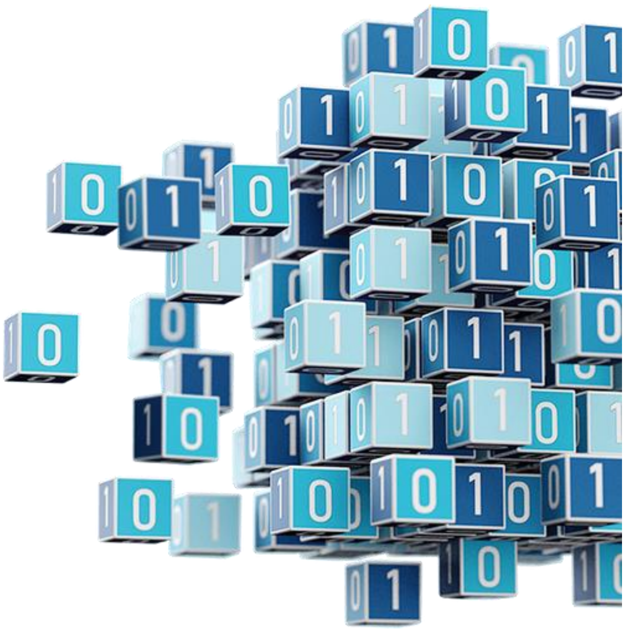
FeatureCloud

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078.

Privacy preserving federated machine learning and blockchaining for reduced cyber risks in a world of distributed healthcare



Deliverable D7.2
"App store ready and extendible by developers"

WP7
"Integrated FeatureCloud health informatics platform and app store"

Disclaimer

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078. Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

Document information

Grant Agreement Number: 826078			**Acronym: FeatureCloud**	
Full title	Privacy preserving federated machine learning and blockchaining for reduced cyber risks in a world of distributed healthcare			
Topic	Toolkit for assessing and reducing cyber risks in hospitals and care centres to protect privacy/data/infrastructures			
Funding scheme	RIA - Research and Innovation action			
Start Date	1 January 2019	**Duration**	60 months	
Project URL	https://featurecloud.eu/			
EU Project Officer	Reza RAZAVI (CNECT/H/03)			
Project Coordinator	Jan BAUMBACH, TECHNISCHE UNIVERSITAET MUENCHEN (TUM)			
Deliverable	D7.2 “App store ready and extendible by developers”			
Work Package	WP7 “Integrated FeatureCloud health informatics platform and app store			
Date of Delivery	**Contractual**	31/12/20	**Actual**	29/12/20
Nature	REPORT	**Dissemination Level**	PUBLIC	
Lead Beneficiary	01 TUM			
Responsible Author(s)	Julian Matschinske (TUM), Julian Späth (TUM), Reza Nasirigerdeh (TUM), Sándor Fejér (GND)			
	Nina Wenke, Jan Baumbach - TUM			
Keywords	Platform, AI Store, Federated Framework			

Table of Contents

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

A) Objectives of the Deliverable

Deliverable 7.2 “App store ready and extendible by developers” relates to task 1 “Programming interfaces and platform” and task 2 “App store and workflow management” of work package 7 as described in the Description of Action. It contains details on the software framework allowing for integration of methods developed in WPs 4-6, the programming interfaces FeatureCloud provides, how artificial intelligence (AI) applications can be contributed by external developers and how workflows are managed by the FeatureCloud system. Hence this deliverable contains all progress related to the FeatureCloud platform and AI store, including the overall system.

B) Executive Summary

FeatureCloud is a federated, privacy-preserving machine learning (ML) platform, aiming to simplify the development and usage of ML algorithms in collaborative settings. The main challenge in collaborative environments is that vast amounts of scattered data exist, particularly in medical facilities but privacy restrictions prevent unleashing the full potential of rapidly emerging and evolving ML algorithms. FeatureCloud overcomes this challenge by providing all software components and libraries necessary to develop and execute federated ML applications. While most existing federated ML frameworks focus on aiding during development and leave deployment to the user, FeatureCloud comes with an AI store of ready-to-use federated ML applications. These apps can be used out of the box or combined into a workflow, covering among other things pre-processing, model training, and result visualization. The AI store can be extended by the apps of external app developers, making it available for custom applications. Our experiments show that federated ML yields similar and sometimes even identical results compared to central approaches that have direct access to the entire dataset. The computational overhead is usually limited, making it a viable solution for various scenarios.

C) FeatureCloud Platform and AI Store

1 Introduction

Driven by advances in machine learning (ML) and rising privacy concerns on sharing data, techniques for collaborative machine learning have received more and more attention. Particularly in biomedicine, where vast amounts of data exist and could aid in diagnostics, understanding disease mechanisms or assessing risk factors, privacy concerns hinder even faster advances and sometimes render usage of ML impossible. Various cryptographic and algorithmic techniques such as homomorphic encryption (HE) or secure multiparty computation (SMPC) have been suggested and successfully employed to address these concerns. However, these techniques are computationally expensive and often require profound changes to the original ML algorithm. In contrast, federated ML is a comparably simple and efficient approach and therefore suited for most ML algorithms yielding comparable or even identical results while still maintaining a sufficient level of privacy. In most cases, an ML model, such as a neural network, a support vector machine (SVM) or a random forest, is trained locally at the data holders’ site and sent afterward to a central instance where the local models are combined into a global model. The general assumption is that these models do no longer contain sensitive information. In some cases where this is not necessarily true, such a naive federation can be enhanced with techniques such as differential privacy (DP) to establish the required level of privacy, usually at the expense of the global model’s quality.

2 Design Considerations

This section contains fundamental considerations for the FeatureCloud platform. It is divided into a methodology section laying out current approaches and available solutions in the area of federated machine learning, and a section about design choices made in FeatureCloud.

2.1 Methodology

Privacy-aware machine learning has received much attention due to the ever growing amounts of biomedical data. Different techniques and approaches exist, which are described in subsection 2.1.1, and several frameworks and platforms implementing these approaches have emerged, which are described and compared in section 2.1.2.

2.1.1 Privacy-aware ML

In recent decades, machine learning techniques have been successfully applied to various fields, including healthcare. However, studies have shown that ML models trained without any privacy consideration are vulnerable to potential privacy attacks, such as membership inference attacks. Besides individual concerns about sensitive data in healthcare, also privacy regulations, such as the European GDPR, request a higher awareness of privacy considerations in machine learning.

The efforts in making ML models privacy-preserving can be categorized into four groups based on the method they employ: (1) federated learning (FL), (2) cryptographic techniques (including HE and SMPC), (3) differential privacy (DP) and (4) hybrid approaches. Each of these categories has its strengths and weaknesses in terms of computational and communication efficiency, utility and privacy guarantee. For example, FL suffers from high communication cost compared to HE and SMPC. However, as FL is based on the “moving computation to data” methodology rather than “moving data to computation”, it is computationally more efficient than HE and SMPC. As another example, an FL model does not provide a privacy guarantee while a differentially private ML model does so (namely epsilon and delta). On the other hand, FL is a more utility-aware technique than DP as it does not inject any noise perturbation to the data or the training process [5].

Considering the advantage and disadvantages of the privacy-aware ML models as well as the curse of dimensionality and importance of achieving high utility in healthcare settings, FL and the hybrid approaches that are based on FL (FL + HE [1,2],FL + DP [3–7] and FL + DP + HE [8–10]) seem to be the most promising and practical privacy-aware ML methods for healthcare applications. By using FL the health institutions (or the clients in general) can collaborate in training a common ML model while ensuring that the individuals’ private data will not move out of their local sites (even in encrypted form). Moreover, if there is a case in which enhanced privacy is required, they can also privatize the FL model parameters using DP or other types of obfuscating techniques.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

2.1.2 Frameworks and Platforms

Several frameworks for FL have been developed in recent years (Table 1). We identified two different approaches: backend-only and all-in-one approaches. On the one hand, backend frameworks, such as Tensorflow Federated and PySyft, mostly focus on deep learning algorithms and provide developers with methods to simplify the implementation of federated and privacy-aware machine learning analysis.

On the other hand, all-in-one frameworks try to bring privacy-aware analyses to users without developer knowledge. Additionally to a backend framework, they also provide a frontend, enabling non-developer users to perform federated and privacy-aware analyses. Most of these frameworks are tailored for specific areas: NVIDIA Clara is a federated learning framework for imaging and genomics, focusing on an easy deployment in medical infrastructures. COINSTAC [11] is a platform for decentralized brain imaging analyses that is available for all major platforms. Melloddy aims to connect the decentralized data of 10 pharmaceutical companies and enable privacy-preserving analyses for drug discovery.

Furthermore, few platforms such as Owkin or Acuratio enable federated learning on non-specific datasets. Acuratio offers various horizontal and vertical federated learning algorithms that can be executed on the platform. Owkin has a strong medical focus and connects data by therapeutic area. For now, with around 70 models (December 18, 2020), the Owkin platform seems to be the largest federated learning platform in medical environments. In contrast to federated approaches, decentralized approaches do not contain a central aggregation instance. As an example, the Personal Health Train [12] (PHT) connects different data centers and allows scientists to create value from the data of different sources. A “train” (model) goes from one “station” (data source) to another and trains with the respective data until the training has finished. It has already successfully learned a distributed model on more than 20.000 lung cancer patients [11,13].

As a matter of fact, each of the frameworks brings its benefits and can be useful in specific scenarios. However, most of them are not suited for application in clinical environments. Deep learning (DL) models have already been useful in healthcare and clinics [14] but are usually not interpretable. The interpretability of models is a massive concern in clinical research to justify medical decisions, and therefore, a focus only on DL models is not applicable in a clinical context. Backend-only approaches in clinical environments are limited to users with a developer background and coding experience. This background can usually not be expected by clinical experts without a statistics department and, therefore, restricts usability massively.

All-in-one platforms either have a strong focus on one specific field (COINSTAC for brain imaging, Melloddy for drug development) or still require a lot of technical knowledge and developers for the set up (NVIDIA Clara, PHT). Unfortunately, the demos of the federated frameworks DiscreetAI and Fed-BioMed [15] could not be executed and also the sign up did not work. This shows that some platforms are still at a very early stage or do not focus on robustness or convenient development. For now, the probably most significant player of the all-in-one platforms in the medical sector is Owkin. While it offers many models and connects therapeutic areas, it is not open-source, and therefore, its algorithms remain a black box for the users. Also, it is not instantly and freely available which makes it difficult if not impossible to use for certain scientific target groups.

Reviewing the players in federated learning frameworks and platforms, it is striking that there is no convenient open-source solution yet that combines two crucial tasks in bringing federated learning to a medical environment: model training and model development. While some platforms certainly try to ease the training of federated learning models for the end-user, models are either not collected to be shared within the framework or the API is not open to developers and end-users can only use predefined models. This restriction is a huge disadvantage in the area of AI, which is rapidly evolving and algorithms get published or enhanced very frequently.

