Liberty Framework
User's Guide

Franck Blettner

Table of Contents

1. Getting Started

Learn the basics of Liberty Framework.

Liberty Framework

Welcome to **Liberty Framework**, a **no-code development platform** designed for rapid and efficient web application creation using the latest in **React**, **Node.js**, and **PostgreSQL** technologies. Whether you're a developer or a non-technical user, Liberty Framework empowers you to build robust applications with **zero coding skills** required.

Go to Demo

```
Login = demo
Password = demo
Appplication = LIBERTY, NOMASX-1 and NOMAJDE
```

Documentation

Download the complete Liberty Framework User Guide in PDF format:

Download Liberty Framework User Guide

2. Release Notes

See what's new in the latest release.

3. Installation

Step-by-step installation guides.

3.1. Architecture

Understand the architecture of Liberty Framework.

This document provides an overview of the functionality and configuration of the services within the **Liberty Framework**, including **Node.js**, **PostgreSQL**, **pgAdmin**, **Airflow**, **OIDC**, and **Gitea**. These services are integrated with **Traefik** as a reverse proxy, enabling both HTTP and HTTPS access with automated routing.

---

1. Node.js Service (`liberty-node`) 🟢

- **Image**: `ghcr.io/fblettner/liberty-node:latest`
- **Command**: Runs the Node.js app (`app.js`) on port `3002` .
- **Security Options**:
- 🔒 `label:disable` : Disables SELinux labels.
- ⚙️ `cap_drop` : Removes unnecessary Linux capabilities like `MKNOD` and `AUDIT_WRITE` .
- **Networks**: Connected to the `liberty-network` .
- **Working Directory**: `/opt/liberty`
- **Depends on**: PostgreSQL (`pg`) service.
- **Traefik Configuration**:
- 🌐 **API Routing**: HTTP and HTTPS routing for `/api` using `PathPrefix` .
- 📡 **Socket Routing**: HTTP and HTTPS routing for `/socket` and `/socket.io` .
- ⚛️ **React Application**: Handles HTTP and HTTPS routing for the React app with a middleware for error pages.
- 🚀 **Compression**: `compress-middleware` applied to several routes for better performance.
- 🔌 **Port Configuration**: Node.js runs on port `3002` .

---

2. PostgreSQL Service (`liberty-pg`) 🐘

- **Image**: `ghcr.io/fblettner/liberty-pg:latest`
- **Command**: Runs the PostgreSQL server with optimized settings for performance:
- `shared_buffers=2GB`
- `track_activity_query_size=1MB`
- `work_mem=256MB`
- `maintenance_work_mem=128MB`
- Other configurations to optimize WAL size, checkpoint timing, and costs.
- **Volumes**: Data stored in the `pg-data` volume.

- **Networks**: Connected to `liberty-network`.
- **Traefik Configuration**:
- 🛠️ **TCP Router**: Routes PostgreSQL traffic via `db` entry point.
- 🔌 **Port**: Exposed on port `5432`.

---

3. pgAdmin Service (`liberty-pgadmin`) 🖥️

- **Image**: `ghcr.io/fblettner/liberty-pgadmin:latest`
- **User**: Root privileges enabled.
- **Volumes**: pgAdmin data stored in the `pgadmin-data` volume.
- **Environment**: Sets the `SCRIPT_NAME=/pgadmin` for pgAdmin web access.
- **Depends on**: PostgreSQL (`pg`).
- **Networks**: Connected to `liberty-network`.
- **Traefik Configuration**:
- 🌐 **HTTP Router**: Routes requests for `/pgadmin`.
- 🔌 **Port**: Exposed on port `3003`.

---

4. Airflow Service (`liberty-airflow`) 🛠️

- **Image**: `ghcr.io/fblettner/liberty-airflow:latest`
- **Security Options**:
- 🔒 Disables SELinux labels.
- ⚙️ Drops capabilities `MKNOD` and `AUDIT_WRITE`.
- **Volumes**:
- Logs stored in the `airflow-logs` volume.
- **Depends on**: PostgreSQL (`pg`), Gitea (`gitea`).
- **Networks**: Connected to `liberty-network`.
- **Traefik Configuration**:
- 🌐 **Routing**: Handles HTTP and HTTPS requests for `/airflow/home`.
- ⚠️ **Error Pages Middleware**: Applied to both HTTP and HTTPS routes.
- 🔌 **Port**: Exposed on port `8080`.

