Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG


Source: https://www.rmit.edu.au/study-with-us/biomedical-sciences

ISMB 2022 - Tutorial Session

Developing Federated Applications

ISMB 2022, Federated Learning in Biomedicine (Tutorial)

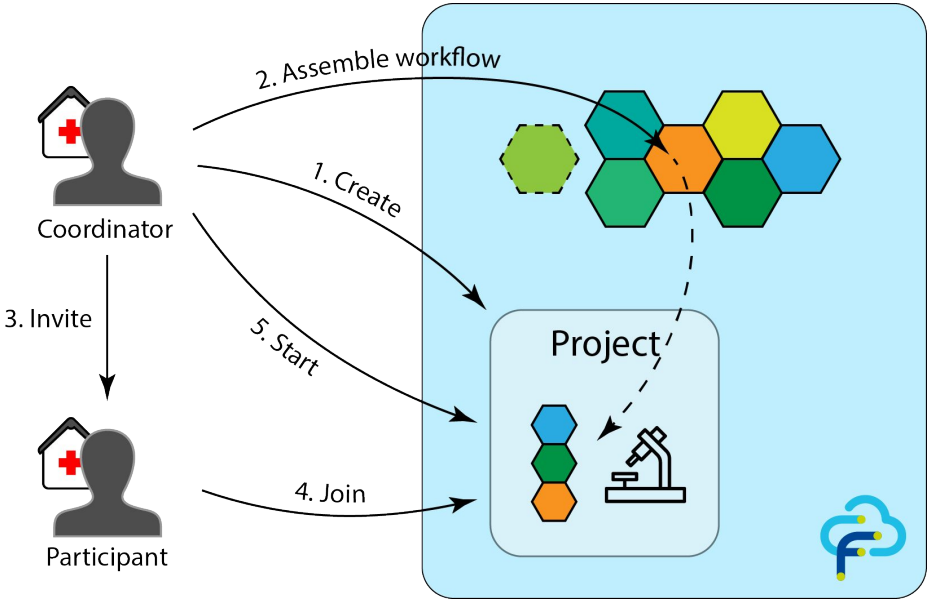
Table of Contents

Developing Federated Applications

FC platform, Apps, and Workflows

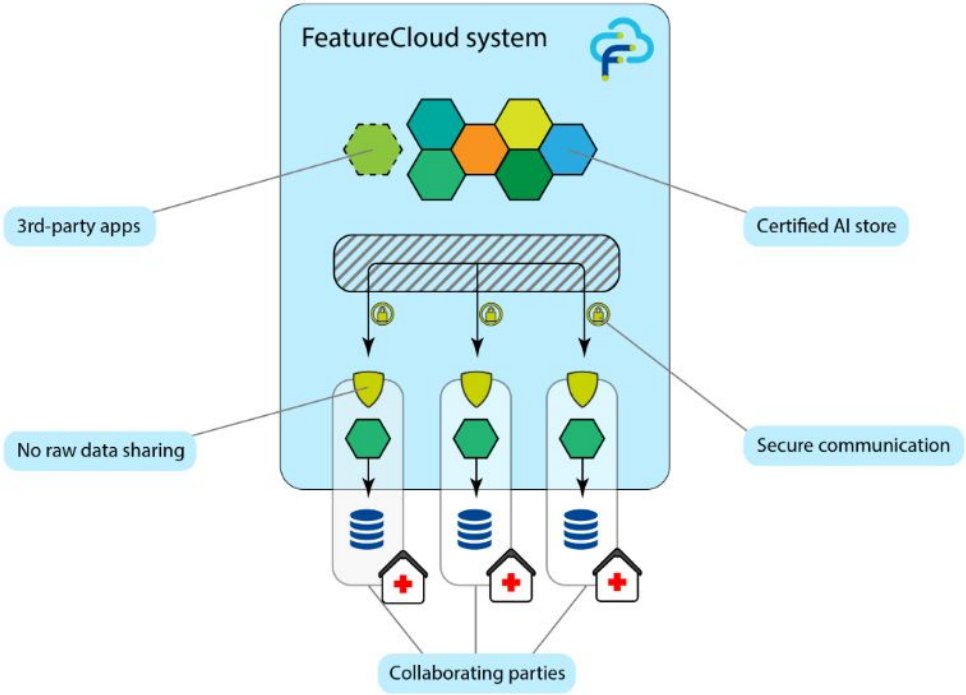
Developing Federated Applications

FeatureCloud: Federated collaboration



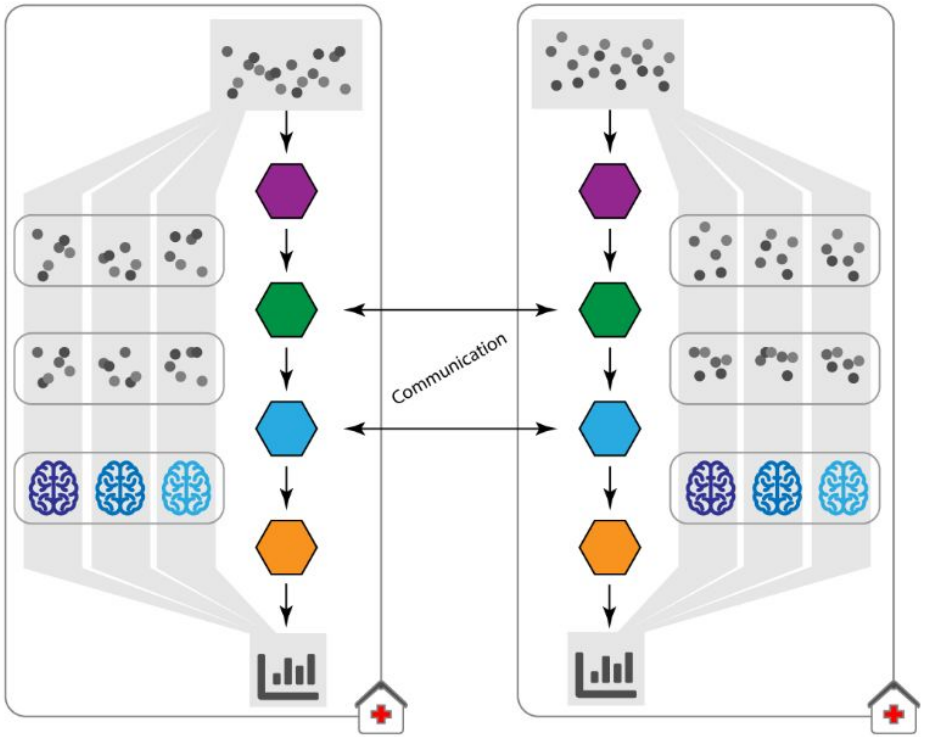
Developing Federated Applications

Federated Learning in FC



Developing Federated Applications

FeatureCloud Workflow



Developing Federated Applications

FeatureCloud App

Back

Update Edit Remove

EVALUATION
CLASSIFICATION
FeatureCloud

Julian Späth
Evaluation (Classif.)

An app, computing various metrics to evaluate the performance of a classification model
Source code: Link

Tags
federated computation classification evaluation f1 precision accuracy recall sensitivity specificity

Image name: fc_roc
Last update: Sep 2, 2021, 1:45:06 PM
Report Bug

Description Reviews Statistics

Classification Evaluation FeatureCloud App

Description

A Classification Evaluation FeautureCloud app, allowing to evaluate your trained models with various classification metrics (e.g. Accuracy).

