**T.R.**

**GEBZE TECHNICAL UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF COMPUTER ENGINEERING**

# A FEDERATED LEARNING PLATFORM ON RASPBERRY PI FOR IMAGING

NEVZAT SEFEROĞLU

SUPERVISOR
DR. YAKUP GENÇ

GEBZE
2023

**T.R.**
**GEBZE TECHNICAL UNIVERSITY**
**FACULTY OF ENGINEERING**
**COMPUTER ENGINEERING DEPARTMENT**

# A FEDERATED LEARNING PLATFORM ON RASPBERRY PI FOR IMAGING

**NEVZAT SEFEROĞLU**

SUPERVISOR
DR. YAKUP GENÇ

**2023**
**GEBZE**


GRADUATION PROJECT
JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 19/01/2023 by the following jury.

**JURY**

Member
(Supervisor) : Dr. Yakup GENÇ

Member : Dr. Yakup GENÇ

Member : Prof. Yusuf Sinan AKGÜL

# ABSTRACT

Federated Learning is a machine learning approach that allows models to be trained across multiple decentralized devices or servers holding local data samples, without exchanging the data itself. It's a strategy aimed at improving privacy and efficiency in machine learning, particularly in cases where sensitive data is involved, such as in healthcare or personal devices.

In a typical federated learning cycle, the central server distributes the model to all participating devices. Each device then trains the model with its local data and sends the updated models back to the server. The server then aggregates these models (often by taking a weighted average) to produce a global model, which is then distributed back to the devices in the next cycle.

Flower is an open-source framework designed to facilitate the process of Federated Learning. Its main goal is to enable developers to implement federated learning systems quickly and easily.

Flower provides the necessary tools and abstractions for managing the communication between the server and the devices (which it calls clients), the distribution and aggregation of models, and the general orchestration of the training process. It is built to be robust, scalable, and flexible, with support for various machine learning libraries such as TensorFlow and PyTorch.

The project aim combining of those two useful technology to into API which can deploy the given the given project source to the remote host machine without dealing with any direct **ssh** connection.

The service relies on infrastructure-as-code practice which is fulfilled with a library called **Ansible**.

**Keywords:** Federated Learning, FastAPI, Ansible, containerization.

# ÖZET

Federated Learning, modellerin yerel veri örneklerini tutan çoklu merkezi olmayan cihazlarda veya sunucularda eğitilmesine izin veren bir makine öğrenmesi yaklaşımıdır, ancak veriyi kendisi alışverişi yapmaz. Bu, özellikle sağlık hizmetleri veya kişisel cihazlar gibi hassas verilerin söz konusu olduğu durumlarda, makine öğrenmesinde gizlilik ve verimliliği artırmaya yönelik bir stratejidir.

Tipik bir federated learning döngüsünde, merkezi sunucu modeli tüm katılan cihazlara dağıtır. Her cihaz daha sonra modeli yerel verileriyle eğitir ve güncellenmiş modelleri sunucuya geri gönderir. Sunucu daha sonra bu modelleri toplar (genellikle ağırlıklı bir ortalama alarak) ve bir global model üretir, bu model daha sonra bir sonraki döngüde cihazlara geri dağıtılır.

Flower, Federated Learning sürecini kolaylaştırmak üzere tasarlanmış açık kaynaklı bir çerçevedir. Ana hedefi, geliştiricilerin federated learning sistemlerini hızlı ve kolay bir şekilde uygulamalarını sağlamaktır.

Flower, sunucu ile cihazlar (bunlara client diyor) arasındaki iletişimi yönetmek, modellerin dağıtımı ve toplanması, ve eğitim sürecinin genel düzenlemesi için gerekli araçları ve soyutlamaları sağlar. Robust, ölçeklenebilir ve esnek olacak şekilde inşa edilmiştir, TensorFlow ve PyTorch gibi çeşitli makine öğrenmesi kütüphanelerini destekler.

Projenin amacı, bu iki faydalı teknolojiyi bir API'ye birleştirmektir, bu API verilen proje kaynağını herhangi bir doğrudan **ssh** bağlantısıyla uğraşmadan uzak ana makineye dağıtabilir.

