Overview

The application consists of an TypeScript-Angular Frontend, a Java-Spring Boot Backend and a Postgres DB.

The application is intended in a cluster-manner and therefore Docker is utilized for the Container Runtimes and Docker Compose as an Ochestration Tool.

Frontend

The Frontend is deployed as one container for the interaction with human users. It consists of 3 main functionalities (users, nodes, backups) to let users interact with the cluster manager interface.

Backend

The Backend is one whole application which consists of 4 parts, the shared part is code, functionality / endpoints that are present on all backend instances. Then there are 3 parts of the application that are each mostly unique and only run depending on as what role the container / backend should be working as decided by runtime configuration.

Cluster Manager

The Cluster Manager Role acts as a Leader Role in this cluster, representing the backbone backend to the Frontend (Frontend exclusivly communicates with CM-Role Backend) handling Metadata, Authentication and Managing / Orchestarting other backend instances aswell as configuration of all the other actual instances.

Backup Manager

The Backup Manager Role acts as the manager of backups, while all backup-related requests that are initiated through human users via the frontend go through the cluster manager (stores Backup Metadata) or tasks like backup-related cronjobs, are propageted to the Backup Manager which then propagetes and manages the communication and delegation to other Backup Node-Backend Instances that actually do the backup work.

Backup Node

The Backup Nodes are providing endpoints to accept delegated tasks to do the work of creating an actual backup and it's backupdata. This aspect of the application is of lesser importance, hence its rather a mocked, static manner,

while storing backup data is represented by storing some random data on different Backuo Node Postgres DB tables for each Backup Node (using the same Postgres Instance in the entire Docker Compose Setup).

Shared Functionality

Shared functionality as an example is the initiation of a registration process of nodes that try to register them self at the cm on backend startup so the CM knows which nodes / backends to use / communicate with for replication, etc.