Input

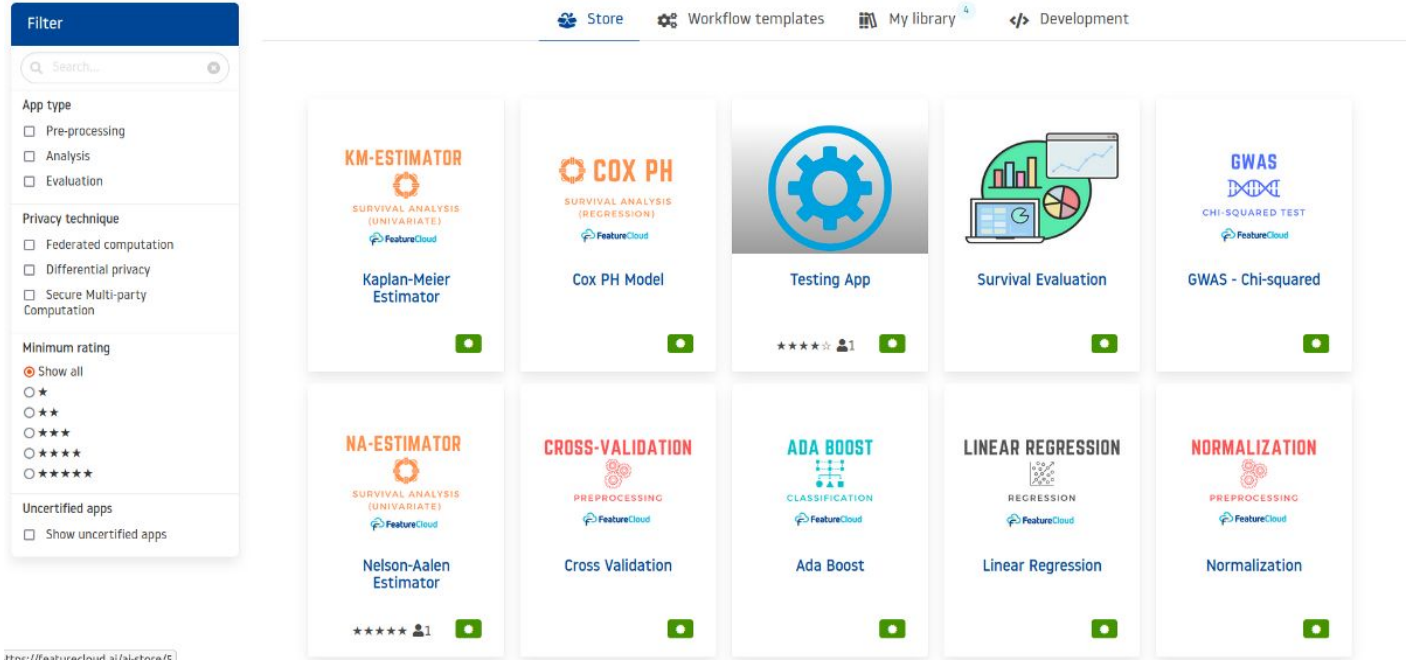
- test.csv containing the actual test dataset
- pred.csv containing the predictions oof the model on the test dataset

Output

- score.csv containing various evaluation metrics

Developing Federated Applications

FeatureCloud AI-Store



FeatureCloud: App execution

- **testbed**:
  - Standalone app
  - providing input data and config file(if required)
  - Suitable for app developers to test their platform
  - Supported by both front-end and the pip package's CLI
- **workflow**
  - Linear execution of a workflow of apps
  - providing first app's data
  - providing a config file for all apps
  - passing any app's results as input for the following app
- **test workflow**
  - Non-linear workflow
  - Supported in the pip package CLI

Developing Federated Applications

App execution: Test-bed for app developers

Setup

1. What is the name of your Docker image?

Image

featurecloud.ai/flimma

2. How many clients do you want to test?

1 2 3 4 5 6 7 8 9 10

3. What is the workspace directory of the clients?

Client 1

/home/mohammad/PycharmProjects/FeatureCloud/data/ flimma/c1

Client 2

/home/mohammad/PycharmProjects/FeatureCloud/data/ flimma/c2

4. What is the generic directory that all clients will have access to?

All clients will have access to 'Generic directory'. You can use this to pass common configuration or data.

Generic directory

/home/mohammad/PycharmProjects/FeatureCloud/data/ flimma/generic

5. What communication channel to use?

Local channel

Internet channel

'Local channel' keeps the traffic on your machine and is faster. 'Internet channel' sends the traffic through the FeatureCloud server and is slower but closer to the real world setting.