---

5. OIDC Service (`liberty-keycloak`) 🔐

- **Image**: `ghcr.io/fblettner/liberty-keycloak:latest`
- **Command**: Starts the Keycloak OIDC server with proxy headers and hostname settings.
- **Environment Variables**:
- 🔄 `PROXY_ADDRESS_FORWARDING`: Enables proxy address forwarding.
- 🌍 `KC_HOSTNAME_PATH` and `KC_HTTP_RELATIVE_PATH`: Configured to `/oidc`.
- **Depends on**: PostgreSQL (`pg`).
- **Networks**: Connected to `liberty-network`.
- **Traefik Configuration**:
- 🌐 **HTTP and HTTPS Routing**: Routes `/oidc` requests.
- 🔌 **Port**: OIDC runs on port `9000` (Keycloak internally uses port `8080`).
- 🌍 **CORS Middleware**: Configures Cross-Origin Resource Sharing (CORS) for all origins and credentials.

---

6. Gitea Service (`liberty-gitea`) 📂

- **Image**: `ghcr.io/fblettner/liberty-gitea:latest`
- **Healthcheck**: Ensures service health by checking `/` endpoint every 30 seconds.
- **Volumes**:
- Configuration and data in `liberty-gitea`.
- **Restart Policy**: Set to `unless-stopped`.
- **Networks**: Connected to `liberty-network`.
- **Traefik Configuration**:
- 🌐 **Routing**: Routes HTTP requests to `/gitea`.
- 🛠️ **Middleware**: Uses `stripprefix` to remove `/gitea` from the path for internal routing.
- 🔌 **Port**: Exposed on port `3000`.

Volumes 🗃️

- **node-logs**: Stores Logs for backend and frontend.
- **pg-data**: Stores PostgreSQL data.
- **pg-logs**: Stores Logs for database.
- **pgadmin-data**: Stores pgAdmin data.

- **liberty-gitea**: Stores gitea config and data.
- **airflow-logs**: Stores logs for Airflow.
- **airflow-dags**: Stores Dags for Airflow.
- **airflow-plugins**: Stores Plugins for Airflow.
- **traefik-certs**: Stores Traefik certificates (external).
- **traefik-config**: Stores Traefik configuration (external).
- **shared-data**: Stores shared data (external).

Networks 🌐

- **liberty-network**: External network for inter-service communication.

This configuration enables a scalable, containerized microservice architecture with **Node.js** for application logic, **PostgreSQL** for database management, **pgAdmin** for database administration, **Airflow** for automation, **Keycloak OIDC** for authentication, and **Gitea** for file management and versioning. **Traefik** serves as the reverse proxy, handling routing and applying security middleware for all services.

3.2. Docker Installation Guide

Set up Liberty Framework using Docker.

This guide covers the installation of Docker and Docker Compose on **CentOS** and **Amazon Linux**. Follow the respective instructions based on your environment.

Docker Installation for CentOS

Prerequisites

- CentOS 8 or higher
- Root or sudo access
- Minimum 2GB of RAM recommended, 8GB of RAM recommended for all Liberty Framework Services.

Step 1: Update System Packages

Before starting the installation, update your system to ensure all packages are up-to-date.

```
sudo yum update -y
```

if Podman is installed, remove all packages, artifacts and containers storage

```
yum remove buildah skopeo podman containers-common atomic-registries docker container-tools
rm -rf /etc/containers/* /var/lib/containers/* /etc/docker /etc/subuid* /etc/subgid*
cd ~ && rm -rf /.local/share/containers/
```

Step 2: Install Required Dependencies

Install the necessary packages required to set up the Docker repository.

```
sudo yum install -y yum-utils
```

Step 3: Set Up the Docker Repository

Add the Docker repository to your CentOS system.

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

Step 4: Install Docker

Install Docker Engine, CLI, and Containerd.

```

```

```
sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-
plugin
```

Step 5: Start and Enable Docker

Start the Docker service and enable it to start on boot.

```
sudo systemctl start docker
sudo systemctl enable docker
```

Step 6: Verify Docker Installation

Verify the installation by running a test Docker container.

```
sudo docker run hello-world
```

If the container runs and displays a welcome message, Docker is installed correctly.

Step 7: Adding Your User to the Docker Group (Optional)

To run Docker commands without `sudo`, add your user to the Docker group.

```
sudo usermod -aG docker $(whoami)
```

Log out and log back in to apply the group changes.

Uninstall Docker

To remove Docker, the CLI, Containerd, and Docker Compose, use the following commands:

```
sudo yum remove docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
docker-ce-rootless-extras
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
```

Docker Installation for Amazon Linux OS

Prerequisites

- Amazon Linux or Amazon Linux 2
- Root or sudo access
- Minimum 2GB of RAM recommended, 8GB of RAM recommended for all Liberty Framework Services.