Name	**Framework Type**	**Open Source**	**Algorithms**	**Privacy Technique**
Tensorflow Federated	Backend	yes	DL	FL
PySyft	Backend	yes	DL	FL, DP, MPC, HE
Flower	Backend	yes	DL	FL
PaddleFL	Backend	yes	DL	FL, MPC, DP
CrypTen	Backend	yes	DL	MPC
XayNet	Backend	yes	DL	FL, HE
FATE	Backend	yes	Various	HE, MPC, FL
NVIDIA Clara	All-in-one	yes	Various	FL
DiscreetAI	All-in-one	yes	Various	FL
Fed-BioMed	All-in-one	yes	Various	FL
PHT	All-in-one	yes	Various	FL
Owkin	All-in-one	no	Various	FL, DP
COINSTAC	All-in-one	yes	Various	FL, DP
Melloddy	All-in-one	yes	Various	FL
Acuratio	All-in-one	no	Various	FL, DP
FeatureCloud	All-in-one	planned	Various	FL

Table 1. ***Privacy-Preserving Machine Learning Frameworks and Platforms.***

2.2 Federated ML in FeatureCloud

Federated ML broadly involves two general operations, possibly alternating in multiple iterations: local optimization and global aggregation. In FeatureCloud, all running instances of a federated application (app) take one of two roles: participant and coordinator, performing the respective federated operation. FeatureCloud expects precisely one coordinator and an arbitrary number of participants, leading to a star-based architecture (see fig. 1).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

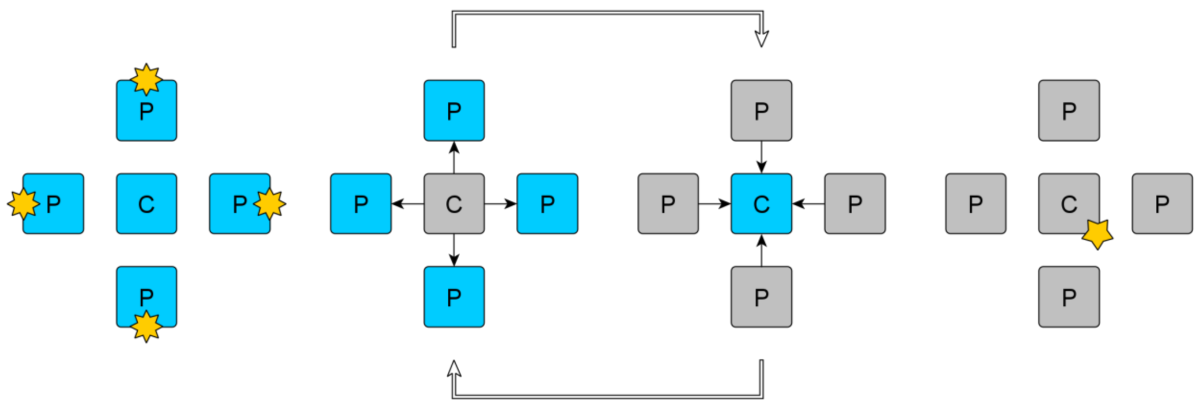


Figure 1. ***Four stages of federated execution in FeatureCloud.*** *The four main stages are 1) local data loading, 2) broadcasting a global model, 3) gathering local models, 4) compiling results. Stage 2 and 3 can be repeated depending on the executed algorithm. ‘C’ and ‘P’ stand for coordinator and participant, respectively. The yellow stars in stage 1 and 4 represent local training data and global parameters, respectively.*

After a local optimization operation has been completed by a participant, it sends the local parameters to the coordinator. The coordinator collects these parameters and aggregates them into a common (global) model, which is shared with the participants. Depending on the type of ML algorithm, these two operations can alternate a couple of times, e.g. until convergence or a pre-defined number of iteration has been reached. For some algorithms (e.g. random forest, linear regression), only one iteration is necessary. However, this strict separation between optimization and aggregation is not actively enforced by FeatureCloud. In many cases, aggregation can already start after the first parameters have been received, thereby increasing efficiency through parallelization of the computation. Figure 1 shows the logical roles of coordinator and participant, however in practice the coordinator usually has local data as well. Therefore, FeatureCloud also allows the coordinator to additionally assume the logical role of a participant.

3 Platform and AI Store

FeatureCloud’s primary goal is to simplify the development and usage of federated ML algorithms. This involves three major challenges: development of apps, distribution of apps, and usage of apps. Each of these challenges is tackled by FeatureCloud and described in the following subsections.

3.1 App Development

Since FeatureCloud does not impose restrictions on the kinds of algorithms it supports, the running environment of the federated apps is kept very general. It allows for implementing any type of ML algorithm and an optional custom graphical user interface (GUI) for user interaction, also referred to as app frontend.

From a technical point of view, a FeatureCloud app acts as a web server (see section 4.2.1), responding to requests sent from the FeatureCloud system or the app frontend. No direct Internet access is granted to apps, which would pose a security risk (see fig. 2). A complete overview of the architecture can be found in section 4.2.

3.1.1 Isolation of Apps

System access of apps should be as limited as possible to rule out attack vectors, such as leakage of sensitive data or access of data, which should not be included in an ML algorithm (e.g. due to lack of consent). FeatureCloud uses Docker as a virtualization technique. Docker has been widely adopted in the developer community, particularly in the area of web development. It is available for all major operating systems (Linux, Microsoft Windows, macOS), making FeatureCloud nearly platform-independent. Docker also offers the necessary level of isolation, preventing Internet and file access if not explicitly granted, and sandboxing to limit the usage of compute and memory resources if necessary. These isolated running environments (containers) are created from pre-defined images, which are the federated apps in our case.

3.1.2 Data Loading and Sharing of Results

ML apps need access to training data to optimize their models. As depicted in fig. 2, apps cannot access sensitive data directly. Instead, the app user needs to provide the data to the FeatureCloud system, making it available to an app. From an app perspective, the data can be expected to reside inside a dedicated input directory mounted to the app container. Analogously, all results generated by an app must be put inside an output directory, which is provided by FeatureCloud as well, whose contents can be downloaded via the FeatureCloud user interface.

This output data can also be picked up by another app that is executed successively and finds the output data of the previously ran app inside its input folder. Chaining these apps is a feature that allows the composition of multiple apps into a workflow, a concept that is further described in section 3.3.

3.1.3 FeatureCloud Interface for Apps

The FeatureCloud controller (fig. 2) regularly asks a running app whether it has new model parameters to share with the other members of the federation. If this is the case, it loads them from the app and hands it over to the other apps, depending on whether it is a participant or a coordinator, as described in section 2. If the global model parameters need to be shared with the app, the controller actively sends them to the app.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

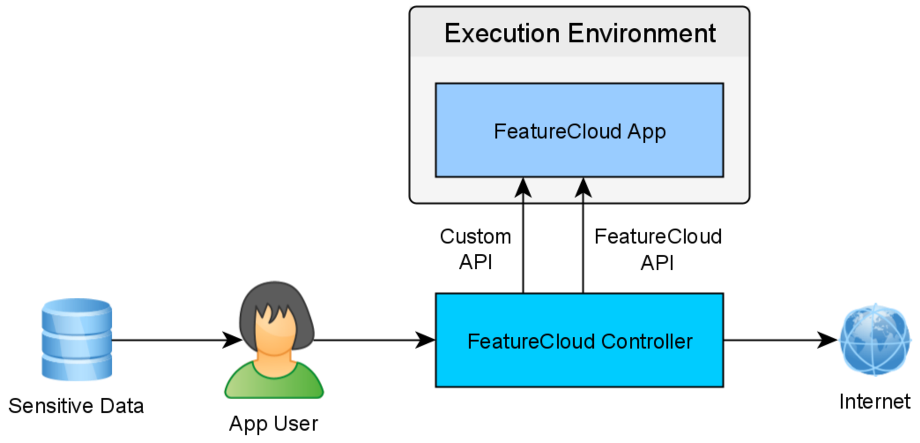


Figure 2: ***Execution environment for FeatureCloud apps.*** *App users decide what data to load into the system. FeatureCloud apps cannot directly access the file system or the internet.*

The API is based on the TCP/IP-based HTTP protocol, which is asynchronous by its nature. In web terms, the FeatureCloud app acts as a web server and the FeatureCloud controller acts as a web client. The full specification of the API is included in the supplement, section 2.3.

An app can also provide its custom user interface to allow for monitoring the computation or for interaction with the user. To this end, additional endpoints need to be defined, which can then be accessed from the app user’s browser. Web technologies are used for GUI design, i.e. HTML/CSS/JavaScript. Custom endpoints cannot be accessed directly due to technical and security reasons. In a typical scenario, the FeatureCloud controller runs on a central server inside a data holder’s local network and users access the frontend from a different machine inside the network. To make this possible, the FeatureCloud controller listens on only one port, which can e.g. be accessed through an SSH tunnel, redirecting all app-targeted traffic to the correct container.

3.1.4 Testing and Debugging

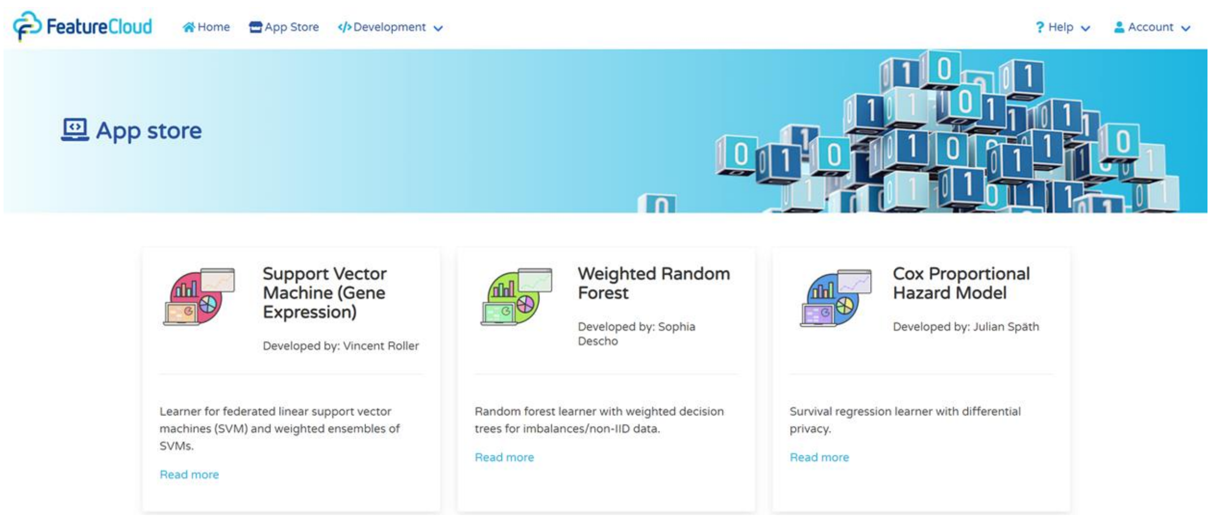
The development of algorithms involves intensive testing and debugging. For rapid development, it is crucial that these testing and debugging cycles are as quick as possible. Therefore, FeatureCloud comes with a local test framework that enables app developers to instantly run their application on their machine without deploying it first. When using this functionality, one has to specify the number of participants, i.e. app instances to simulate, and a data directory for each instance containing the respective input data. When started, the FeatureCloud controller creates one container for each instance and connects them logically identically on the developer’s machine to a truly federated setup on different machines.