6. How often do you want the testbed clients to exchange information?

Query interval (seconds)

3

Setting to a lower number is advised when having a large number of iterations each taking little time, setting to a higher value is recommended when calculations/operations need more time. Default value is 3 seconds.

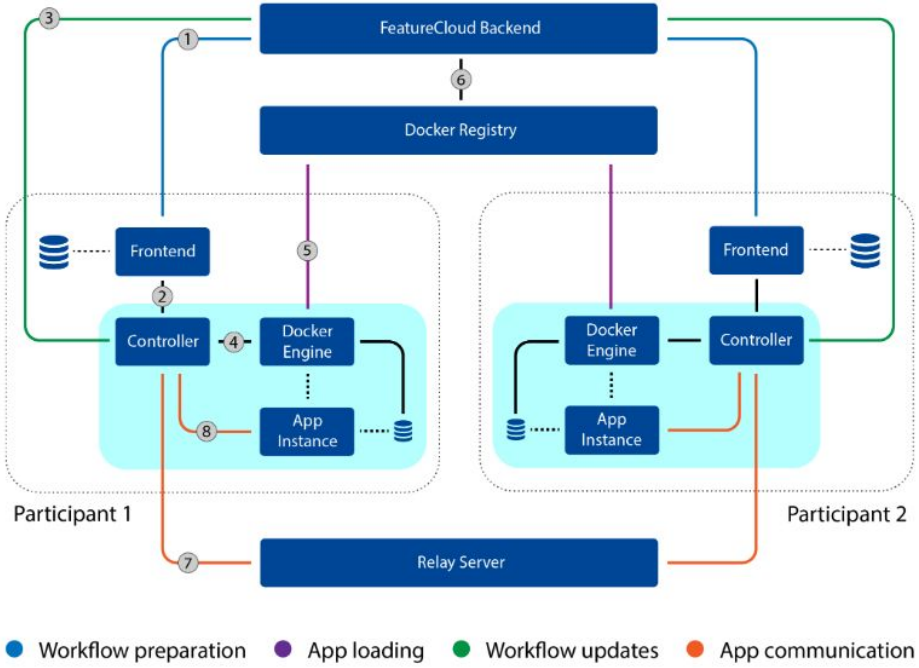
7. Where should the results of the test run be saved?