Hizmet, **Ansible** adlı bir kütüphane ile yerine getirilen altyapı-kod-pratiğine dayanır.

**Anahtar Kelimeler:** Federated Learning, FastAPI, Ansible, containerization.

# ACKNOWLEDGEMENT

This endeavor would not have been possible without Dr. Yakut GENÇ. I am also thankful to the open-source community of Flower Framework, Ansible, FastAPI.

**Nevzat SEFEROĞLU**

# LIST OF SYMBOLS AND ABBREVIATIONS

| **Symbol or Abbreviation** | : | **Explanation** |
|---|---|---|
| FastAPI | : | Python API ASGI library |
| Ansible | : | RedHat Automation library |
| Federated Learning | : | Machine learning technique which preserves privacy |

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1. FL-SERVICE

This section contains general overview of the project and explains its design principal with diagram created with excalidraw. The entire project consist of two program; the one is the API implementation the other is CLI client for the implemented API.

You can reach the source codes of the implementations here in the GitHub link below.

- `https://github.com/nevzatseferoglu/fl-service`.
- `https://github.com/nevzatseferoglu/flsvc`.

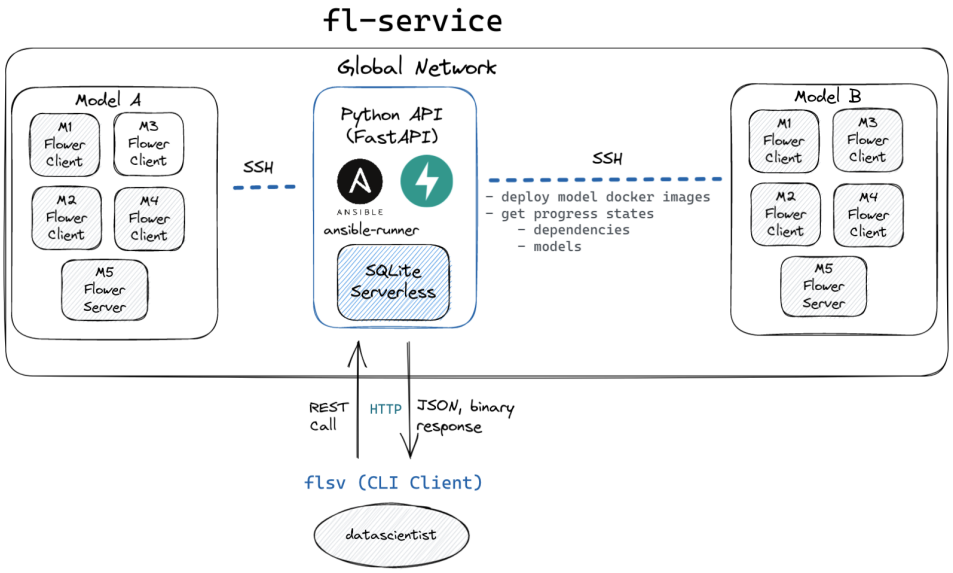


Figure 1.1: fl-service architecture diagram.

## 1.1. Details of fl-service

fl-service is actually an **FastAPI** implementation. FastAPI is based on the concept of 'Python type hints', a new feature that was added in Python 3.5, which makes it possible to specify the type of a variable. This allows FastAPI to automatically generate data schemas, validate incoming data, produce helpful error messages, and even auto-generate request and response bodies in your API documentation.

Integration of FastAPI with Ansible offers a highly efficient mechanism to construct, deploy, and manage web applications and APIs. FastAPI's utilization of Python

3.6+ type declarations and its capacity for asynchronous request handling ensure high performance and coding efficiency. Meanwhile, Ansible's declarative language facilitates streamlined configuration management and software provisioning. This amalgamation empowers me to create robust and scalable applications with a high degree of automation. The automatic generation of API documentation by FastAPI synergistically dovetails with Ansible's ability to automate application deployment, reducing the margin for human error and substantially enhancing operational efficiency.

In the FastAPI application I developed, I embedded Ansible and Ansible Runner as core components. FastAPI, thanks to its asynchronous nature, facilitated high-performance handling of RESTful API requests. Each request to specific endpoints is mapped to particular Ansible tasks through the Ansible Runner's Python API.