3.1.5 App Templates

The API has deliberately been designed in an algorithm and usage agnostic way. This leads to high flexibility but requires the app developer to implement all algorithm-specific functionality by themselves. To quickly introduce developers to the API and provide a convenient starting point for app development, FeatureCloud comes with a collection of easily extendable templates. This collection includes a minimal template with a demo Python/Flask implementation, stubs for all API calls and a blank demo frontend, and a federated mean app. In section 6, we describe how to extend the API in the future so that less functionality needs to be implemented inside the apps and more functionality is provided by the FeatureCloud system itself.

3.2 AI Store

FeatureCloud presents all implemented apps in an easily searchable AI store to make federated ML available for as many users as possible. Conversely, app developers need a simple way to share their apps and attach important information, such as the required input data format, the format of the produced output, usage instructions, and privacy considerations.



Support Vector Machine (Gene Expression)
Developed by: Vincent Roller
Learner for federated linear support vector machines (SVM) and weighted ensembles of SVMs.
Read more

Weighted Random Forest
Developed by: Sophia Descho
Random forest learner with weighted decision trees for imbalances/non-IID data.
Read more

Cox Proportional Hazard Model
Developed by: Julian Spath
Survival regression learner with differential privacy.
Read more

Figure 3: ***FeatureCloud AI store.*** *Users can select from a variety of ready-to-use apps.*

3.2.1 Pushing New Apps and App Updates

As described in section 3.1.1, all apps are stored as Docker images. Conventionally, docker images are shared using a Docker registry, to which new or updated images can be pushed and existing images can be pulled. FeatureCloud uses a standard Docker registry and controls its access through a proxy server (see fig. 4).

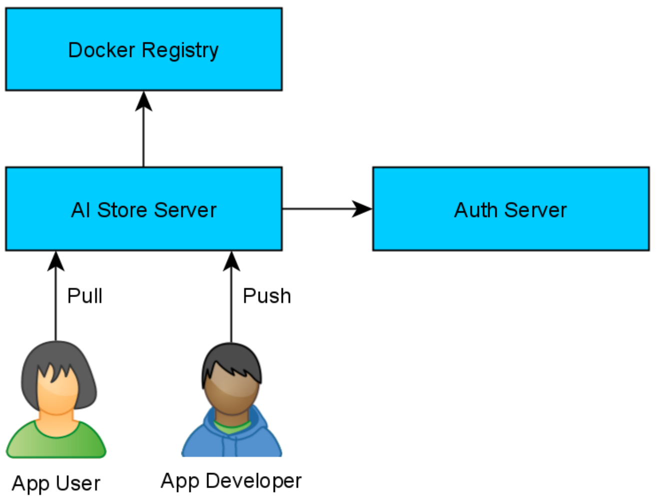


Figure 4: ***Users and developers access the Docker registry through an AI Store Server.*** *App users can only pull, app developers can also push new images.*

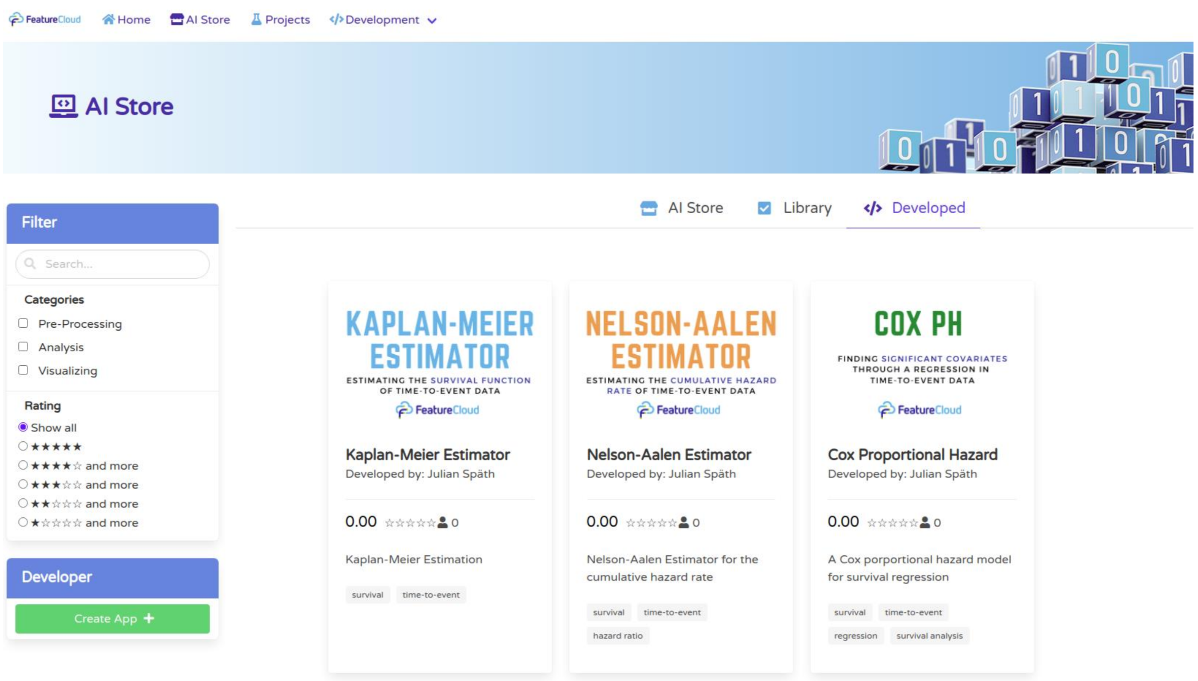
Sharing the app after implementation on the FeatureCloud AI store involves the FeatureCloud website and usage of the Docker CLI (see supplement, section 2). When a new image is pushed to the registry, the AI Store Server assures that the developer has the required permissions to push the respective image by connecting to the Auth Server (see fig. 4). If it is successfully authenticated, the image is uploaded to the FeatureCloud Docker registry and becomes available to other users.

Pulling the images is done automatically when the workflow starts running (see section 3.3.2 and supplement, section 3). The only technical requirement is a Docker installation on the user end. All the other steps are performed without user interaction.

3.2.2 Publishing Apps in the AI Store

After the app image is available in the FeatureCloud App Registry, developers can publish their app in the AI Store. As a first step, a FeatureCloud account needs to be created and verified by a confirmation link sent to the corresponding email address. As soon as the user activates the developer mode in the profile settings, the AI store (see fig. 5) provides an additional tab “Developed” which will list all of the user’s developed apps. Furthermore, a developer sidebar, including a “Create App” button, becomes visible.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.



Kaplan-Meier Estimator
Developed by: Julian Späth
0.00 ☆☆☆☆☆ 0
Kaplan-Meier Estimation

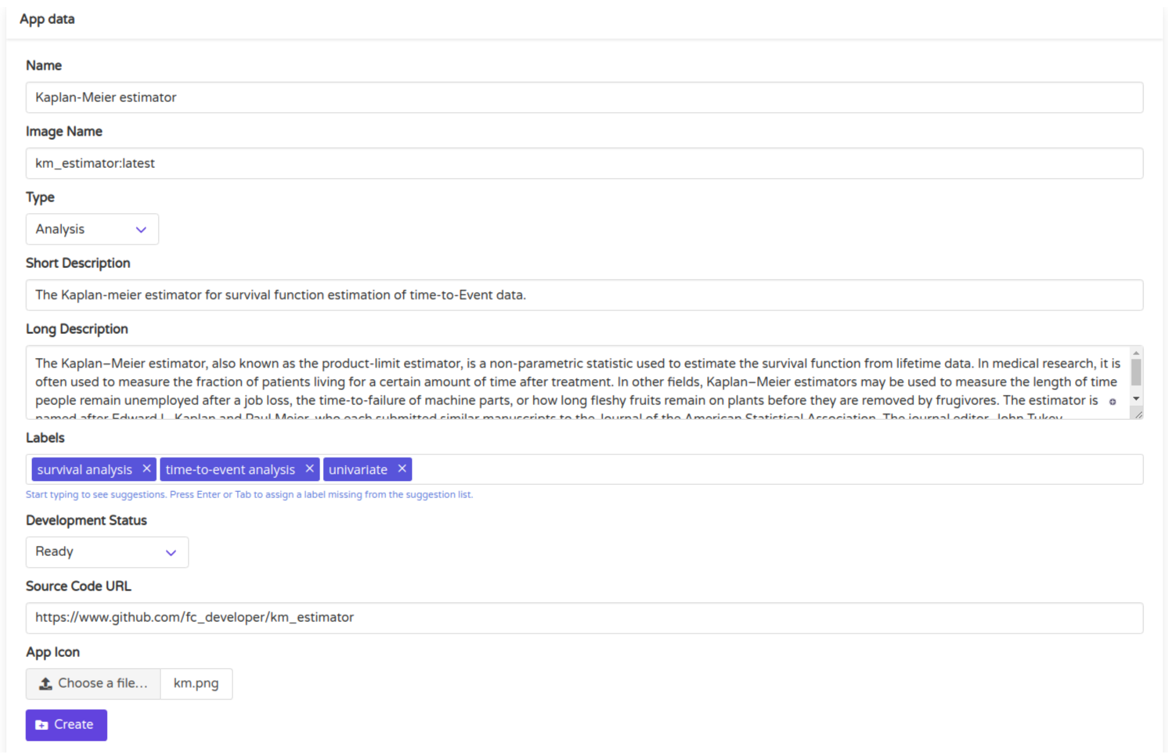
Nelson-Aalen Estimator
Developed by: Julian Späth
0.00 ☆☆☆☆☆ 0
Nelson-Aalen Estimator for the cumulative hazard rate

Cox Proportional Hazard
Developed by: Julian Späth
0.00 ☆☆☆☆☆ 0
A Cox porportional hazard model for survival regression

*Figure 5: **AI Store for developers.** Developers can see the apps they already published in the "Developed" tab.*

Clicking the "Create App" button forwards the user to an input form (see fig. 6) that is used to describe the information about the app. An app consists of a name, short and large description, an icon, and one or multiple labels that can be used as search tags. Furthermore, an app type needs to be selected that is either "Preprocessing", "Analysis", or "Visualization". The privacy technique defines what methods are used in the app to preserve privacy. For now, this can be "Federated Learning", "Differential Privacy", "Secure Multi-Party Computation", "Homomorphic Encryption", or combinations of them. Finally, the image name needs to be defined to connect it to the corresponding app in the FeatureCloud App Registry. The app information can always be updated by the app author at a later point.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078.


App data

Name

Kaplan-Meier estimator

Image Name

km_estimator:latest

Type

Analysis

Short Description

The Kaplan-meier estimator for survival function estimation of time-to-Event data.