/home/mohammad/PycharmProjects/FeatureCloud/data/tests/ results

Start

Developing Federated Applications

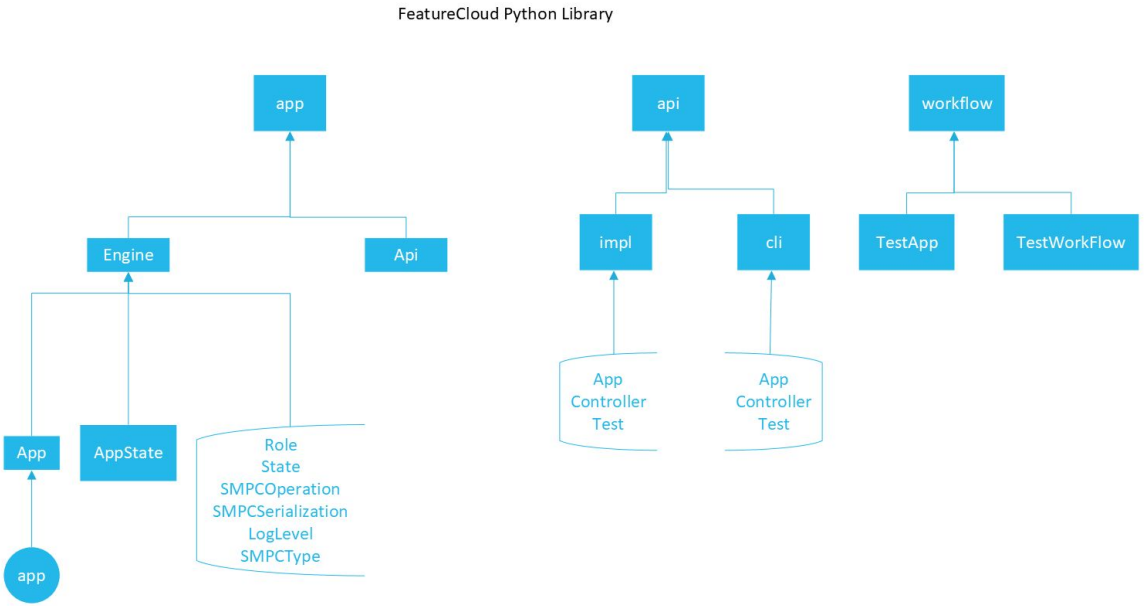
FeatureCloud Architecture



Developing Federated Applications

App development

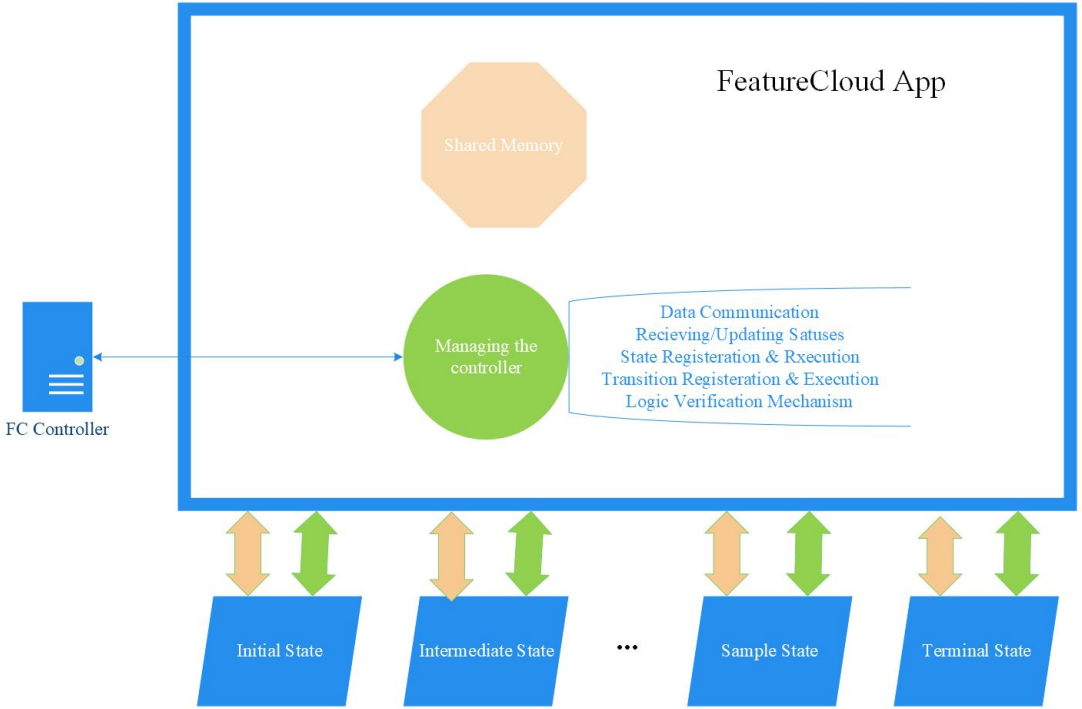
FeatureCloud pip package: Overview



Developing Federated Applications

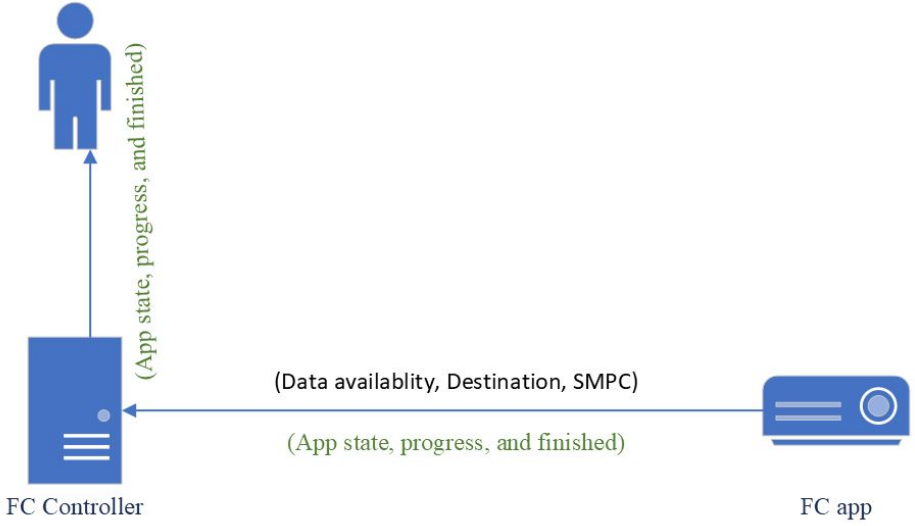
FC App, State, anc

- **AppClass**
- **SateClass**
- **Interactions:**
  - With the controller
  - Among the states



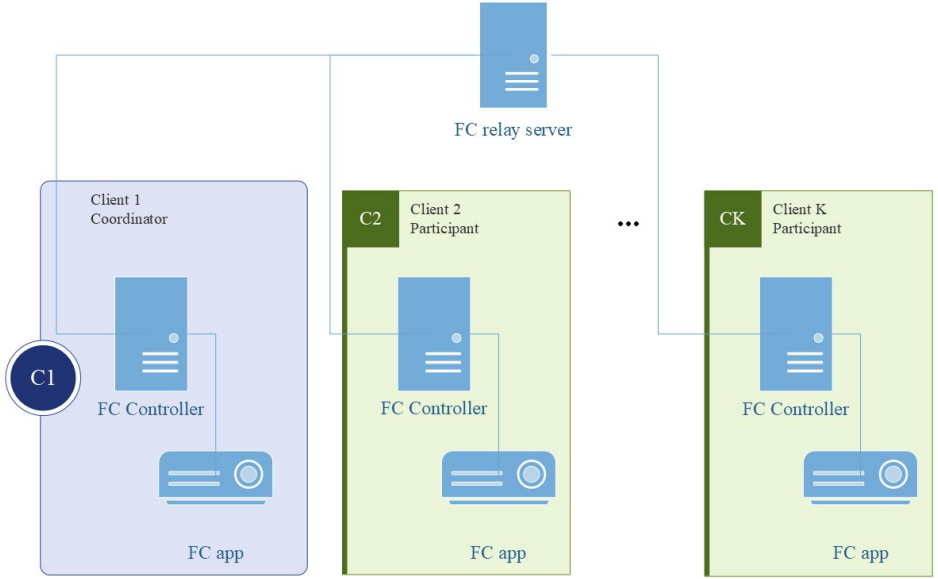
FeatureCloud: Status Attributes

- **For data Communication:**
  - status_available
    - Signalling the controller to execute the communication
  - Status_destination
    - Informing the controller about destination client that you want to communicate to
  - status_smpc
    - SMPC parameters
- **For the end-user:**
  - status_finished
    - signalling the controller the end of app execution.
  - status_message
    - messaging the end-user in front-end.
  - status_progress
    - Informing the end-user of the app progress
  - status_state
    - Informing the end-user of the state of the app



App Roles (Coordinator vs Participant)

- **Different roles, different access level**
  - Data access: Aggregate data
  - State and transition permission
- **Logic Verification Mechanism (LVM)**
- **Role Tuples**
  - Role.COORDINATOR
  - Role.PARTICIPANT
  - Role.BOTH



