

Instructions for the deployment of FEDn

Teacher Team of DE-II

May 2023

The instructions are aimed to deploy FEDn in the NAISS cloud with the example MNIST application in a fully distributed manner. We will deploy FEDn in 3 virtual machines:

VM a: *Base services* and the *reducer*

VM b: One *combiner*

VM c: Two *clients*

All modules will be running in Docker containers and orchestrated by Docker compose.

Note 1: Please check the syntax before executing them if you copy the codes here directly to your shell (copying texts from PDF files sometimes leads inconsistent syntax).

Note 2: All commands in the instructions are supposed to be run in virtual machines as *ubuntu* user (not *root*, unless specified).

1 Basic Contextualization

First we need to start three virtual machines of medium flavour with operating system "Ubuntu 22.04 - 2023.01.07" in the NAISS cloud and do the basic contextualization, including installing Docker, solving the MTU problem in the NAISS cloud and post-installation steps of Docker Engine.

1.1 Installation of Docker

We install Docker Engine with the official installation bash scripts provided in the link here.

```
$ sudo apt update
$ wget
  ↪ https://raw.githubusercontent.com/docker/docker-install/master/install.sh
  ↪ -O install_docker.sh
$ chmod +x install_docker.sh
$ ./install_docker.sh
```

1.2 Fixing the MTU problem

The MTU problem is that the MTU size of the NAISS cloud is smaller than the default MTU size of Docker networks, which may cause internet inaccessibility of Docker containers. This is a common problem in cloud platforms. Here we are fixing the MTU problem of the **bridge** network of Docker, which is used during the building stage for docker images.

```
$ echo '{"mtu":1450}' | sudo tee /etc/docker/daemon.json
```

Note: Now the updated `daemon.json` has *not* been reloaded yet. Please continue to follow Section 1.3.

1.3 Post-installation steps

Now we want to grant Docker privileged permission so we can run Docker as *ubuntu* user, following the official instructions here as follows.

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ sudo systemctl restart docker
```

Then exit your virtual machine and reconnect to it via **ssh**. Now the contextualization of the virtual machines for FEDn is finished. Please follow Section 1.4 to confirm it.

1.4 Self check and get FEDn

To confirm Docker is installed with privileged permission, you could run `docker run hello-world` as *ubuntu* user.

To confirm the MTU problem has been fixed, you could run `docker network inspect bridge` to check if the MTU size has been changed to 1450 as Figure 1.

Finally, clone the source code of FEDn to your virtual machine.

```
$ git clone https://github.com/scaleoutsystems/fedn.git
```

2 Deployment of FEDn

Now we have three contextualized virtual machines, we can start to deploy FEDn to the VMs. Recall that we want to have VM a for the base services and the reducer, VM b for the combiner and VM c for two clients, we note the IP address of the three VMs as `<ipa>`, `<ipb>` and `<ipc>` respectively. Please replace the internal IP addresses of your own VMs correspondingly in the following codes.

```
[
  {
    "Name": "bridge",
    "Id": "923f910c789c50797516f3b65dd12433ae7363847e66f38faa6d840f5547dd98",
    "Created": "2023-05-09T07:36:12.824717776Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1450"
    },
    "Labels": {}
  }
]
```

Figure 1: Expected information of fixed MTU in the **bridge** network of Docker

2.1 VM a

Here to deploy the base services and the reducer to VM a, three modifications in the FEDn example need to be done: modify the docker compose file `docker-compose.yaml`, authenticate the combiner, and create configuration files.

2.1.1 Docker compose file

First, we need to modify `docker-compose.yaml`, to fix the MTU problem of the new-created network for the running stage. Modify the **network** field of `docker-compose.yaml` as following and keep other parts the same as the example.

```
# Setup network
networks:
  default:
    name: fedn_default
```

```
driver_opts:
  com.docker.network.driver.mtu: 1450
```

2.1.2 Authenticate combiner

We need to create a new file named `./config/extra-hosts-reducer.yaml` with following contents:

```
version: '3.3'

# Map the "host" parameter from combiner-settings.yaml to the
↪ combiner host IP
# Make one entry for each combiner in the network.
services:
  reducer:
    extra_hosts:
      combiner: <ipb> # combiner IP address
```

Remember to replace `<ipb>` with the internal IP of your combiner VM. You can copy the template file in `./config/` and modify it.

2.1.3 Configure reducer

Then we need to create a file named `./config/settings-reducer.yaml` with following contents:

