LDAP SUBTREE MERGING PROXY DAEMON

AY 2023-2024

Eugenio Tampieri

Michele Ravaioli

REQUIREMENTS

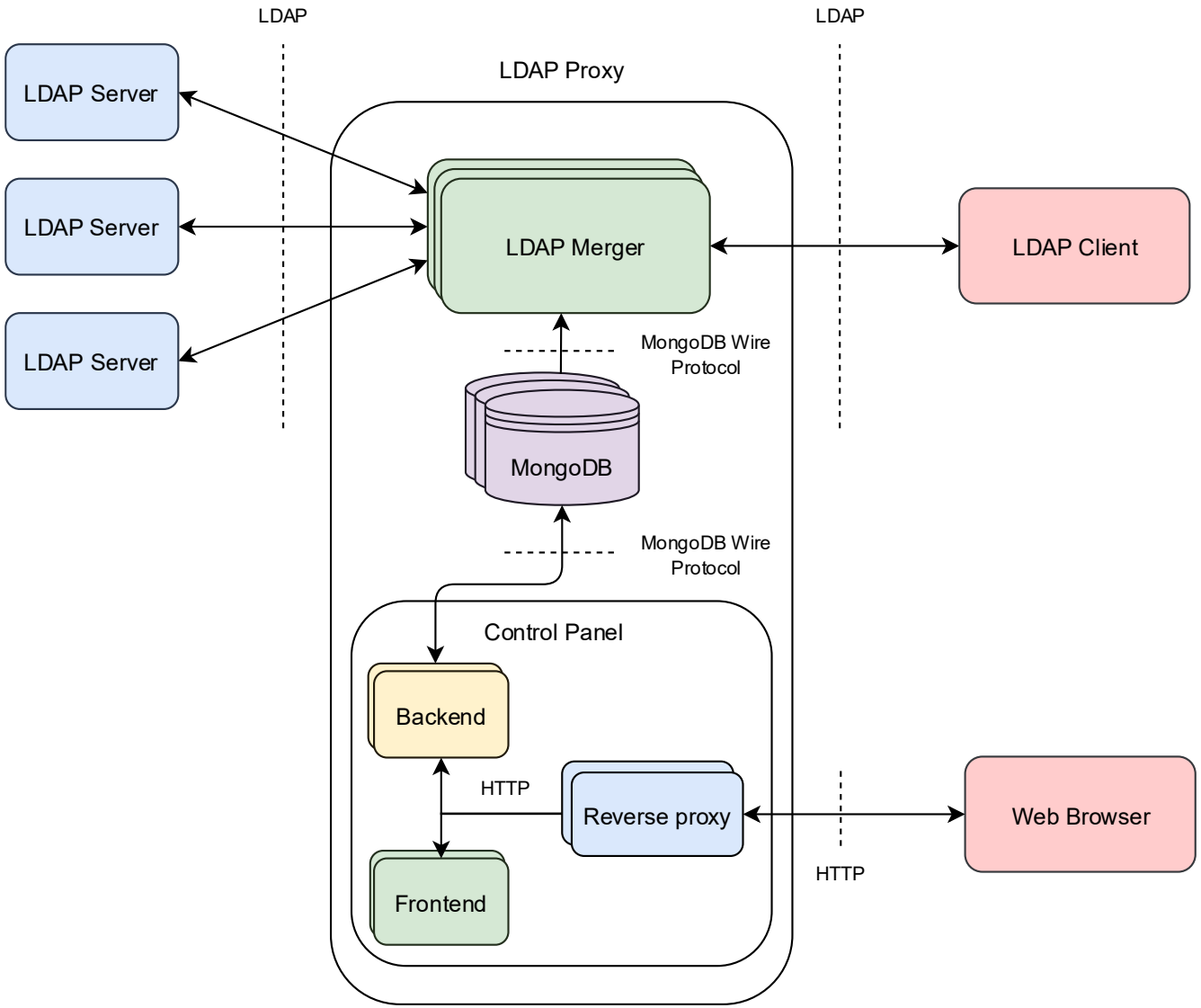
- Request Forwarding
- Multi-Target Handling
- Consistency

… but also …

- Runtime Configurability
- Containerization

WHY IS THIS A DISTRIBUTED SYSTEM?