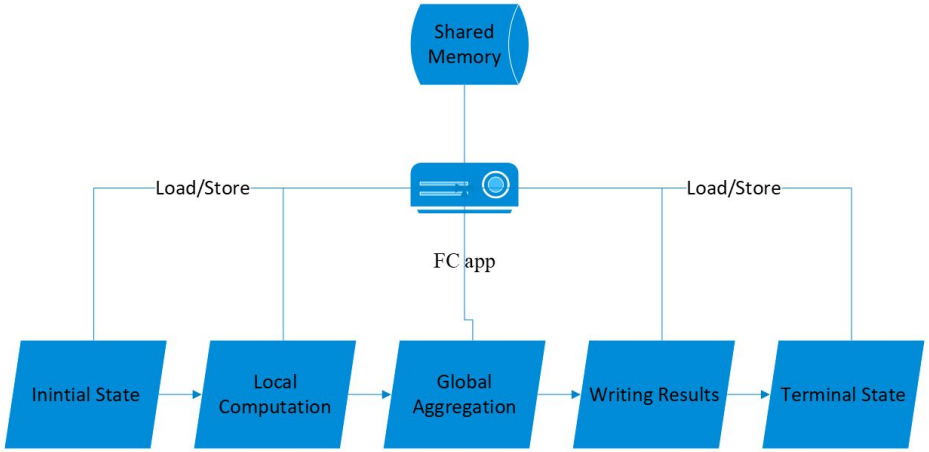
App logs and operational states

- **Operational States**
  - RUNNING: running
  - ERROR: error
  - ACTION: action_required

- **Log levels**
  - DEBUG: debug-related logs.
  - ERROR: message the error in the front-end
  - FATAL: fatal events that the app may encounter during the execution and cannot recover from.

FC: Shared Memory

- Local data sharing among states
- App.internal
  - Shared memory in the app instance.
    - States are separate instantiation of AppState class to handle local and/or global computations.
    - No Intrinsic shared memory.
    - AppState store and load methods.
  - Python Dictionary



FeatureCloud App: Summary

- **Managing App execution:**
  - Instantiate states
  - Register states
  - Verify the logic
  - Execute states
  - Transit between states
  - Log the app execution
  - Supports shared memory for states
- **Fully transparent:**
  - No instantiation by developers
  - No registration by developers
  - Avoidable direct call by developers

FeatureCloud State: Overview

- **Abstract class**
- **Abstract methods**
  - Registering transitions
  - Executing local computations

- **Communication methods**
  - Aggregating clients data
  - Gathering clients data
  - Waiting to receive data
  - Communicating Data to others
  - Communicating data to the coordinator
  - Broadcasting data

- **Registering a specific transition for state**
- **Updating local app status: update**
- **Configuring SMPC Module**
  - Secure Multi-Party Computation
    - Exponent
    - Shards
    - Operations
    - Serialization

State: Checking roles and IDs

- **Roles and IDs are available once states are registered (in run() method)**
- **Role of the local app and ID of all clients will be set by the Controller**
- **Checking the role of the app instance:**
  - is_coordinator()
- **Checking the clients' ID**
  - clients()
- **Checking the role of the app instance:**
  - id()

State: Defining a custom state

- **Extending the AppState class**
- **Assigning roles to states**
- **defining permitted transitions and permitted roles to take them**
- **Each state should have a unique name**
  - used for naming transitions
- **roles, developers should set participant and coordinator**

Practical issues

- **Unique Naming for states and transitions**
- **Sharing data across clients**
  - Serialization (SMPC: JSON vs other: Pickle)

State: Extending the AppState class

- **Extend AppState class to define states**
  - implement two abstract methods: register and run
- **Registering transitions**
  - register all possible transitions
    - self.register_transition(target, role, name)
    - just a declaration of transitions
    - eligibility of the transition will be checked.

register(self)

- **Computations**
  - All the operations
  - Role base set of operations to handle.
  - Call communication methods
  - Logging
  - Updating operational states
  - report progress.

run(self)

State: Communication methods

Sending Data:

- **Communicating Data to other clients:**
  - `Send_data_to_participant`
    - `data`
    - `destination`
  - Communicate any serializable data using Pickle

- **Communicating data to the coordinator:**
  - `send_data_to_coordinator`
    - `data`
    - `send_to_self=True`
    - `use_smpc=False`

- **Broadcasting data:**
  - `broadcast_data`
    - `data`
    - `send_to_self=True`
  - Only for the coordinator