Long Description

The Kaplan–Meier estimator, also known as the product-limit estimator, is a non-parametric statistic used to estimate the survival function from lifetime data. In medical research, it is often used to measure the fraction of patients living for a certain amount of time after treatment. In other fields, Kaplan–Meier estimators may be used to measure the length of time people remain unemployed after a job loss, the time-to-failure of machine parts, or how long fleshy fruits remain on plants before they are removed by frugivores. The estimator is named after Edward L. Kaplan and Paul Meier, who each submitted similar manuscripts to the Journal of the American Statistical Association. The journal editor, John Tukey,

Labels

survival analysis ✕ time-to-event analysis ✕ univariate ✕

Start typing to see suggestions. Press Enter or Tab to assign a label missing from the suggestion list.

Development Status

Ready

Source Code URL

https://www.github.com/fc_developer/km_estimator

App Icon

Choose a file... km.png

Create

*Figure 6: **Publishing an app.** Developers can publish an app by defining the app information and link it to a Docker image in the FeatureCloud App Registry.*

3.2.3 Using Apps and Providing Feedback

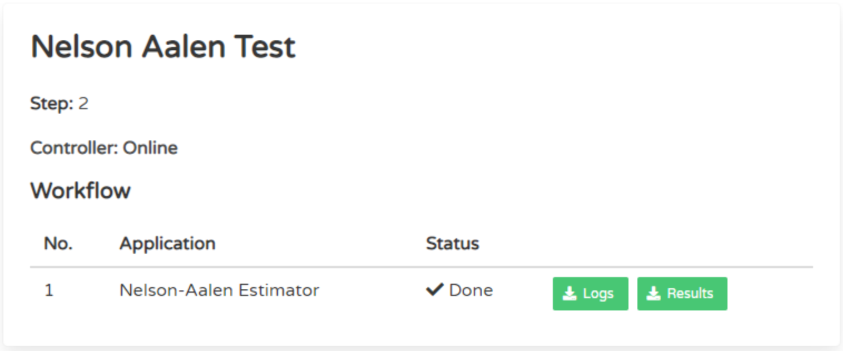
Users can search the AI store using full-text search or by choosing an app category or tag (see fig. 5). Apps that have been reviewed by FeatureCloud are marked as such and shown by default. Other apps are only shown if the user explicitly accepts unsafe apps. Before using an app, users need to add it to their personal library of apps. This serves as bookmarking and allows for adding an extra licensing step in the future. Once added to their library, users can include an app in their workflow and provide the developer with feedback, i.e. a star-based rating and a clarifying comment.

3.2.4 App Certification

Allowing third-party developers to quickly and easily push apps to the app store and use them for collaborative studies is one of FeatureCloud’s selling points. However, it is difficult to automatically ensure privacy awareness of such apps (see deliverable 2.2, KPI ‘Privacy Requirements’). Therefore, FeatureCloud distinguishes between two types of apps: 1) certified ones and 2) uncertified ones. By default, the app store only displays apps that have been certified by a privacy expert of the FeatureCloud consortium. The user needs to actively choose to display uncertified apps and is warned and informed about the risks. In general, users are advised to only use uncertified apps from a source they trust, e.g. a collaboration partner they already work together with.

If developers want their apps to be certified, the source code needs to be completely accessible to the responsible FeatureCloud member. After the code has been reviewed and deemed secure, the Docker image is built by the consortium member and pushed to the app store via the Docker CLI. The FeatureCloud system recognizes the author of the image as a trusted party and marks the corresponding image version, identified by its SHA256 digest, as certified. After a successful certification, third-party developers can still push new uncertified versions to update the app or fix bugs. However, each update of the app, and respectively each new version, needs to be certified again to make sure that the update does not raise any privacy issues. In the meantime, all users who added a particular version of an app from the app store to their personal library of apps will automatically keep this version in their library. At this point, if a user wants to update their app to a new (certified) version, they need to remove it and add it again. This process will be simplified in future AI store releases and replaced with a convenient update functionality as it is already established for mobile phone app updates.

3.3 Workflow Management and Execution


Nelson Aalen Test

Step: 2

Controller: Online

Workflow

No.	Application	Status	
1	Nelson-Aalen Estimator	✓ Done	Logs Results

Figure 7: ***Workflow has finished successfully.*** *A workflow which has successfully completed offers its results as download in the FeatureCloud frontend. Logs can also be downloaded for debugging purposes.*

To run a study with other collaborators, a project needs to be created in the FeatureCloud frontend first. Projects consist of a name and a brief description to provide information for invited collaborators. Additionally, they contain a workflow, defining which apps will be executed in which order. The creation of a project is only possible for FeatureCloud users assigned to a data holder site (see section 4.2.3, fig. 12). When they create a project, they act on behalf of their site. In practice, users typically are medical doctors or academic researchers who administer a FeatureCloud project, and sites are medical facilities or academic institutions.

The following two subsections describe how a workflow can be composed from a user (coordinator) perspective and how the consecutive execution is performed from a technical perspective.

3.3.1 Workflow Composition and Invitation

All apps that should be part of the workflow need to be added from the user’s library of apps (see fig. 5). However, it is not required that all participants later have the apps in their library. After all apps have been added, the project is finalized and becomes immutable. To invite other

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

collaborators, tokens (i.e. large random strings) need to be shared with them. Tokens are uniquely linked to a project and allow for joining the project. They can only be used once for security reasons and need to be entered in the FeatureCloud frontend. Once all participants have joined, the coordinator can take a final look and start the project. From that moment on, no one can join anymore and the execution begins.

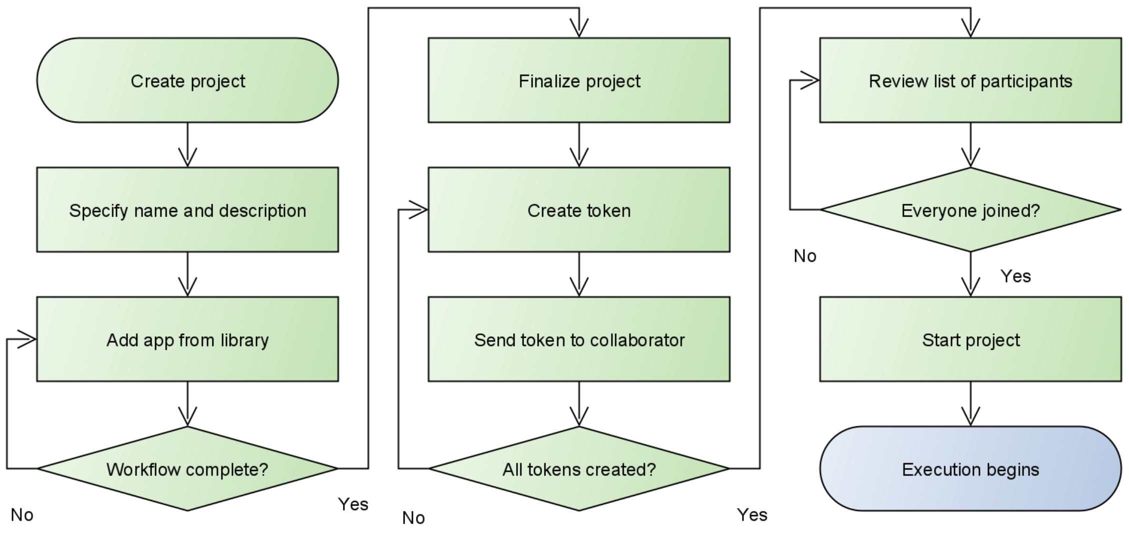


Figure 8: ***Process of composing a project, inviting participants and starting a project.*** *Green symbolizes human interaction, blue symbolizes automatic behaviour.*

3.3.2 Execution of a Workflow

Once the coordinator has triggered the execution, the FeatureCloud controller creates the input volume for the first app in the workflow at each participating site. This volume needs to be provided with the actual data relevant to the study, which is processed by the workflow. The users need to select the data via the FeatureCloud frontend, which is then sent to their local controller, importantly not leaving the data holder’s domain. As described in section 3.1.2, each app has an input and output directory, which serves as a file-based data interface to FeatureCloud.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

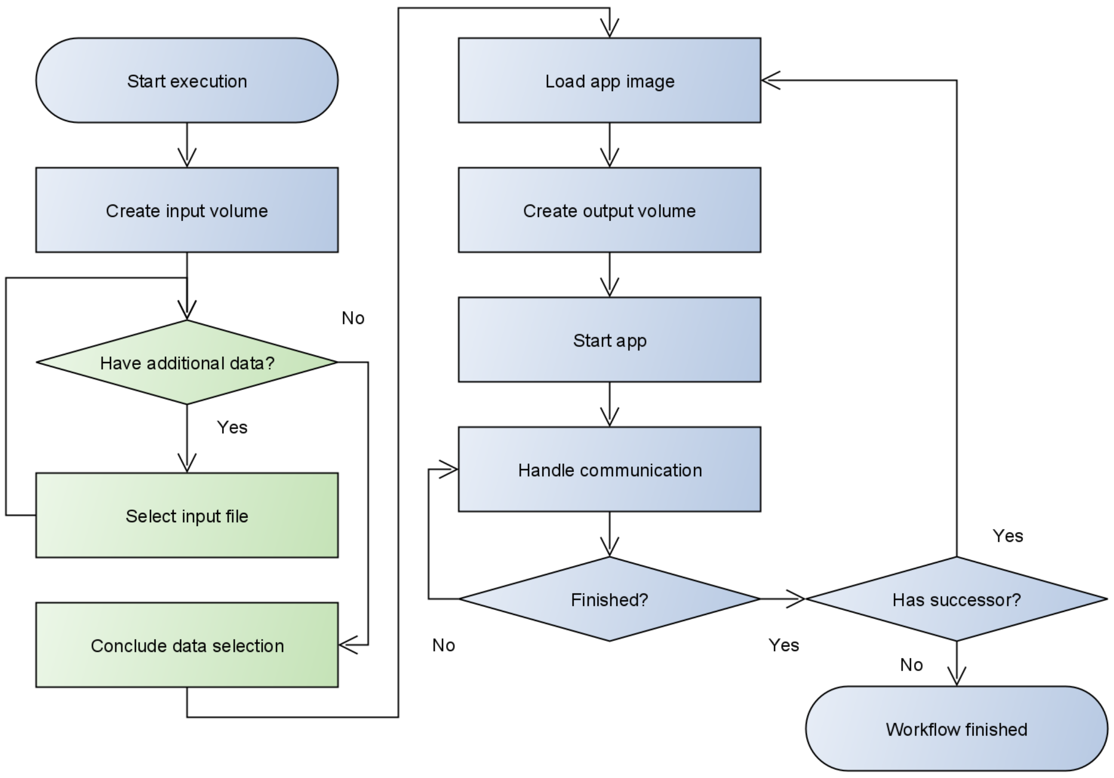


*Figure 9: **Workflow execution managed by the FeatureCloud system.** Green symbolizes human interaction, blue symbolizes automatic behaviour.*

After each participant has selected their input data, the first app is started as a docker container. The current progress of the workflow can be monitored in the frontend, showing the currently executed step (i.e. app) and providing its container logs if required (see fig. 7). In general, no user interaction is necessary from this point on unless an app in the workflow actively requires so through its custom frontend. The app frontends can be accessed from the workflow page as well, usually to monitor app-specific events or view visualizations provided by the app. When the computation of a step has been completed, indicated by the coordinator app instance, all containers of this step are shut down and the contents of the output directory are placed inside the input directory of the next app in the workflow. These intermediate results can also be downloaded from the frontend for later investigation or detecting potential errors in the analysis. All debugging output produced by apps is stored in a directory on the controller machine, to investigate errors that might occur during execution.