```
network_id: fedn-test-network
token: fedn_token

control:
  state: idle
  helper: keras

statestore:
  type: MongoDB
  mongo_config:
    username: fedn_admin
    password: password
    host: <ipa>
    port: 6534

storage:
  storage_type: S3
  storage_config:
    storage_hostname: <ipa>
    storage_port: 9000
    storage_access_key: fedn_admin
```

```
storage_secret_key: password
storage_bucket: fedn-models
context_bucket: fedn-context
storage_secure_mode: False
```

Remember to replace `<ipa>` with the internal IP of your current VM (since this will be also used by the combiners, the exact ip address is required and `localhost` will not work here). Likewise you can copy the template file in `./config/` and modify it.

2.1.4 Start the services

Now that we have finished all modifications, we run following commands to start the services:

```
$ docker compose -f docker-compose.yaml -f
↪ config/extra-hosts-reducer.yaml -f
↪ config/reducer-settings.override.yaml up minio mongo
↪ mongo-express reducer
```

To run services in the background, use following command:

```
$ docker compose -f docker-compose.yaml -f
↪ config/extra-hosts-reducer.yaml -f
↪ config/reducer-settings.override.yaml up -d minio mongo
↪ mongo-express reducer
```

To confirm that your reducer and the base services are running, you could check the page `<floating_ip_a>:8090` with your browser.

2.2 VM b

Here we deploy the combiner to VM b. Two modifications need to be done on the example application of FEDn: fix the MTU problem in `docker-compose.yaml` and configure the combiner.

2.2.1 Docker compose file

Exactly the same as we did in VM a, we need to modify the `network` field of `docker-compose.yaml` as following and keep other parts the same as the example.

```
# Setup network
networks:
  default:
    name: fedn_default
    driver_opts:
      com.docker.network.driver.mtu: 1450
```

2.2.2 Configure combiner

Then we need to create a file named `./config/settings-combiner.yaml` with following contents, to make the combiner "know" how to connect to the reducer:

```
network_id: fedn-test-network
controller:
  discover_host: <ipa>
  discover_port: 8090

combiner:
  name: combiner
  host: <ipb>
  port: 12080
  max_clients: 30
```

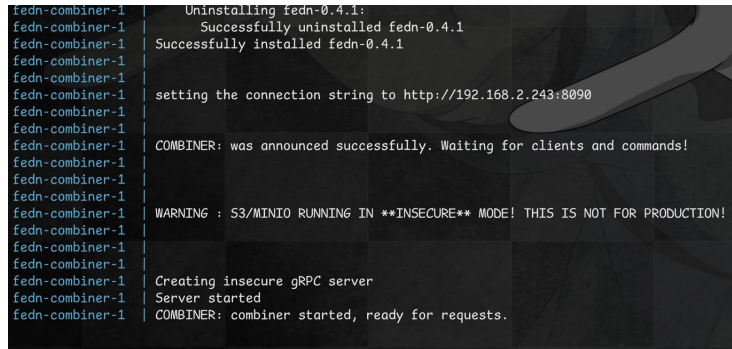
Remember to replace `<ipa>` and `<ipb>` to the internal IP address of the VM where the reducer is deployed.

2.2.3 Start the combiner

To start the combiner, we execute the following command:

```
$ docker compose -f docker-compose.yaml -f
↪ config/combiner-settings.override.yaml up combiner
```

You could see the output as shown in Figure 2 if the combiner is running properly.

A terminal window showing the output of the command to start the combiner. The output is as follows:

```
fedn-combiner-1 | Uninstalling fedn-0.4.1:
fedn-combiner-1 |   Successfully uninstalled fedn-0.4.1
fedn-combiner-1 | Successfully installed fedn-0.4.1
fedn-combiner-1 |
fedn-combiner-1 | setting the connection string to http://192.168.2.243:8090
fedn-combiner-1 |
fedn-combiner-1 | COMBINER: was announced successfully. Waiting for clients and commands!
fedn-combiner-1 |
fedn-combiner-1 | WARNING : S3/MINIO RUNNING IN **INSECURE** MODE! THIS IS NOT FOR PRODUCTION!
fedn-combiner-1 |
fedn-combiner-1 |
fedn-combiner-1 | Creating insecure gRPC server
fedn-combiner-1 | Server started
fedn-combiner-1 | COMBINER: combiner started, ready for requests.
```

Figure 2: Expected output of a running combiner

Or to run it in the background, use the following command:

```
$ docker compose -f docker-compose.yaml -f
↪ config/combiner-settings.override.yaml up -d combiner
```

2.3 VM c

In VM c, we first need to generate the seed model, prepare the datasets and the compute package for the federated training. And then we want to deploy two FEDn clients to connect to the framework.

2.3.1 Prepare model, data and package

The preparation is done on the host virtual machine (not in containers). First we need to initialize a python virtual environment.