Step 1: Update System Packages

Before starting the installation, update your system to ensure all packages are up-to-date.

```
sudo yum update -y
```

Step 2: Install Docker

Install Docker using the Amazon Linux Extras & yum package manager.

```
sudo amazon-linux-extras install docker -y
```

Step 3: Start and Enable Docker

Start the Docker service and enable it to start on boot.

```
sudo systemctl start docker
sudo systemctl enable docker
```

Step 4: Verify Docker Installation

Verify the installation by running a test Docker container.

```
sudo docker run hello-world
```

If the container runs and displays a welcome message, Docker is installed correctly.

Step 5: Install Docker Compose

Download the current stable release of Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/releases/download/$(curl -s
https://api.github.com/repos/docker/compose/releases/latest | grep -Po '"tag_name": "\K.*?(?
=")')/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Apply executable permissions to the binary:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Verify that the installation was successful:

```
docker-compose --version
```

Step 6: Adding Your User to the Docker Group (Optional)

To run Docker commands without `sudo`, add your user to the Docker group.

```
sudo usermod -aG docker $(whoami)
```

Log out and log back in to apply the group changes.

Uninstall Docker

To remove Docker, the CLI, Containerd, and Docker Compose, use the following commands:

```
sudo yum remove docker
sudo rm -rf /var/lib/docker
sudo rm /usr/local/bin/docker-compose
```

Post installation Tasks

If you want to set a custom directory for docker and if you are running behind a proxy, the docker service must be modified

Edit the service: /lib/systemd/system/docker.service

```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd --data-root <CUSTOM_DIRECTORY> -H fd:// --
containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutStartSec=0
RestartSec=2
Restart=always
Environment="HTTP_PROXY=<PROXY_URL>"
Environment="HTTPS_PROXY=<PROXY_URL>"
```

If you want to change the default IP range (172.17.x.x) for docker Edit the file: /etc/docker/daemon.json

```
# Set the ip range according to your requirements
# bip is for the internal interface
# default-address-pools is for all new networks
{
  "bip": "172.26.0.1/16",
  "default-address-pools": [
    { "base": "172.27.0.0/16", "size": 24 }
  ]
}
```

Conclusion

You have successfully installed Docker and Docker Compose on your CentOS or Amazon Linux OS system. You can now begin deploying and managing your Docker containers for Liberty Framework.

References

- Docker Documentation
- AWS Documentation

3.3. Installation Tools Deployment Guide

Learn about tools for deploying Liberty Framework.

Prerequisites

Before we begin, ensure you have the following installed on your system:

1. **Docker** and **Docker Compose**: Installation instructions can be found here.
2. **Git**: Installation instructions can be found here.

Step 1: Logging into Docker

To access a private Docker registry, you'll need to authenticate with your Docker credentials.

1. Log in to Docker:

```
docker login
```

   Follow the prompts to enter your Docker username and password.

Step 2: Create a Directory for Deployment

Create a directory where you will download and store the Docker Compose file.

1. Open a terminal.
2. Create a new directory:

```
mkdir -p /app/liberty-admin
cd /app/liberty-admin
```

Step 3: Download the Docker Compose File

Next, download the Docker Compose file from the provided URL.

1. Using `curl`:

```
curl -L -o docker-compose.yml https://github.com/fblettner/liberty-public/blob/main/release/latest/liberty-admin.yml
```

2. Alternatively, using `wget`:

```
wget -O docker-compose.yml https://github.com/fblettner/liberty-public/blob/main/release/latest/liberty-admin.yml
```

Step 4: Deploy the Docker Container using Docker Compose

Once you have the `docker-compose.yml` file downloaded into your `liberty-admin` directory, use Docker Compose to deploy the container.

1. In the terminal, navigate to the `liberty-admin` directory (if not already there):

```
cd /app/liberty-admin
```

2. Deploy the Docker container:

```
docker-compose up -d
```

This command will pull the necessary images from the registry (if they are not already available locally) and start the containers in detached mode.

Step 5: Verify the Deployment

To ensure the deployment is successful, you can check the status of the containers.

1. List the running containers:

```
docker ps
```

You should see the following containers running as defined in the `docker-compose.yml` file:

- **traefik**: This service is managing routing and load balancing, and exposes several endpoints for web (port 3000), websecure (port 3443), dashboard (port 8080), and database (port 5432).
- **portainer**: This service provides a UI for managing Docker environments, accessible via paths prefixed with `/portainer`.
- **error-pages**: This service handles error pages and is available to respond to general HTTP requests.