I implemented custom route handlers in FastAPI that use the Ansible Runner interface to invoke Ansible playbooks or ad-hoc tasks based on the incoming API request. The parameters for the tasks were extracted from the request payload, and the type validations were carried out by Pydantic models that FastAPI provides.

The tasks executed could range from infrastructure configuration, software provisioning to more complex orchestration tasks - all invoked directly through RESTful API calls. This effectively turned my FastAPI application into a management layer for various system tasks, exposing these functionalities through an HTTP-based API.

The results from the **Ansible** tasks were then collected and serialized into a response format to be returned as an HTTP response. This provides a real-time feedback mechanism for all Ansible operations conducted.

## 1.2. API Endpoints

API documentation can be reached from here; the doc is based on the commit trigger in GH pipeline. That means the the link below is always contains the latest reference of the API.

- `https://fl-service-api-doc.netlify.app/`.

Each subsection of the below explains design and functionality of the unique router.

### 1.2.1. SSH

To automate the remote host machine connection, public key exchange is required for the Ansible. After the key exchange happened, service endpoints can make automation on given IP Address.

- ***/ssh/generate-ssh-key-pair***

To make a key exchange, host machine which runs the API must have a key pair. When this endpoint triggered, it looks for an key pair, if it does not exist, it generates and puts under the **/.ssh** directory. If it does exist, it does not do anything, returns proper HTTP error to the owner of the request.

- ***/ssh/copy-ssh-key-to-remote-host***

Starts the key-exchange with given remote host. If the key exchange is made successfully, it records to the database. Otherwise, in any exceptional situation, returns a proper HTTP error.

The endpoint waits of its payload in the body of the request. Client can send all the credentials for the remote host connection and also federated learning related identifiers (such details about whether the deployed code is flower client or server).

**Note:** Due to network issues, there can be encountered with **timed out** error. The caller should retry the operation again. The situation is not always happening, it is all related to SSHD server configuration of the host machine and underlying routers which is used by the package.

### 1.2.2. Remote Host

This endpoint is provided for check the states of the recorded remote host machines recorded in database. Filtering and custom search can be made through this endpoinsts.

- ***/remote-hosts***

Returns the all remote host machine in the database.

- ***/remote-hosts/ip_address***

Filter the certain IP Address among the other ones. It is not possible to make a subnet filtering etc. But IP Address is validated before querying database for the security reason.

- ***/remote-hosts/contact-info/contact_info***

Returns all host machines which belong to given contact information as a list. There is upper limit which is already set as 100, it cannot be altered. Assigning the same contact info to more than 100 is a bad practice from the service perspective.

- ***/remote-hosts/fl-identifier/fl_identifier***

Returns all host machines which belong to given federated learning identifier as a list. There is upper limit which is already set as 100, it cannot be altered. Assigning the same contact info to more than 100 is a bad practice from the service perspective.

### 1.2.3. Docker

Before deploying, API needs to ensure that the docker daemon is up and running on the host machine. For achieving this, there is a few endpoints are provided to the API user.

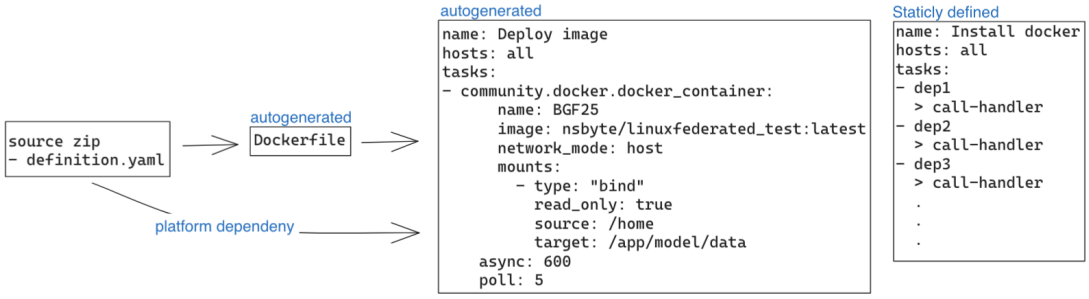


Figure 1.2: Docker architecture diagram.