```
$ cd fedn/examples/mnist-pytorch/  
$ bin/init_venv.sh
```

Here you may need to replace `python` to `python3` in `init_venv.sh` and install `python3-venv` if required.

Then do

```
# working directory: fedn/examples/mnist-pytorch/  
$ bin/get_data  
$ bin/split_data  
$ bin/build.sh
```

The generated files are `package.tgz` and `seed.npz`. Then you need to download these two files to your *local* machine (use `scp`) and we will use them later with the browser.

2.3.2 Configure and run the clients

Then we start to run client containers in VM c.

First, like the reducer and the combiner, we first fix the MTU problem in `docker-compose.yaml` (refer to Section 2.1.1). Then to run the client containers, we need to configure the setting to let them "know" how to connect to the reducer and the combiner (like we did before). We need to first create a file named `fedn/config/settings-client.yaml` with following content:

```
network_id: fedn-test-network  
discover_host: <ipa>  
discover_port: 8090
```

Again you can copy the template file and modify it.

Then we need to modify the file `fedn/examples/mnist-pytorch/docker-compose.override.yaml`, by adding the ip address of your reducer and combiner in the docker compose override file.

```
# Compose schema version  
version: '3.3'
```

```

# Overriding requirements
services:
  client:
    build:
      args:
        REQUIREMENTS: examples/mnist-pytorch/requirements.txt
    deploy:
      replicas: 2
    volumes:
      - ${HOST_REPO_DIR:-.}/fedn:/app/fedn
      - ${HOST_REPO_DIR:-.}/examples/mnist-pytorch/data:/var/data
      - /var/run/docker.sock:/var/run/docker.sock
    extra_hosts:
      reducer: <ipa> # reducer IP address
      combiner: <ipb> # combiner IP address

```

Then we can start the client containers:

```

# working directory: fedn/
$ docker compose -f docker-compose.yaml -f
↪ examples/mnist-pytorch/docker-compose.override.yaml up
↪ client

```

To run client containers in the background, do

```

# working directory: fedn/
$ docker compose -f docker-compose.yaml -f
↪ examples/mnist-pytorch/docker-compose.override.yaml up -d
↪ client

```

3 Run the example

Then we have our cluster deployed, we can start the federated training for the example.

First we navigate to the dashboard of FEDn with your local browser with address <floating-ip.a>:8090. Here you need to upload your compute package and seed model first. You have downloaded them in Section 2.3.1. As we are using `pytorch` for the training, here please select `pytorch` as the helper.

After you've uploaded required files, you can access the full functions of the dashboard. First you could make sure the framework is up as we expected in the Network tab. The expected architecture is shown as Figure 3.

Then you could navigate to the Control tab and start the federated training. Figure 4 shows the expected output.

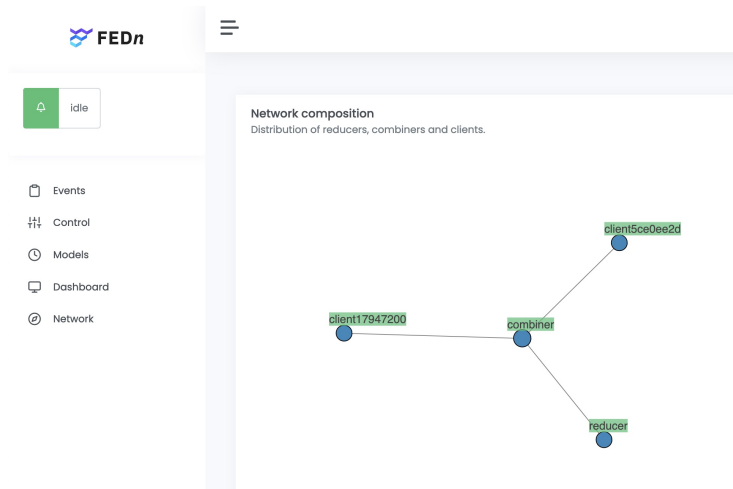


Figure 3: Expected architecture of the FEDn framework



Figure 4: Expected output when FEDn is training models