Summary of Commands

```
# Log in to Docker
docker login

# Create and navigate to the admin directory
mkdir -p /app/liberty-admin
cd /app/liberty-admin

# Download the Docker Compose file
curl -L -o docker-compose.yml https://raw.githubusercontent.com/fblettner/liberty-
public/release/latest/liberty-admin.yml
```

```
# or using wget
wget -O docker-compose.yml https://raw.githubusercontent.com/fblettner/liberty-public/release/latest/liberty-admin.yml

# Deploy the Docker container
docker-compose up -d
```

Accessing Services

After deployment, you can access the services with the following URLs:

- **Traefik Dashboard**: Accessible at http://`<your_server_ip>`:8080/dashboard/ (authentication may be required).
- **Portainer**: Accessible at http://`<your_server_ip>`:3000/portainer or https://`<your_server_ip>`:3443/portainer.

Replace `<your_server_ip>` with the IP address or hostname of your server. Feel free to reach out if you have any further questions or run into any issues!

3.4. Liberty Deployment Guide

Guide to deploying Liberty Framework.

This guide will walk you through deploying Liberty Framework using Portainer, based on the Compose file located at the following URL: liberty-framework.yml.

Prerequisites

Before you begin, ensure the following prerequisites are met:

- You have Docker installed and running on your server. Installation instructions can be found here.
- You have Portainer installed and running on your server. Installation instructions can be found here.
- You have access to the Portainer web interface. The URL typically looks like `http://your-server-ip:3000` or `https://your-server-ip:3443`.

Accessing Portainer

1. Open a web browser and navigate to the Portainer web interface.
2. Log in with your Portainer credentials.
3. Set a password first time you log into Portainer

Logging into a Custom Registry

1. In the Portainer web interface, navigate to `Registries` from the sidebar.
2. Click on the `+ Add registry` button.
3. Provide the following details for your custom registry:
    - **Name:** A friendly name for your registry.
    - **URL:** The URL of your custom registry (e.g., `ghcr.io/fblettner`).
    - **Username:** Your registry username (this user will be provided by Nomana-IT).
    - **Password:** Your registry password (this token will be provided by Nomana-IT).
4. After filling in the details, click on the `Add Registry` button to save the registry.

Deploy the Stack

1. In the Portainer web interface, navigate to `Stacks` from the sidebar.
2. Click on the `+ Add Stack` button.
3. Provide a name for your stack in the `Name` field.
4. Under the `Git repository` tab:

- Enter the **Repository URL**:

```
https://github.com/fblettner/liberty-public
```

- In the **Compose path** field, specify:

```
release/latest/liberty-framework.yml
```

5. Scroll down and click on the `Deploy the stack` button.

Verify Deployment

1. Once the stack is deployed, navigate to `Containers` from the sidebar.
2. Verify that the containers listed in the Compose file are running.
3. Access the services through the designated ports to ensure everything is functioning as expected.

Alternative: Pull Docker Images from Terminal

If you prefer to pull Docker images directly from the terminal, you can do so using the following commands:

1. Open a terminal and log in to the custom registry:

```
docker login ghcr.io
```

When prompted, enter your username and password (token).

2. Pull the required Docker images manually:

```
docker pull ghcr.io/fblettner/liberty-node:latest
docker pull ghcr.io/fblettner/liberty-pg:latest
docker pull ghcr.io/fblettner/liberty-pgadmin:latest
docker pull ghcr.io/fblettner/liberty-rundeck:latest
docker pull ghcr.io/fblettner/liberty-keycloak:latest
docker pull ghcr.io/fblettner/liberty-filebrowser:latest
```

Steps for AWS Users

If you are using AWS and need to connect via AWS CLI, follow these steps:

1. Configure your AWS CLI:

```
aws configure
```

Follow the prompts to enter your AWS Access Key, Secret Access Key, default region name, and output format.

2. Log in to the AWS Elastic Container Registry (ECR):

```
aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin <your-aws-account-id>.dkr.ecr.eu-west-1.amazonaws.com
```

Replace `<your-aws-account-id>` with your actual AWS account ID.

Additional Resources

- Portainer Documentation
- Docker Compose Documentation
- GitHub Repository - liberty-framework.yml

By following this guide, you should be able to deploy Liberty Framework using Portainer seamlessly. If you run into any issues or have any questions, refer to the additional resources provided or reach out to the respective support communities.

Summary

URLs: - **Web Application:** `/` - **API:** `/api` - **PgAdmin:** `/pgadmin` - **Rundeck:** `/rundeck` - **OIDC:** `/oidc` - **Filebrowser:** `/filebrowser`

Services: - **node:** ghcr.io/fblettner/liberty-node:latest (Port 3002) - **pg:** ghcr.io/fblettner/liberty-pg:latest (Port 5432) - **pgadmin:** ghcr.io/fblettner/liberty-pgadmin:latest (Port 3003) - **rundeck:** ghcr.io/fblettner/liberty-rundeck:latest (Port 4440) - **oidc:** ghcr.io/fblettner/liberty-keycloak:latest (Port 8080) - **filebrowser:** ghcr.io/fblettner/liberty-filebrowser:latest (Port 80)

Details of all Liberty Framework Services can be found here.