- ***/docker/states/ip_address***

Returns a docker daemon dependencies states. Each dependency corresponds to a single task in the docker installation. After each task event handler invokes and update the database about the dependency table.

- ***/docker/install/ip_address***

Install Docker daemon in on the host machine according to given IP Address of it. Installation playbook is idempotent, the same endpoint can be invoked more than ones daemon state won't be changed.

- ***/docker/upload-source-files/ip_address/platform/arch/***

Creates a **dockerfile** according to given path variables. For example, if the given platform is pytorch is will use a pytorch base image. Then in the deployment state architecture of the machine will creates an image based on the given one.

Zipfile is uploaded with a certain MIME type, CLI does not implement deployment invocation but it can be doable.

```
`curl -X POST "http://localhost:8000/docker/upload-source-files/192.168.1.105/pytorch/amd64/" \
-H "Content-Type: multipart/form-data" \
-F file=@/Users/nevzatseferoglu/Desktop/flower/example-project/project-source.zip`
```

Figure 1.3: Image creation curl request example.

- ***/docker/deploy/ip_address/***

Deploy the already recorded image, if there is no image in the regarding unzipped source directory inside server host machine, it will return a HTTP error. If there is, it

will create a deployment playbook based on the information in the **definition.yaml**.

It contains essential information to create an autogenerated playbook.

```
entrypoint:
  - "python3"
  - "-u"
  - "./client.py"
sourcedir: /home
targetdir: /app/model/data
image_name: nsbyte/linuxfederated_test:latest
```

Figure 1.4: Example definition.yaml.

**entrypoint**: Represents an initial command which runs the deployed application in the container.

**sourcedir**: Represents which directory will be mounted into the docker container. It generally contains the model training data in it. It can be any directory in the host machine if the permission is satisfied, some directories cannot be mounted because of their nature.

Also, when the directory or file is mounted, it should not override the existing directory or a file in the container. If it happens, mounted one will override all the content and hides the existing content.

**targetdir**: Represents container destination directory which will be replaced with mounted one (sourcedir)

**image_name**: It is the name of the image, as a default it takes the image from the Docker registry because of the limitation in the Ansible, the given image should be located in some registry to deploy it.

Deployment file called deployment.yaml will be created in the source directory in the server host. Server looks into that file to deploy the image which is written in it.

Each invocation creates a unique container in the host, that means each deployment will be unique.

### 1.2.4. Ping

Uses **Ansible** ping module and check the health of remote SSH connection.

- ***/ping/ip_address***

Makes a PING protocol call with default Ansible ping module settings. If the return HTTP code is 200, remote host SSHD is up and running.

# 2. FLSVC COMMAND LINE CLIENT FOR FL-SERVICE

## 2.1. Details of flsvc

In the landscape of software development, the judicious choice of programming language and supporting frameworks is of paramount importance. In this context, my predilection for the Go language, coupled with the Cobra framework, to implement a Command Line Interface (CLI) arises from a careful analysis of their distinctive features and advantages.

The Go language, with its statically-typed, compiled nature, has surfaced as a highly efficacious tool in CLI development. The syntax of Go is parsimonious, aiding in the construction of lucid, efficient code. This characteristic amplifies its suitability for CLI applications, where brevity and performance are held at a premium.

Furthermore, Go's rich standard library propounds an assortment of useful packages for managing operating system interactions and command-line flags, enhancing the range and capability of the resulting CLI application. In addition, the inclusion of robust concurrency primitives like goroutines and channels in Go empowers developers with the ability to create multitasking capabilities in their CLI applications, thereby increasing efficiency and responsiveness.

Complementing Go in this endeavor is the Cobra framework, a potent library specifically designed for creating CLI in Go. With built-in commands, flags, and arguments, Cobra economizes the development process and aids in creating high-quality, consistent, and user-friendly CLIs. Its capacity to generate help commands and auto-completion scripts further refines the user experience and interactivity of the CLI application.