A workflow can thus be regarded as a processing pipeline, composed of apps provided by FeatureCloud, enabling an additional level of customizability. For a selection of the currently available apps, see section 5. For a complete workflow sequence diagram, see supplement, section 3.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

4 Architecture and Implementation

In this section, the technical details of the FeatureCloud system are described. It is split into an overview of the system architecture, i.e. the high-level constellation of the system components and their tasks, and the respective software architectures and details on behaviour and applied technologies of these components.

4.1 System Architecture

The FeatureCloud architecture consists of the following system components (see fig. 10): local Controller, relay server, global backend, frontend, and AI store server.

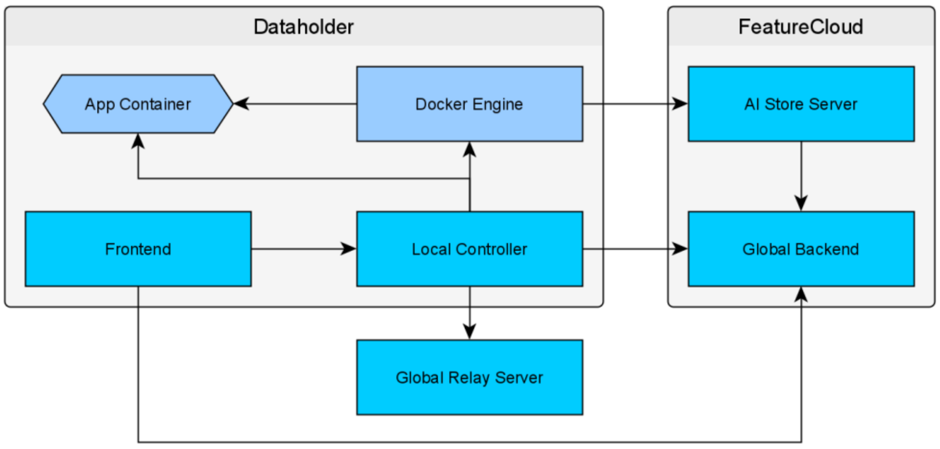


Figure 10: ***Interactions between the FeatureCloud system components.*** *Frontend and local controller are at the data holder's site, AI store server and global backend run on FeatureCloud servers.*

On the data holder's site, the controller and frontend web application are running. On the FeatureCloud servers, the AI store server, including a Docker registry, and the global backend, are running. Optionally, a global relay server is provided by FeatureCloud as well, in case setting up a custom relay server is not required or not possible.

The controller orchestrates app execution by instructing the Docker engine to create or shut down app containers, create and mount input and output volumes, and expose the required ports for the FeatureCloud API. It also serves as a proxy between the frontend and the app containers to decouple containers from the frontend.

The frontend is used to access the controller and manage the FeatureCloud account, federated apps, and projects, which involves the global backend.

The relay server acts as a communication hub for all participants of a workflow. Since it relays model parameters and has access to this data, users might want to use their own relay server instead of using the one provided by FeatureCloud.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078.

The AI store server is used to host app images and is described in section 4.2.5 in more detail.

The global backend stores all user information, information about data holders, apps, projects and workflows, and is involved during workflow execution by saving the current step and progress. However, it never has access to any raw data or traffic between apps participating in a workflow, which is one of the crucial properties of FeatureCloud.

4.2 Implementation

This section contains information about technology, software architecture and implementation details for each of the integral FeatureCloud system components.

4.2.1 Local Controller

The local controller needs to be able to handle large amounts of data and asynchronous tasks as well as keep up multiple socket connections and support HTTP-based and raw byte traffic. For this reason, this component has been implemented in Go (aka Golang), a native programming language developed for server applications. It allows for lightweight co-routines to monitor app containers and regularly query for updates from the global backend.

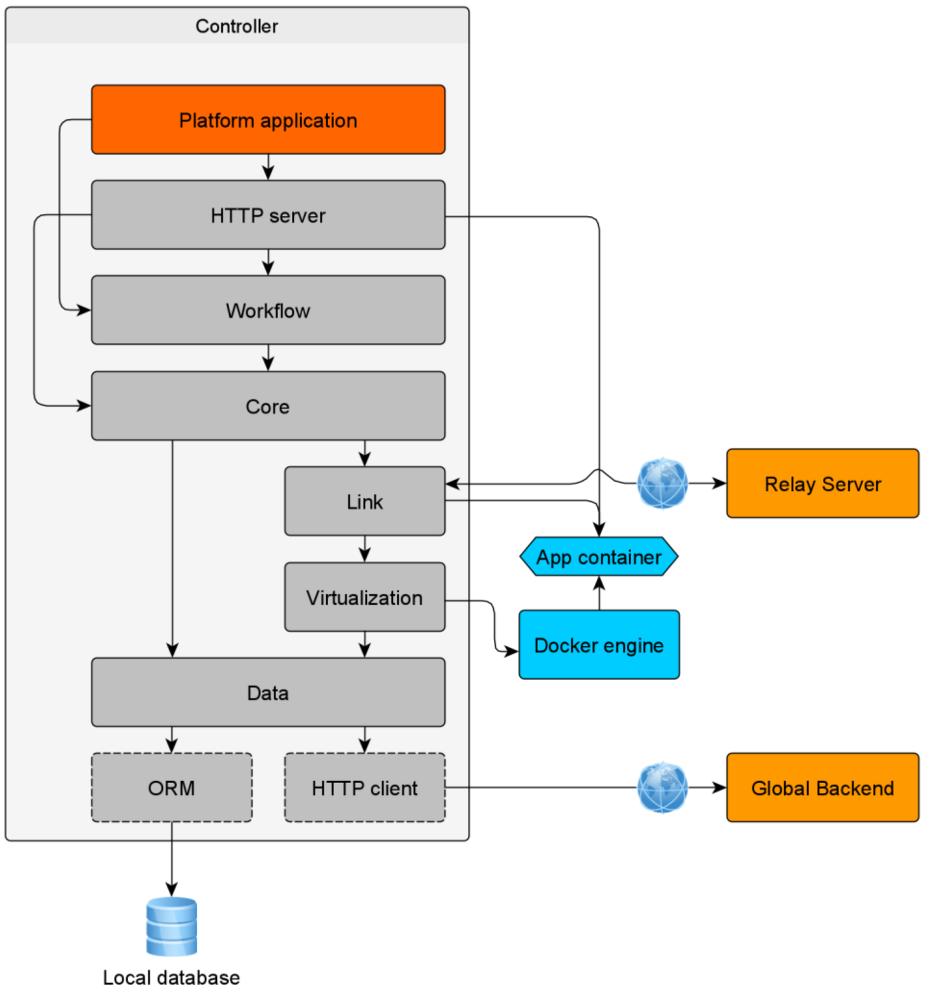


Figure 11: ***Software architecture of the local controller.*** *It uses a layered architecture preventing arbitrary access across layers by enforcing a partially ordered access hierarchy.*

The software architecture has a layered structure, with a decreasing level of abstraction from top to bottom (see fig. 11).

The platform application layer is the main entrypoint responsible for reading configuration values (e.g. local database credentials, address of the global backend) and starting an HTTP server and polling routines. The HTTP server provides endpoints for the frontend to control workflow-related tasks, such as loading data into the first input volume, show container logs. It also relays traffic to the app-specific frontends. The workflow layer offers abstract functions for the HTTP server and takes care of workflow management, such as setting up and attaching volumes, starting containers, shutting them down, reacting to updates from the global backend (by using the data layer through the core layer). The core layer provides an abstraction of the core business logic, especially app container management and functions for testing apps during development. The link layer handles communication between app containers and the relay server, translating raw byte-traffic from the relay server to HTTP-based traffic for the containers and vice versa. The controller acts as an HTTP client in this case, and the app containers as HTTP servers. This way, active access by the app containers to the Internet can be avoided. The virtualization layer is a direct abstraction of Docker, which allows for replacing the virtualization technique in the future if needed for security or compatibility reasons.

4.2.2 Relay Server

The relay server implements basic relay functionality for star-based federations of clients. It knows the role of each client (i.e. participant or coordinator) and treats their traffic accordingly. If data is received from a participant, it relays it to the coordinator. If it is received from the coordinator, it is broadcast to all clients. A relay server can handle multiple workflows at once. For that, it uses workflow-specific credentials chosen by the coordinator and automatically distributed to the participants by the global API. Like the controller, it is written in Go since it needs to efficiently handle large amounts of binary data, which Go is capable of.

4.2.3 Global Backend

The global backend mainly offers an HTTP API for controllers and the frontends. It is responsible for managing all necessary data related to projects, apps, users and data holders (sites). It is implemented in Django, a Python web framework that offers the functionality for this kind of task, particularly database abstraction, URL routing and web-related utilities (e.g. JSON serialization, HTTP abstraction).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

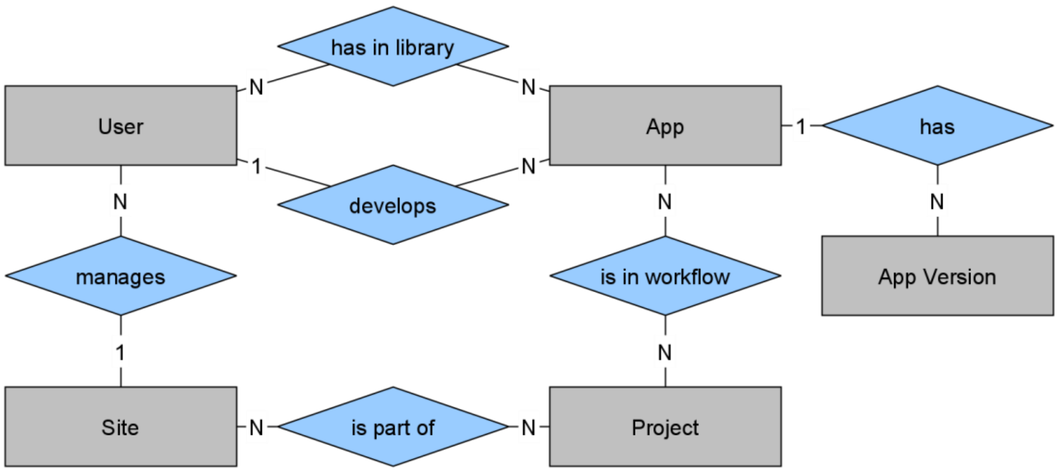


*Figure 12: **E/R diagram of the data model in the backend.** Grey boxes represent entities, blue diamonds represent relationships.*

The E/R diagram of the data model is shown in figure 12. The global backend allows controlled access to instances of these entities.