3.5. Create Linux Services

Create Linux services for Liberty Framework.

This guide will walk you through creating systemd services to manage your Docker Compose deployments. This ensures that your services start automatically on boot and can be managed easily using standard systemd commands.

Prerequisites

Before you begin, ensure the following prerequisites are met:

- You have Docker and Docker Compose installed on your server.
- You have completed the deployment steps for Liberty Framework using Docker Compose.

Creating the Systemd Service for Admin Tools

1. Create a service file for `docker-admin`:

```
sudo nano /etc/systemd/system/docker-admin.service
```

2. Paste the following content into the file:

```
[Unit]
Description=Liberty Admin Tools Service
PartOf=docker.service
After=docker.service

[Service]
Type=simple
RemainAfterExit=true
WorkingDirectory=/app/liberty-admin/
ExecStart=/usr/local/bin/docker-compose -f /app/liberty-admin/docker-compose.yml start
ExecStop=/usr/local/bin/docker-compose -f /app/liberty-admin/docker-compose.yml stop

[Install]
WantedBy=multi-user.target
```

3. Save and close the file.

Creating the Systemd Service for Liberty Framework

1. Open a terminal.
2. Create a new directory:

```
mkdir -p /app/liberty-framework
cd /app/liberty-framework
```

3. Download the Docker Compose file from the provided URL, Using `curl`:

```
curl -L -o docker-compose.yml https://github.com/fblettner/liberty-public/blob/main/release/latest/liberty-framework.yml
```

4. Create a service file for `docker-liberty`:

```
sudo nano /etc/systemd/system/docker-liberty.service
```

5. Paste the following content into the file:

```
[Unit]
Description=Liberty Framework Service
PartOf=docker.service
After=docker.service

[Service]
Type=simple
RemainAfterExit=true
WorkingDirectory=/app/liberty/
ExecStart=/usr/local/bin/docker-compose -f /app/liberty-framework/docker-compose.yml start
ExecStop=/usr/local/bin/docker-compose -f /app/liberty-framework/liberty-compose.yaml stop

[Install]
WantedBy=multi-user.target
```

6. Save and close the file.

Enabling and Starting the Services

1. Enable the created services to start on boot:

```
sudo systemctl enable docker-liberty.service
sudo systemctl enable docker-admin.service
```

2. Start the services immediately:

```
sudo systemctl start docker-liberty.service
sudo systemctl start docker-admin.service
```

3. Check the status of the services to ensure they are running:

```
sudo systemctl status docker-liberty.service
sudo systemctl status docker-admin.service
```

Additional Resources

- Systemd Documentation

- Docker Documentation
- Docker Compose Documentation

By following this guide, you should be able to create and manage systemd services for your Docker Compose deployments seamlessly. If you run into any issues or have any questions, refer to the additional resources provided or reach out to the respective support communities.

3.6. Enable SSL with Traefik

Enable SSL using Traefik for enhanced security.

By default, SSL is enabled with a self signed certificate. You have to copy your own certificates according to your domain

Prerequisites:

- `mkcert` installed to create a new self-signed certificate.
- Certificates for your domain

Step 1: Copy your certificates files

1. Copy your certificates files to the server hosting Liberty Framework
2. Transfer you certificate to the Docker container

```
docker cp <your_certificate_directory>/cert.pem traefik:/etc/certs/cert.pem
docker cp <your_certificate_directory>/key.pem traefik:/etc/certs/key.pem
```

Final Administrator Note: Certificates must be transferred to the Docker container with each renewal

Step2: Create a self-signed certificate (optional)

1. Connect to the server hosting Liberty Framework
2. Create a new self signed certificate

```
mkcert -key-file ./certs/key.pem -cert-file ./certs/cert.pem '<server_name>'
```

3. Transfer you certificate to the Docker container

```
docker cp ./certs/cert.pem traefik:/etc/certs/cert.pem
docker cp ./certs/key.pem traefik:/etc/certs/key.pem
```

Final Administrator Note: After updating both files, it is required to restart the Traefik service to apply the new settings.

4. Nomasx-1

Guides and settings for Nomasx-1.

4.1. Administrator's Guide

Administrator resources and tools.