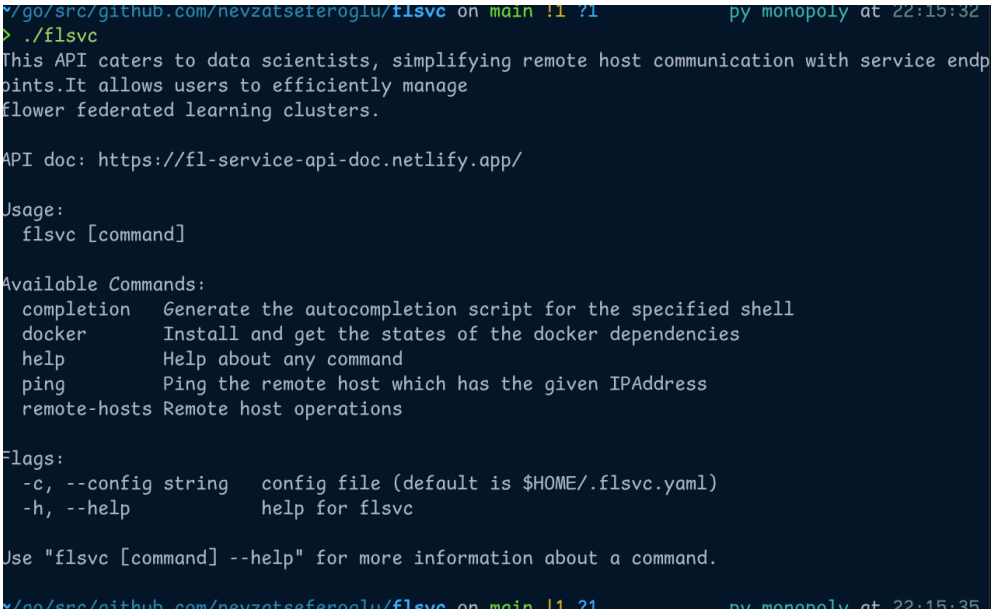


Figure 2.1: Help page of flsvc

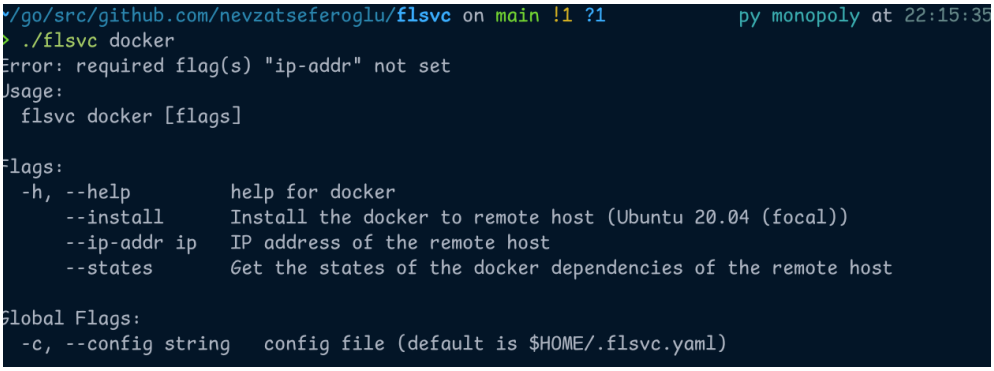


Figure 2.2: Help page of flsvc docker


Figure 2.3: Help page of flsvc remote-host

```
> ./flsvc ping
Error: required flag(s) "ip-addr" not set
Usage:
  flsvc ping [flags]

Flags:
  -h, --help         help for ping
      --ip-addr ip   IP address of the remote host

Global Flags:
  -c, --config string   config file (default is $HOME/.flsvc.yaml)
```

Figure 2.4: Help page of flsvc ping

You can also configure destination URL and port address with configuration file of the application. Configuration path can be set by the CLI while executing commands.

```
< ./flsvc remote-hosts --ip-addr 192.168.1.105
{
  "contact_info": "nevzatseferoglu@gmail.com",
  "description": null,
  "fl_identifier": "test_model",
  "ip_address": "192.168.1.105"
}
```