User. Users have an email address and a hashed and salted password allowing them to log in to the FeatureCloud frontend, which then queries the global backend. In practice, a user is either a developer who has apps linked to them through the ‘develops’ relation, or an end user. Both, developers and end users, can add apps to their library (relation ‘has in library’) and manage a site (relation ‘manages’).

Site. Sites have necessary contact information and represent a data holder location, e.g. a hospital or academic research institution. Each site needs to run a controller instance (see section 4.2.1) to participate in projects (relation ‘is part of’). When a site is part of a project, it can either assume the role of the coordinator or a participant.

Project. Projects encompass a workflow, descriptive information and a set of tokens allowing for joining a project (see section 3.3.1). Tokens are not modeled explicitly. Instead, the ‘is part of’ table is used, which can have entries with a token string and where the related site is `NULL`. Once a site joins a project, this entry is linked accordingly and can no longer be used by anyone else.

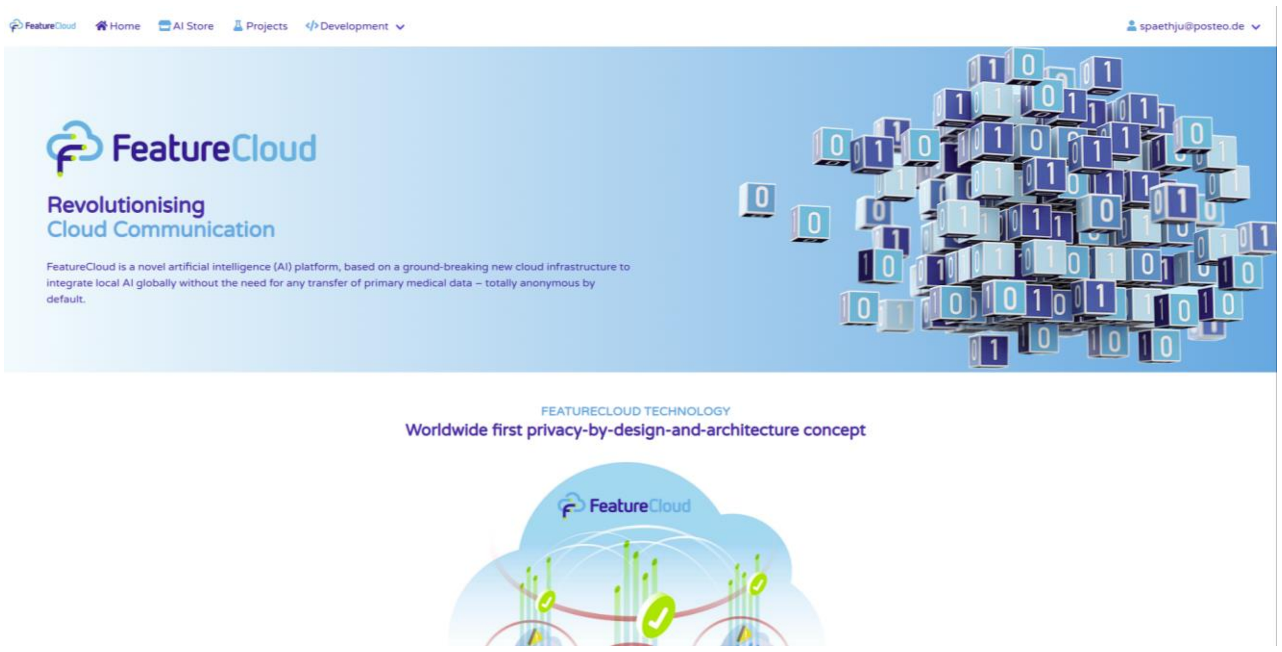
App. Apps are AI applications which appear in the app store. They contain an image name, which needs to be used when pushing new versions of the app, an icon, a short and long description, tags, a category and link to the source code. They are linked to a developer through the ‘develops’ relation and workflows they are part of through the ‘is in workflow’ relation.

App Version. New versions of apps are tracked automatically when pushing a new version via Docker by the developer and are linked to the respective app through the ‘has’ relation.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

4.2.4 Frontend

The frontend serves as a graphical user interface (GUI) for FeatureCloud users and developers. It is the only component FeatureCloud users directly interact with. It then calls the API of the controller or the global backend on behalf of the user, depending on the nature of the task (see sections 4.2.1 and 4.2.3). Since the frontend needs to be platform-independent, it has been implemented as a web application running inside a browser. This enforces a clear separation between GUI-related concerns and backend-related tasks by employing an HTTP-based API, as described earlier. Angular has been chosen as a web framework due to its high popularity, long-term support and extensive functionality.



FeatureCloud is a novel artificial intelligence (AI) platform, based on a ground-breaking new cloud infrastructure to integrate local AI globally without the need for any transfer of primary medical data – totally anonymous by default.

Figure 13: ***FeatureCloud Frontend.*** *The frontend serves as a GUI for the users and allows intuitive project management, workflow execution, presentation of apps in the AI store, and many more.*

The GUI is structured into the following sections (accessible through the menu): 1) Account management, 2) Site management, 3) App management, 4) Project management and 5) App testing, each divided into subsections again. For more details and walkthroughs, see supplement, section 1.

4.2.5 AI Store Server

As described in section 3.2.1, the AI store server is connected to the global backend that serves as an auth server and a Docker registry (see fig. 4). It performs two main tasks: relay queries from the local Docker engines using the Docker registry API[1] and protecting images from unpermitted access, in particular restricting pushing of images to the respective app developers. For that, the AI store server provides endpoints to request a JWT token which is then attached automatically by

[1] https://docs.docker.com/registry/spec/api/

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

the Docker CLI to authenticate consecutive actions. App developers need to be FeatureCloud users and use their FeatureCloud credentials to login. That way, the global backend acting as an auth server can check whether the user pushing an image is the corresponding app owner.

Like the controller and relay server it is written in Go for performance reasons. App images can be several GB large and pulling images is a task performed each time before a workflow step is executed, making performance a critical consideration.

5 Results

This section describes different apps already implemented to show the generic nature of FeatureCloud and demonstrate its operability.

5.1 FeatureCloud Apps

5.1.1 Linear and Logistic Regression

Linear and logistic regression algorithms are very popular in many fields of science including bioinformatics and biomedicine due to their simplicity and interpretability [16]. The aim of regression models is to estimate (predict) the relationship between a dependent variable (label or output) and one or more independent variables (features or predictors). The data is represented as a table in which rows are the samples and columns are the values of the variables. We use *S, X,* and *Y* to indicate the sample space, feature space, and label space, respectively. Linear regression is used for quantitative label values, whereas logistic regression predicts the binary (0/1) label values.

Federated linear and logistic regression apps in FeatureCloud are based on their corresponding implementations in sPLINK [17]. The apps implement a horizontal federated learning approach [18–20], where the sample space is different among the datasets distributed across the clients, but the datasets share the same feature and label spaces. The federated logistic and linear regression apps implement the privacy-preserving versions of the Newton-Raphson method for logistic regression and the ordinary least square method for linear regression. The former algorithm is an iterative one, while the latter is a single-step algorithm.

In the federated linear regression app, each client *i* computes two local model parameters from its local dataset and shares them with the server (coordinator): $P^1_i=X_i^TX_i$ and $P^2_i=X_i^TY_i$, where T is the transpose matrix operation. The server adds up the local parameters from all clients to compute global parameters P^1_G and P^2_G and compute the coefficients (weights) of the model using Beta = $(P^1_G)^{-1}\,(P^2_G)$, where *-1* indicates inverse matrix operation[17]. The important assumption here is that the number of features is less than the number of samples, so that the server cannot reconstruct the data from the model parameters.

In the federated logistic regression, each client *i* calculates local gradient and Hessian matrices as well as the log-likelihood value as the local parameters using the following [17]:

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

$$\hat{Y}_i = \frac{1}{1+e^{-X_i\beta}}$$

$$\nabla_i = X_i^T(Y_i - \hat{Y}_i)$$

$$H_i = (X_i^T \circ (\hat{Y}_i \circ (1-\hat{Y}_i))^T)X_i$$

The server adds up the local parameter values to compute the corresponding global values. Next, it updates the coefficient using $\text{Beta}_{\text{new}} = \text{Beta}_{\text{old}} + H^{-1}\nabla$, where Beta_{old} is the beta values from the previous iteration, and H and ∇ are the global hessian and gradient matrices, respectively.

It has been proved mathematically and shown empirically that the federated linear and logistic regression algorithms provide the same results as those from the corresponding centralized versions [17]. Therefore, the regression apps are robust from the accuracy perspective, i.e. they incur no accuracy loss regardless of the data distribution across the clients (homogeneous or heterogeneous).

5.1.2 Time-to-Event Analysis

Time-to-event analysis, sometimes called survival analysis, describes a particular type of algorithm developed to analyse so-called time-to-event data [21]. This data includes information about the time until a certain event happens, e.g. death, and comes with the difficulty that the event often has not occurred for all samples in the dataset (right-censored samples) during observation time [22]. As this kind of data is often collected in clinical trials [23], an implementation of the most common algorithms as FeatureCloud apps can help bring together data from different sites, enlarge sample size, and enhance the quality of the models.

Comparing our results with the implementations of the state-of-the-art survival analysis package Lifelines, we could show that the results of the federated Kaplan-Meier Estimator and federated Nelson-Aalen Estimator are equal to the centralized algorithms. Also, the federated logrank test produces highly similar results and only differs in the seventh decimal place. As the federated implementation of the Cox proportional hazard model varies slightly from the lifelines implementation, results are similar to the third decimal place but model quality measured using the concordance index is still comparable. All algorithms keep the results even with an increasing number of participants or if the samples are unequally distributed between the clients. A publication proofing these results will follow next year.

Kaplan-Meier Estimator. The Kaplan-Meier estimator is a statistic to estimate the survival function of time-to-event data [24]. The FeatureCloud Kaplan-Meier Estimator app expects a csv/tsv/sass-file as input, including a time and an event column (1 event occurred, 0 censored). Optionally, a category column can be included, which e.g. defines the different treatment arms of a study. If this is the case, a survival function is estimated for each category separately and subsequently compared pairwise using the logrank-test statistic to see if they significantly differ in their survival.

The results will consist of a CSV table containing the survival function matrix, a plot showing the survival curves, and, if a category column was included, the p-values of the pairwise logrank-test statistic.

Nelson-Aalen Estimator. The Nelson-Aalen estimator is a statistic to estimate the cumulative hazard function of time-to-event data [25]. The Nelson-Aalen Estimator FeatureCloud app works in the same way as the Kaplan-Meier Estimator FeatureCloud app, with the only difference that it computes the cumulative hazard function instead of the survival function.

Cox Proportional Hazard Model. The Cox proportional hazard model is a regression algorithm for time-to-event data used to find biomarkers significantly associated with the survival of patients [26,27]. The Cox proportional hazard model expects a csv/tsv/sass-file as input, including a time ad event column. Furthermore, it expects at least one column containing the numerical values of a covariate, e.g. age, or blood oxygen level.