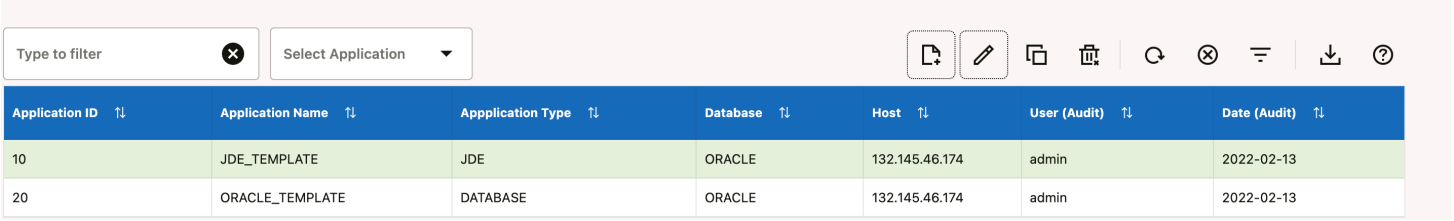
4.1.1. Global Settings

Manage global settings for Nomasx-1.

Global Settings

1. Applications

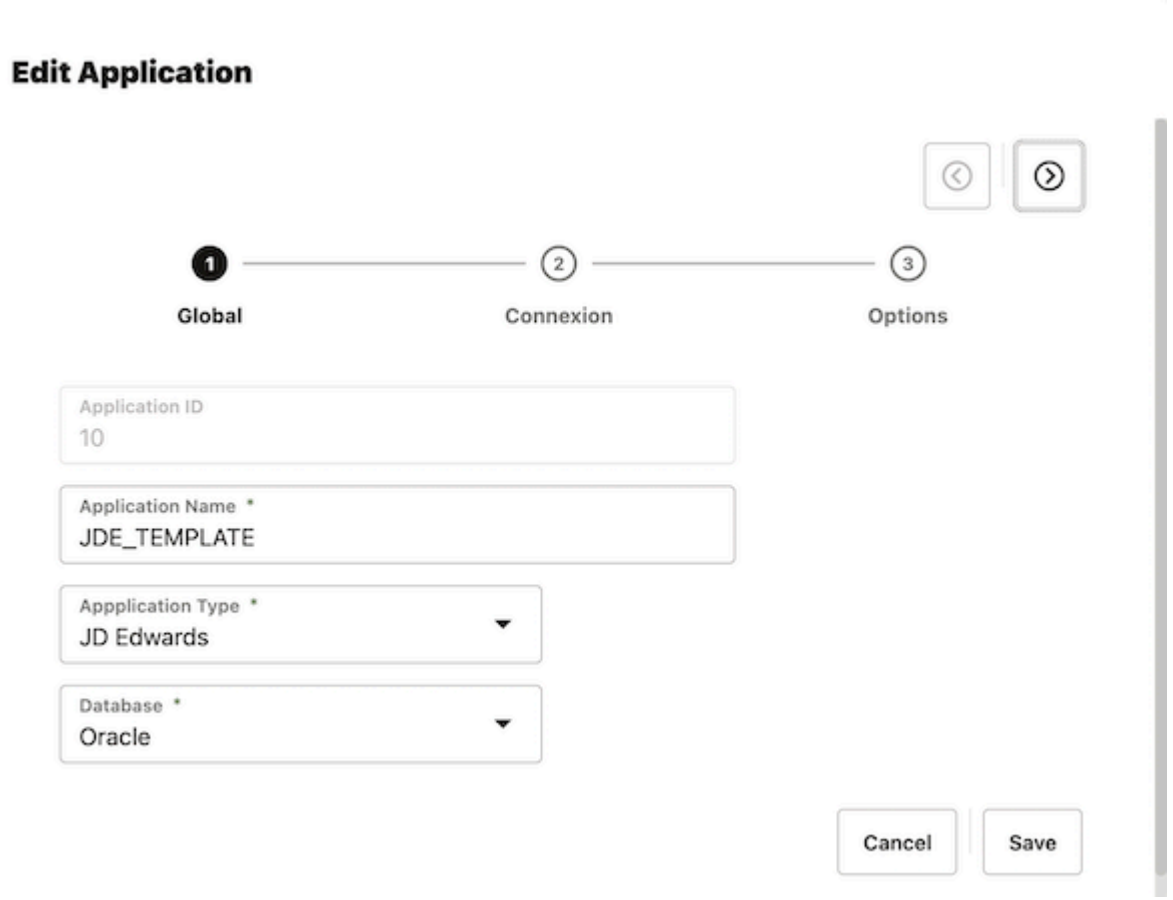
- Native connector for JD Edwards (Oracle, DB2 or MS-SQL)
- Native connector for Oracle Database
- Native connector for Microsoft Active Directory
- All databases accessibles with jdbc can be set