Figure 2.5: Response of flsvc filtering

```
~/go/src/github.com/nevzatseferoglu/flsvc on main !1 ?1
< ./flsvc docker --states --ip-addr 192.168.1.105
{
  "state_add_docker_gpg_apt_key": "ok",
  "state_add_docker_repository": "ok",
  "state_check_docker_command": "ok",
  "state_install_aptitude": "ok",
  "state_install_docker_module_for_python": "ok",
  "state_install_required_system_packages": "ok",
  "state_update_apt_and_install_docker_ce": "ok"
}
```

Figure 2.6: Response of flsvc docker states

## 2.2. Database view

| id | host_id | state_install_aptitude | state_install_required_system_packages | state_add_docker_gpg_apt_key | state_add_docker_repository | state_update_apt_and_install_docker_ce | state_install_docker_module_for_python | state_check_do |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | ok | ok | ok | ok | ok | ok | ok |

Figure 2.7: View of the database, remote host table

| id | host_id | state_install_aptitude | state_install_required_system_packages | state_add_docker_gpg_apt_key | state_add_docker_repository | state_update_apt_and_install_docker_ce |
|---|---|---|---|---|---|---|
| 1 | 1 | ok | ok | ok | ok | ok |

Figure 2.8: View of the database, dependency table

**I exploited pydantic, because of the reasons below;**

**Validation and Serialization:** Pydantic is a data validation library that uses Python type annotations. It enables the validation of complex data structures with ease and elegance. With Pydantic, you can ensure that the data you're receiving in your API is of the expected format, and it simplifies the serialization process by offering a way to create models that automatically convert your data into Python's native data types.

FastAPI integrates seamlessly with Pydantic to allow automatic request validation and serialization. This means you can define Pydantic models for your request bodies and FastAPI will automatically validate incoming JSON requests and return detailed error messages in the case of invalid data.

**Performance:** Both FastAPI and Pydantic are very fast. FastAPI is one of the fastest Python frameworks available, only slightly slower than NodeJS and Go. Pydantic's validation is also quite quick, making the combination of FastAPI and Pydantic a great choice when performance is a priority.

**Robustness and Error Handling:** Pydantic provides robust data validation which, when combined with FastAPI's error handling mechanisms, makes for very reliable applications. Errors are caught early and responded to with clear, informative error messages.

**Asynchronous Programming:** FastAPI supports asynchronous request handling, which means it can handle many requests concurrently without blocking, leading to higher throughput. Pydantic models can be used with these asynchronous handlers without issues.

# 3. CONCLUSIONS

In conclusion, this thesis has presented a comprehensive exploration of FastAPI, Ansible, and the flower federated learning framework, all combined in an innovative manner to provide a streamlined solution for deploying desired images to a host machine. In this solution, the user can simply make a REST call, effectively abstracting away the complexities and intricacies associated with containerization, providing a significant ease of use.

I delved into the utilization of Ansible, a potent automation tool, that has been effectively integrated with ansible-runner, to execute the playbook programmatically. This integration has been shown to have vast potential in significantly improving deployment efficiencies, while maintaining a high level of reliability and consistency. It showcased Ansible's flexibility and capacity to seamlessly integrate with other software, in this case, ansible-runner, to enhance the automation process.

Moreover, the implementation of FastAPI within the architecture demonstrated its high performance and its user-friendly characteristics, further simplifying the process for end-users. The addition of the flower federated learning framework allowed for efficient, scalable learning, paving the way for a more decentralized model training, thus harnessing the power of modern machine learning paradigms.

As we navigate through the ever-evolving landscape of software deployment and machine learning, this thesis has demonstrated how the combination of robust automation tools and cutting-edge learning frameworks can bring about notable improvements in deployment processes and machine learning models. The findings highlight the increasing importance of automation and federated learning, and suggest potential avenues for future research and innovation. This could potentially open new horizons for both the machine learning community and DevOps, promising an efficient, reliable, and streamlined future.

**??** [1]–[4].

# BIBLIOGRAPHY


[1] S. Hykes, *Docker: An open source platform for distributed applications for developers and sysadmins*, `https://www.docker.com/`, 2013.

[2] S. Ramírez, *Fastapi: A modern, fast (high-performance), web framework for building apis with python 3.6+*, `https://fastapi.tiangolo.com/`, 2018.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[4] F. contributors, *Flower: A friendly federated learning framework*, `https://flower.dev/`, 2020.