The results will consist of a table including statistics such as the coefficient, hazard ratio, and p-value of each covariate, a plot showing the log hazard ratio of each covariate together with its 95% confidence intervals, and the c-index of the model.

5.2 Summary

When comparing the conventional, centralized algorithms with their federated versions, their results can be identical or differ slightly. However, even when they differ, they generally still benefit from the greater amount of data that can be taken into account. From the algorithms that have been investigated so far, and this is confirmed by literature, it can be concluded that the federated approach taken in FeatureCloud is practically feasible and allows for better insights through incorporating more data.

6 Discussion

The FeatureCloud platform has been developed to an extent in which it can be applied to practical problems in the area of biomedicine and beyond. It is general enough to allow for a variety of ML algorithms yet offers pre-built solutions for common use cases, in the form of apps in the AI store or app templates for developers. The concept of arbitrarily composing apps in a workflow proves to be challenging due to the necessity of a common data format, which is not always available, or reduces flexibility. The same applies to the initial data, which needs to be provided in a form processable and understandable by the desired apps.

Since it is necessary to understand which functionality and which types of data will be used precisely, which ML techniques prove to be most prevalent in federated settings, and which challenges arise when using the platform, few assumptions can be made in advance. The approach FeatureCloud takes is to keep the platform as flexible and extensible as possible and align new functionality closely to the demand of its users.

The important question is what can be moved from the app developers shoulders to the FeatureCloud platform (e.g. pre-implemented cryptographic techniques, pre-implemented communication modes) and what needs to remain in the domain of the apps themselves. This will be assessed and decided together with the FeatureCloud community, further strategic meetings within the FeatureCloud consortium and be based on further research outcomes in the area of privacy-preserving techniques in ML.

7 References

1. Sadat, M. N. *et al.* SAFETY: Secure gwAs in Federated Environment through a hYbrid Solution. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* vol. 16 93–102 (2019).
2. Hardy, S. *et al.* Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. (2017).
3. Li, W. *et al.* Privacy-Preserving Federated Brain Tumour Segmentation. *Machine Learning in Medical Imaging* 133–141 (2019) doi:10.1007/978-3-030-32692-0_16.
4. Li, X. *et al.* Multi-site fMRI analysis using privacy-preserving federated learning and domain adaptation: ABIDE results. *Medical Image Analysis* vol. 65 101765 (2020).
5. Geyer, R. C., Klein, T. & Nabi, M. Differentially Private Federated Learning: A Client Level Perspective. (2017).
6. Truex, S. *et al.* A Hybrid Approach to Privacy-Preserving Federated Learning. *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security - AISec’19* (2019) doi:10.1145/3338501.3357370.
7. Wei, K. *et al.* Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Transactions on Information Forensics and Security* vol. 15 3454–3469 (2020).
8. Raisaro, J. L. *et al.* MedCo: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* vol. 16 1328–1341 (2019).
9. Kim, M., Lee, J., Ohno-Machado, L. & Jiang, X. Secure and Differentially Private Logistic Regression for Horizontally Distributed Data. *IEEE Transactions on Information Forensics and Security* vol. 15 695–710 (2020).
10. Froelicher, D. *et al.* UnLynx: A Decentralized System for Privacy-Conscious Data Sharing. *Proceedings on Privacy Enhancing Technologies* vol. 2017 232–250 (2017).
11. Gazula, H. *et al.* COINSTAC: Collaborative Informatics and Neuroimaging Suite Toolkit for Anonymous Computation. *Journal of Open Source Software* vol. 5 2166 (2020).
12. Beyan, O. *et al.* Distributed Analytics on Sensitive Medical Data: The Personal Health Train. *Data Intelligence* vol. 2 96–107 (2020).
13. Deist, T. M. *et al.* Distributed learning on 20 000+ lung cancer patients - The Personal Health Train. *Radiother. Oncol.* 144, 189–200 (2020).
14. Mittal, S. & Hasija, Y. Applications of Deep Learning in Healthcare and Biomedicine. *Studies in Big Data* 57–77 (2020) doi:10.1007/978-3-030-33966-1_4.
15. Silva, S., Altmann, A., Gutman, B. & Lorenzi, M. Fed-BioMed: A General Open-Source Frontend Framework for Federated Learning in Healthcare. *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning* 201–210 (2020) doi:10.1007/978-3-030-60548-3_20.
16. Ren, X., Mi, Z. & Georgopoulos, P. G. Comparison of Machine Learning and Land Use Regression for fine scale spatiotemporal estimation of ambient air pollution: Modeling ozone concentrations across the contiguous United States. *Environ. Int.* 142, 105827 (2020).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

17. Nasirigerdeh, R. *et al.* sPLINK: A Federated, Privacy-Preserving Tool as a Robust Alternative to Meta-Analysis in Genome-Wide Association Studies. doi:10.1101/2020.06.05.136382.
18. Konečný, J. *et al.* Federated Learning: Strategies for Improving Communication Efficiency. (2016).
19. Yang, Q., Liu, Y., Chen, T. & Tong, Y. Federated Machine Learning. *ACM Transactions on Intelligent Systems and Technology* vol. 10 1–19 (2019).
20. Torkzadehmahani, R. *et al.* Privacy-preserving Artificial Intelligence Techniques in Biomedicine. (2020).
21. Altman, D. G. & Bland, J. M. Time to event (survival) data. *BMJ* 317, 468–469 (1998).
22. Prinja, S., Gupta, N. & Verma, R. Censoring in clinical trials: review of survival analysis techniques. *Indian J. Community Med.* 35, 217–221 (2010).
23. Singh, R. & Mukhopadhyay, K. Survival analysis in clinical trials: Basics and must know areas. *Perspect. Clin. Res.* 2, 145–148 (2011).
24. Kaplan, E. L. & Meier, P. Nonparametric Estimation from Incomplete Observations. *Springer Series in Statistics* 319–337 (1992) doi:10.1007/978-1-4612-4380-9_25.
25. Aalen, O. Nonparametric Inference for a Family of Counting Processes. *The Annals of Statistics* vol. 6 701–726 (1978).
26. Cox, D. R. Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)* vol. 34 187–202 (1972).
27. Tibshirani, R. THE LASSO METHOD FOR VARIABLE SELECTION IN THE COX MODEL. *Statistics in Medicine* vol. 16 385–395 (1997).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

D) Table of Acronyms and Definitions

AI	Artificial intelligence
API	Application programming interface
CLI	Command line interface
CI/CD	Continuous integration / continuous deployment
concentris	concentris research management GmbH
CSS	Cascading style sheets
CSV	Comma-separated values
DL	Deep learning
DP	Differential privacy
E/R	Entity/relationship
GDPR	General Data Protection Regulation
GND	Gnome Design SRL
GUI	Graphical user interface
HE	Homomorphic encryption
HTML	Hypertext markup language
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol (secure)
IP	Internet protocol
JSON	JavaScript object notation
JWT	JSON web token
ML	Machine learning
MR	Merge request
MS	Milestone
MUG	Medizinische Universitaet Graz
Patients	In this deliverable, we use the term “patients” for all research subjects. In FeatureCloud, we will focus on patients, as this is already the most vulnerable case scenario and this is where most primary data is available to us. Admittedly, some research subjects participate in clinical trials but not as patients but as healthy individuals, usually on a voluntary basis and are therefore not dependent on the physicians who care for them. Thus to increase readability, we simply refer to them as “patients”.
RF	Random forest
SDU	Syddansk Universitet
SMPC	Secure multiparty computation
SSH	Secure shell
SSL	Secure sockets layer
SVM	Support vector machine
TCP	Transmission control protocol
TUM	Technische Universitaet Muenchen
WP	Work package

E) Other Supporting Documents, Figures and Tables

1 Demo and Manual

A running version of the FeatureCloud platform can be found here:
https://staging.featurecloud.eu

A user manual, largely in the form of tutorial and demo videos, can be found on this subpage:
https://staging.featurecloud.eu/manual

The staging area is a replica of the public production area which will be made public upon release of the platform. It offers the same functionality and is used for manual testing.

2 Manual for App Developers

2.1 Introduction

To be executable on the FeatureCloud platform, a federated app must implement the FeatureCloud API. The API is designed in a generic way, it puts minimal constraints on the actual implementation of the app, so any kind algorithm can be implemented. The app must be able to act both as coordinator or participant. This is needed because the same app is downloaded by the platform to all participant sites, and the app's role (coordinator or participant) is provided by the platform at setup.

2.2 FeatureCloud API

A federated app should act as a web server polled by the FeatureCloud platform, so implementing the FeatureCloud API basically means implementing a web server that handles the following requests.

POST /setup

When the participants are ready to start the federated execution (they are connected and prepared the input data) the platform will send the setup request. This is the starting point of the federated execution, the app can use it as a trigger to start the computation based on it's local data.

The request body contains the following information:

- id (string): the app instance identifier, determined by the platform
- master (boolean): this value specifies the role of the app instance: true for coordinator, false for participant
- clients (array of string): contains the identifiers of all participants

Example of setup data for a coordinator, when there are 3 participants in total:
```
{
	id: "0"
	master: true
	clients: ["0", "1", "2"]
}
```

GET /status

With the response to this request the federated app reports its current status. The app indicates if there is data to be transferred to the coordinator or if the execution of the app is finished.
The response should contain the following information:

- available (boolean): true if there is data to be transferred, otherwise false.
- finished (boolean): true if the app execution finished, otherwise false.
- size (int, optional): This value can be used to indicate the size of the data that will be transferred.

Example:

```
{
	available: true
	finished: false
	size: 16
}
```

GET /data

Using this API call apps can transfer data to the platform.
The response body should contain the data to be transferred. If size was specified in the /status response, the platform will check if the content length matches the size value.
The platform reads the data and redirects in the following way, depending on the sender:
- if the data is coming from a participant, it will be redirected to the coordinator
- if the data is coming from a coordinator, it will be broadcast to all other participants.

POST /data

Using this API call the platform transfers data to the app. The request body should contain the data to be transferred. The app should handle/consider the received data in the following way, depending on their role:

- If the receiver is a coordinator, the data is a packet from a participant (the ID of the sender is provided as a GET parameter ‘client’, e.g. /data?client=1)
- If the receiver is a participant, the data is a broadcast message from the coordinator

Besides implementing the above defined API, a federated optionally can have its own GUI, which is displayed by the FeatureCloud platform. The GUI is served by the app’s web server, but it’s implementation is fully up to the app developer.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

2.3 App templates

In general, you can develop FeatureCloud apps in any programming language and framework you want, as long as the API is addressed correctly. However, to make it as easy as possible, we already provide templates shipped with the essential features to efficiently develop an app.

Example: Python Template

Our Python template includes a Flask web server. The main directory contains the following files that might need changes:

- .gitignore: Add files and folders that should not be uploaded to the git repository
- README.md: Describe your app
- build.sh: Used to build your docker image. Here you can decide how your image shall be named
- requirements.txt: Used to install all python packages that are needed for your app. Add all requirements here.