State: Communication methods

Waiting to receive data:

```
await_data(self, n: int = 1, unwrap=True, is_json=False)
```

- **N pieces of data**
- **unwrap=True**
  - Deserialize or not
  - More flexibility, less transparency
- **Is_json**
  - Either data is serialized using JSON
  - Either SMPC was used or not
- **poll for data arrival**
  - DATA_POLL_INTERVAL

- **Returns**
  - list of data
    - For data of more than one client
    - When no deserialization ir required
  - single data piece
    - deserialization of data for one client
- **Used by all receive methods**

State: Communication methods

Receiving data:

- **Gathering clients data**
  - Only coordinator
  - For all clients' data
  - Without Aggregation

```
gather_data(self, is_json=False)
```

- **Aggregating clients data**
  - automatically handles SMPC
  - the same data structure and shape as the one was sent out
  - Structural and data consistency
  - SMPC usage:
    - Using SMPC: looks like waiting for just one client.
    - Without SMPC: waits for all clients

```
aggregate_data(self, operation:
SMPCOperation, use_smpc=False)
```

Simple initial state

- **First state in any FC app**
- **register method:**
  - introducing transition to terminal state
    - last state in FC app
    - works as a flag to show the end of app execution
- **run method**
  - executes all the local/global computation
  - Data communication and I/O
  - Transitions
  - Reporting

```
@app_state(name='initial')
class InitialState(AppState):
    def register(self):
        self.register_transition('terminal')

    def run(self):
        self.log('Hello World!')
         return 'terminal'
```

Developing Federated Applications

FeatureCloud CLI

FeatureCloud: Creating an app

```
$ featurecloud app new <APP_NAME> <template_name>
```

- **APP_NAME:**
  - a desired app name
  - if not provided, the name of the containing directory will be used
- **template_name:**
  - one of the provided templates by FC GitHub repositories can be used
  - Default app-blank
    - Sort of hello-world template
    - Only includes initial state

FeatureCloud: Creating an app

```
$ featurecloud app new -app-name fc_test -template-name app-blank
```

```
fc_test
├── Dockerfile
├── LICENSE
├── main.py
├── README.md
├── requirements.txt
├── states.py
└── server_config
    ├── docker-entrypoint.sh
    ├── nginx
    └── supervisord.conf
```

FeatureCloud: Creating an app

```
$ featurecloud app new -app-name fc_test -template-name app-blank
```

```
fc_test
├── Dockerfile
├── LICENSE
├── main.py
├── README.md
├── requirements.txt
├── states.py
└── server_config
    ├── docker-entrypoint.sh
    ├── nginx
    └── supervisord.conf
```

App deployment

```
$ featurecloud app build <app-name>
```

- app-name:
  - developed app name
- The app's docker image will be created based on Dockerfile, requirement, and the app implementation

```
$ featurecloud app publish -name fc_test -tag latest
```

- publishing apps in AI-store
- pushing apps image into the FC docker repository
  - All app names should start with featurecloud.ai/
  - fc_test is not valid
  - either build the app with another name or tag it

App execution on the testbed using CLI

Running an app as a testrun in FC testbed

- **start**
- **stop**
- **logs**
- **list**
- **delete**
- **info**
- **trafic**

Developing Federated Applications

Hands on federated apps **(~25 minutes)**

1. Develop Hello world application
   a. one state to greet the world!
2. Develop a Mean app
   a. clients send out a number
   b. coordinator average the received values
   c. coordinator broadcasts the mean to clients

Developing Federated Applications

Hands on federated apps (Homework): federated regression

1. Load the datasets
   a. read input files: mnt/input/
   b. write output files: mnt/output/
2. local update: apply regression on sklearn diabetes dataset
   a. sklearn.datasets.load_diabetes
3. global update: aggregated the models and broadcast the model
4. train until achieving the centralized training performance
5. Join FeatureCloud community on Slack: feel free to ask questions!

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ISMB 2022 - Tutorial - Federated Learning in Biomedicine

Thank you!