Application ID	Application Name	Appplication Type	Database	Host	User (Audit)	Date (Audit)
10	JDE_TEMPLATE	JDE	ORACLE	132.145.46.174	admin	2022-02-13
20	ORACLE_TEMPLATE	DATABASE	ORACLE	132.145.46.174	admin	2022-02-13

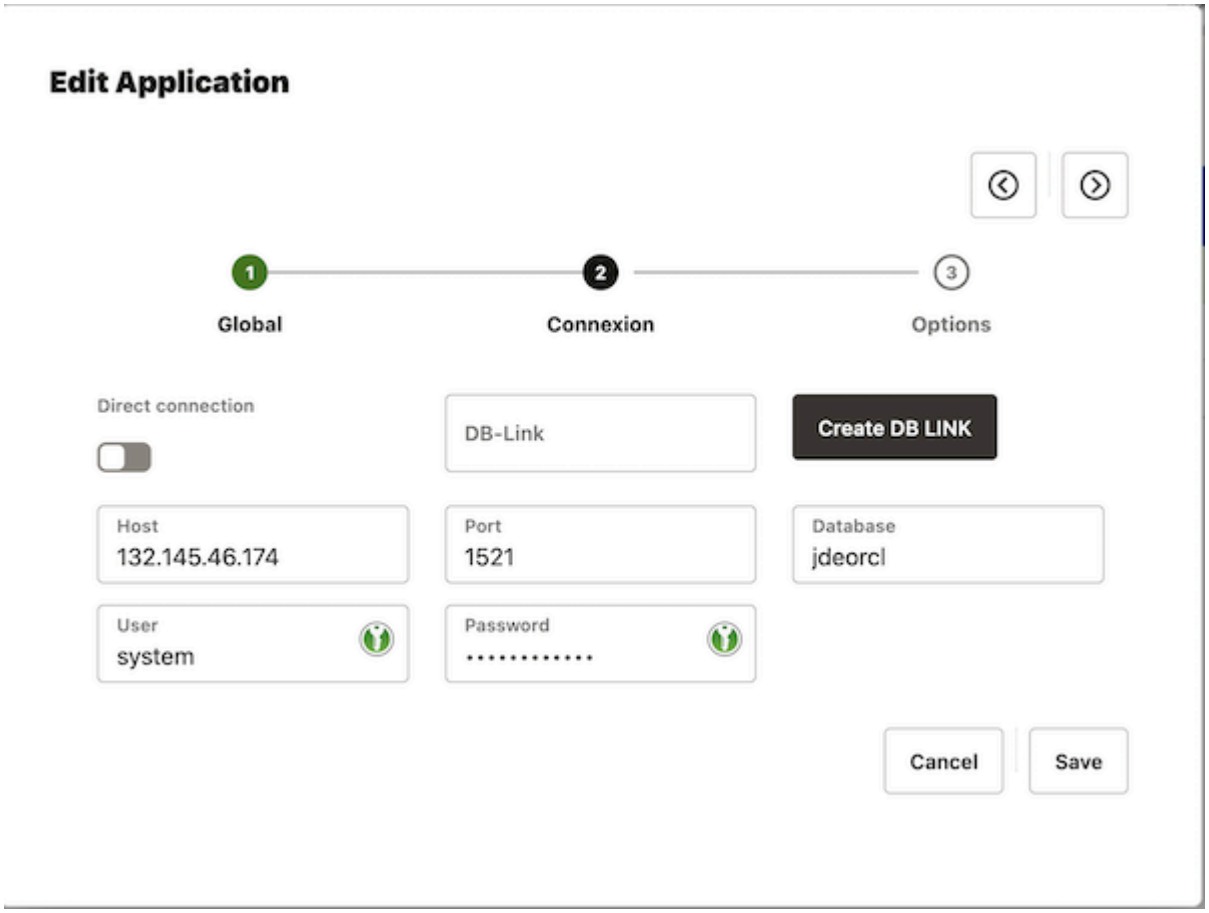
Click on add or edit to set a new datasource or modify an existing datasource and follow the wizard

1.1. Global Settings



Parameter	Description	Comments
Application ID	Unique ID	Automatic increment number used in all table joins
Application Name	Name of your application	
Application Type	Native or custom connector	JD Edwards, Database, LDAP, Weblogic, Custom Application
Database	Type of database	Oracle, MySQL, IBM DB2, Microsoft SQL Server, LDAP