When your app is ready to get tested, run the build.sh to create a docker image that subsequently can be tested in the Feature Cloud Testing Environment. The actual app development happens in the fc_app, more precisely in the api.py and web.py file. We recommend outsourcing the logic of your algorithm into a separate file, e.g., algorithm_name.py.

api.py

In the api.py file, the basic API methods (status, data, setup and retrieve_setup_parameters) for Feature Cloud are implemented and can be extended to fulfill your algorithm's requirements. All crucial variables should be stored in Redis to be used between the api.py and web.py files. They can be set using the redis_set() method and read using the redis_get() method. Important, predefined variables are:

- available: true if data is available for sharing with the coordinator, else false
- is_coordinator: true if the client is the coordinator of the analysis., else false
- finished: true if the computation is done, else false. After set to true, the server will end the analysis.
- nr_clients: Number of clients participating in the analysis.

STEPS should be defined to structure the process of your app. Especially after the setup, different kinds of data need to be exchanged during the analysis. By defining different steps, you can distinguish what data will be exchanged in the data call. You can set the current step using the set_step() method. The get_step() method will give you the current step.

Requesting data. The data() method is probably the most crucial in your app development. With the POST request, clients can pull global data from the coordinator, or the coordinator can pull data from the clients. With the GET request, clients can send data to the coordinator, or the coordinator can broadcast data to the clients. Depending on the step, different data can be exchanged between clients and the coordinator.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Reading the input. Input files are always located at the directory "/mnt/input/". You can either directly use the files from there (e.g., if there is only one possibility for a file) or implement a file selector in the frontend (web.py).

Writing the output. All result files need to be saved in the "/mnt/output" directory before the coordinator's finished flag is set. The storage of the results in this directory is essential for the user to download the results or continue with them in the next workflow step. You can also store intermediate results that might be interesting for the users to have.

Finishing the analysis. As soon as the coordinator has computed the final global result, the analysis can be finished. Therefore, the coordinator's finished flag needs to be set (redis_set('finished', true)).

web.py

In contrast to the api.py file, the web.py should contain no app logic itself. It is responsible for the app frontend and has access to all Redis variables set in api.py. It should know about the app's STEPS and show the corresponding frontend for each step. You are entirely free in what you offer in the frontend. Sometimes no frontend is needed at all, but in other cases, a loading screen or even a whole frontend with user input is necessary.

Showing different frontend pages. To show different frontend pages, use if-else in the root() method to distinguish between the various steps and return the corresponding HTML page stored in the templates folder with return render_template('test.html').

Example mean app. We also provide an extensive example of a Feature Cloud Mean App (https://github.com/FeatureCloud/mean_template).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

3 Workflow Sequence Diagram

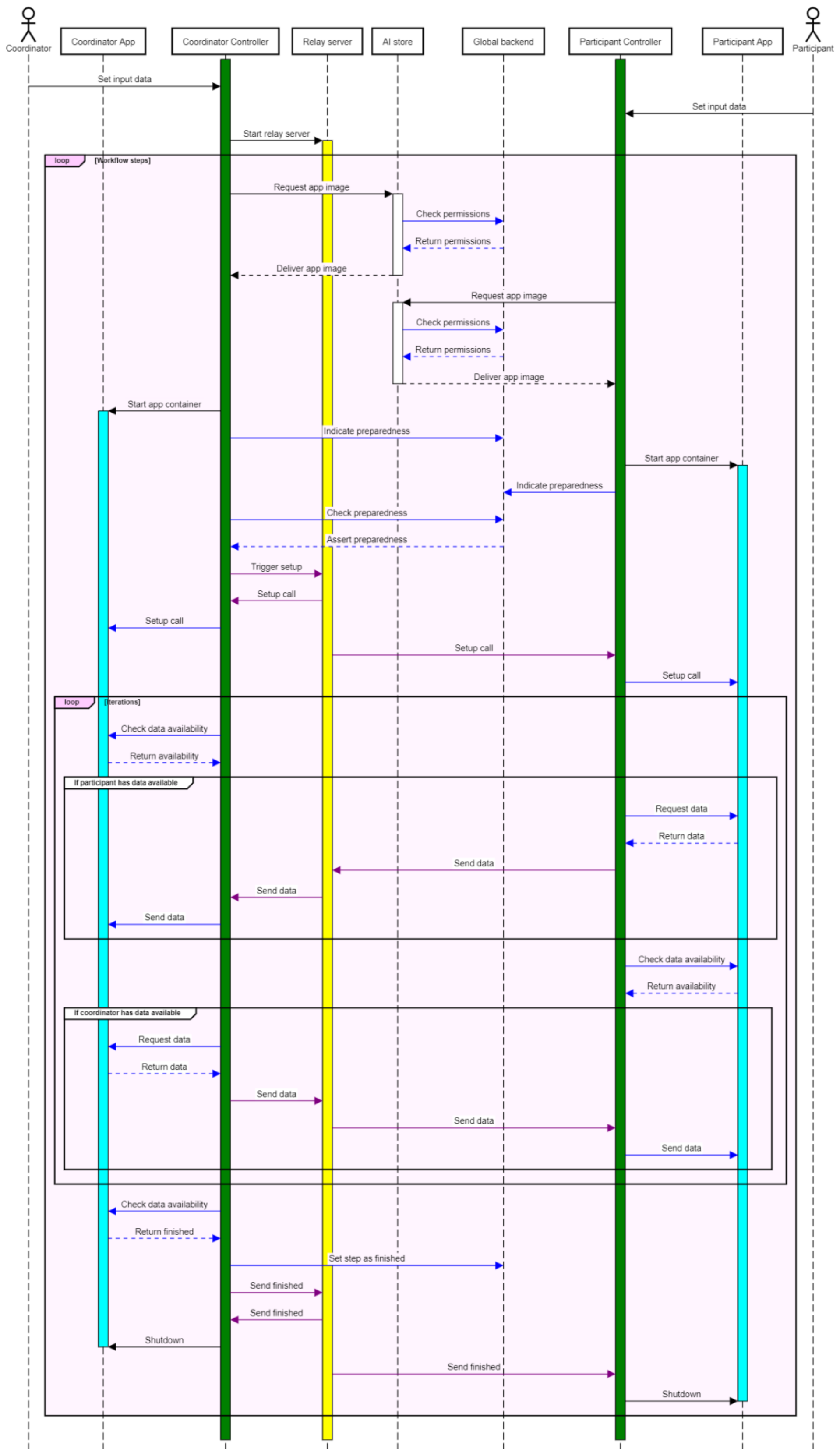


Figure S.1: ***Sequence diagram of the workflow execution.*** *Purple arrows represent binary data traffic, blue arrows HTTP traffic and black arrows are trigger events.*

Traffic involving the relay server generally contains model parameters that might be harmless for single workflow runs. However, if put together and collected over multiple workflows, one might be able to infer information about the local data. Therefore, it is split from the global backend which is centralized to allow for project management involving all FeatureCloud users and data holders. The global backend however only contains meta information which is unrelated to the sensitive data stored at the data holder sites (see also E/R diagram in fig. 12).

4 Development Process

As described in section 2.1.2, FeatureCloud focuses on robustness and convenience for developers and plays a major role in the whole development process. This affects requirements management and integration of requirements into the development process as well as maintaining a high level of code quality and robustness of the system. We use SCRUM as a development process framework with sprints of 2 weeks.

4.1 Requirements and SCRUM

Requirements are collected from all stakeholders of the FeatureCloud project, of which we identified the following:

- App developers
- Hospital IT personnel
- Patients
- Medical doctors/researchers
- Platform developers

Up until now, due to the set of features required for the first stage, we mainly considered app developers, researchers and platform developers as stakeholders. Patient consent management and frontends for patients are due at a later stage.

All requirements, parts of which have been reported in deliverable 7.1, revised version (as of now not officially accepted yet), are tracked as GitLab issues, marked with a ‘user story’ label to distinguish them from lower-level, development-related tasks. These high-level user stories are regularly discussed in biweekly review meetings, where all developers and users of the platform can take part, currently mainly within the consortium.

User stories are broken down into tasks and discussed during the sprint planning meetings, which take place at the beginning of each sprint. However, fixes for bugs that are reported can be injected into the sprint at any time. During the sprint, the platform developers (currently 4 from TUM and 2 from GND) meet 3 times per week to discuss the latest progress and coordinate development. A figure illustrating the development process also has been reported in Fig. 1 of deliverable 7.1 mentioned above.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

4.2 Testing, Linting and CI/CD

Maintaining a high level of code quality first and foremost requires a high level of discipline by the developers. The following process, which is usual for agile development, is maintained in FeatureCloud:

- Every change to the code is made in a separate branch
- When done, the branch is turned into a merge request and needs to be reviewed by another developer

Both of these requirements can be enforced in GitLab.

On top of that, a series of automated measures are taken to detect bugs early and ensure that important functions of the system are working:

- Automated builds
- Automated tests (unit tests, end-to-end tests)

To make sure that the software is tested can be enforced automatically. Therefore, a test coverage of over 90% needs to be met.

To increase code quality, linting is used for both frontend and backend:

- Frontend: tslint with strict ruleset enabled (including enforced typing, e.g. no any type)
- Backend: flake8 and pycodestyle

Figure S.2 shows the frontend pipeline containing all the above mentioned steps. It needs to run through without errors (except for the ‘Deploy’ stage) before a merge request can be merged.

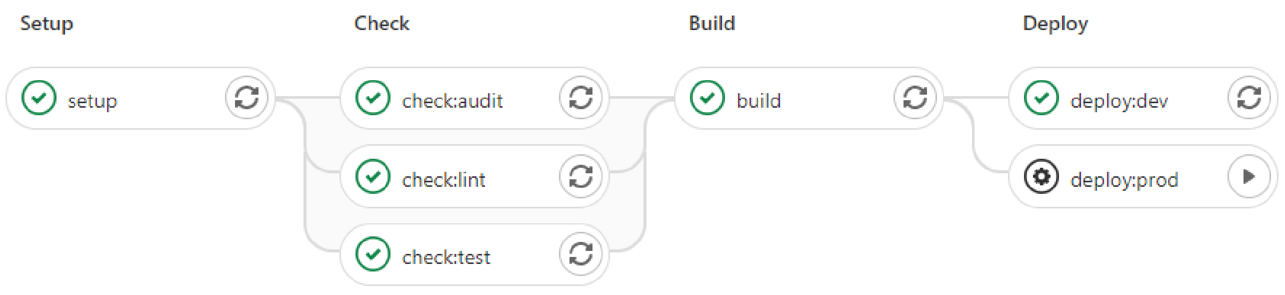


*Figure S.2. **Frontend pipeline.** The production deployment step needs to be triggered manually, staging is being deployed automatically after a MR has been merged into the master branch.*

When a merge request (MR) has been merged into master, a Docker container is built automatically from the code and pushed to a private registry. The staging deployment detects a new version of the frontend image, pulls it and restarts the container without further user interaction. This way continuous deployment is achieved.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.