1.2. Connections



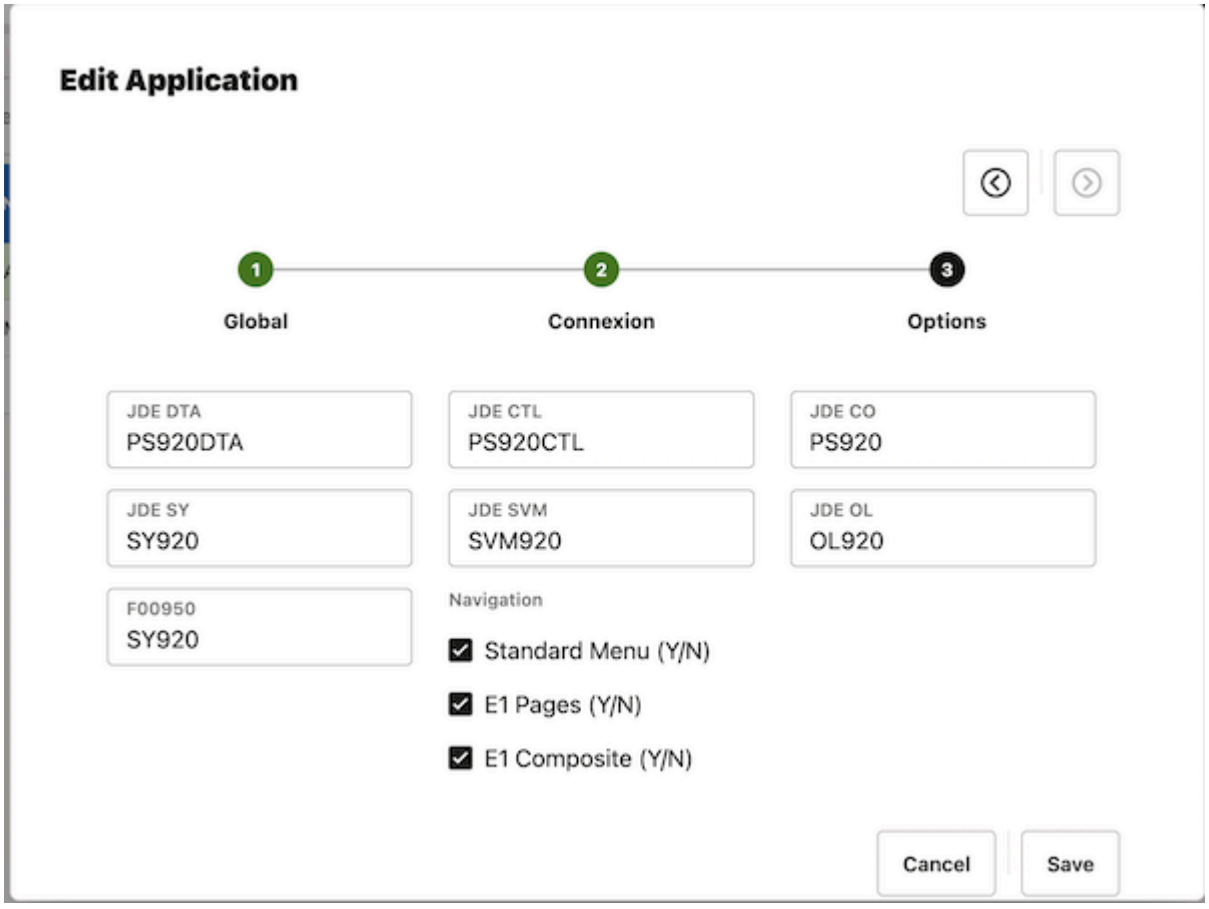
Some parameters could be hidden depending on the type of the application

Parameter	Description	Comments
Host	Database server	
Port	Database port	
Database	Service Name	Service Name and not SID for Oracle later than 12.2
User	Login to database	login could have read-only rights but with access to dictionary or catalog
Password	Password for the user	

1.3. Options

Parameters differs depending on the type of the application

1.3.1. JD Edwards



Parameter	Description	Comments
JDE DTA	Business Data	PRODDTA
JDE CTL	Control Tables	PRODCTL
JDE CO	Central Objects	PD920
JDE SY	System Tables	SY920
JDE SVM	Server Map	SVM920
JDE OL	Object Librarian	OL920
F00950	Security table location (sometimes not in SYSTEM)	SY920

Parameter	Description	Comments
Standard Menu (Y/N)	Collect Tasks Menus	
E1 Pages (Y/N)	Collect E1 Pages	Before Tools Release 9.2 and E1 composite
E1 Composite (Y/N)	Collect E1 Composite Pages	After Tools Release 9.2

1.3.2. Database / Custom Application

NONE

1.3.3. LDAP

Parameter	Description	Comments
LDAP Context	Search	OU=Utilisateurs,DC=nomana-it,DC=fr
LDAP Filter	Filtering type of object	(&(objectClass=user))
LDAP Exclude	Exclude node	OU=Applications,OU=Utilisateurs,DC=nomana-it,DC=fr

2. Users

3. Query

4. DWH