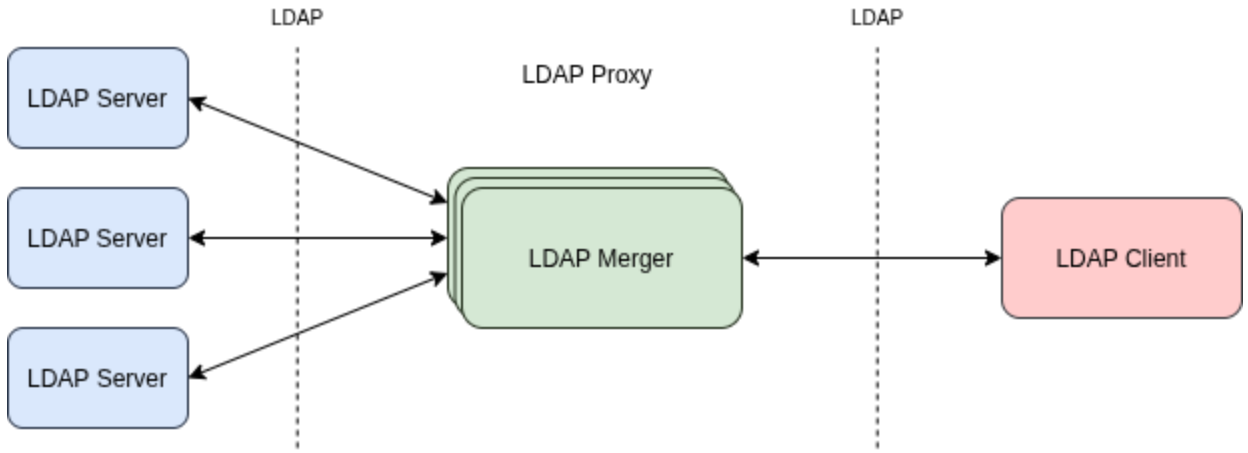
- Scalability
- Inherent nature of the problem at hand
  - Multiple servers
  - Potentially in different spatial-temporal contexts



ARCHITECTURAL OVERVIEW

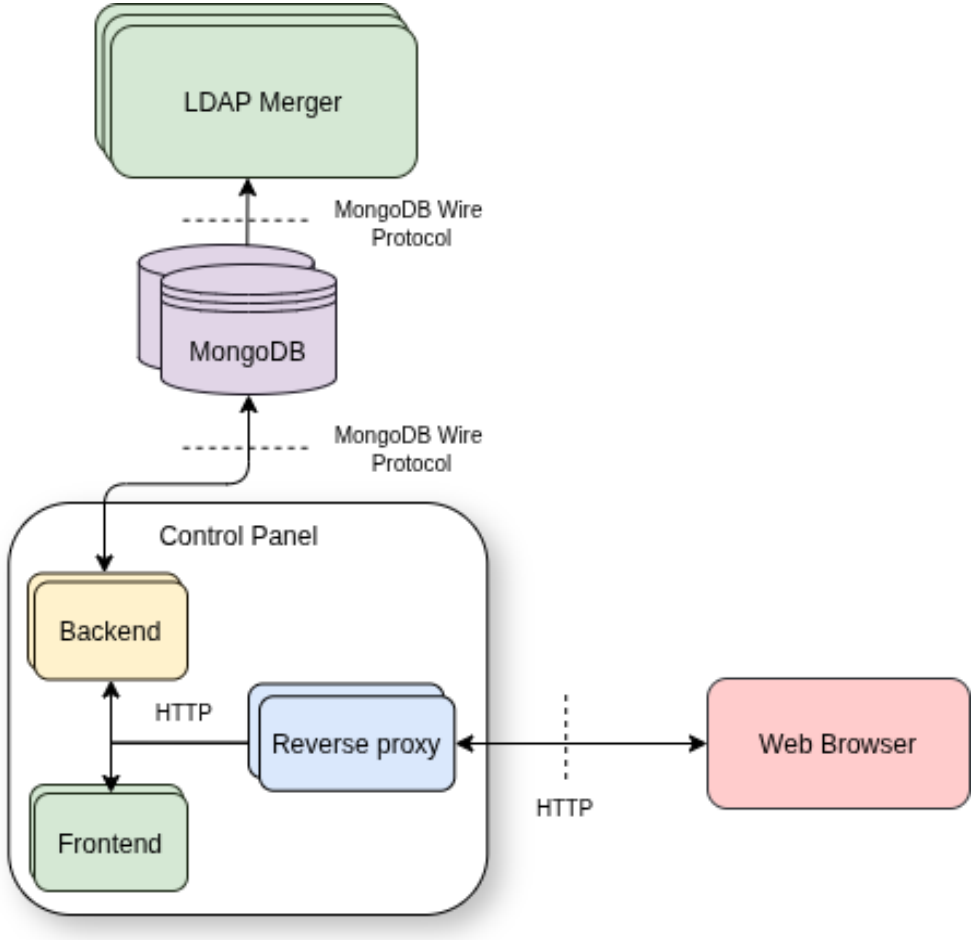
LDAP MERGER

- Read-only service
- Handles incoming LDAP requests
- Intercepts bind requests
  - On successful bind, authenticate to all backends
- Sends each query to every backend
- **Consistency over availability**
  - If a server goes down, the request fails
  - So that replies maintain consistency



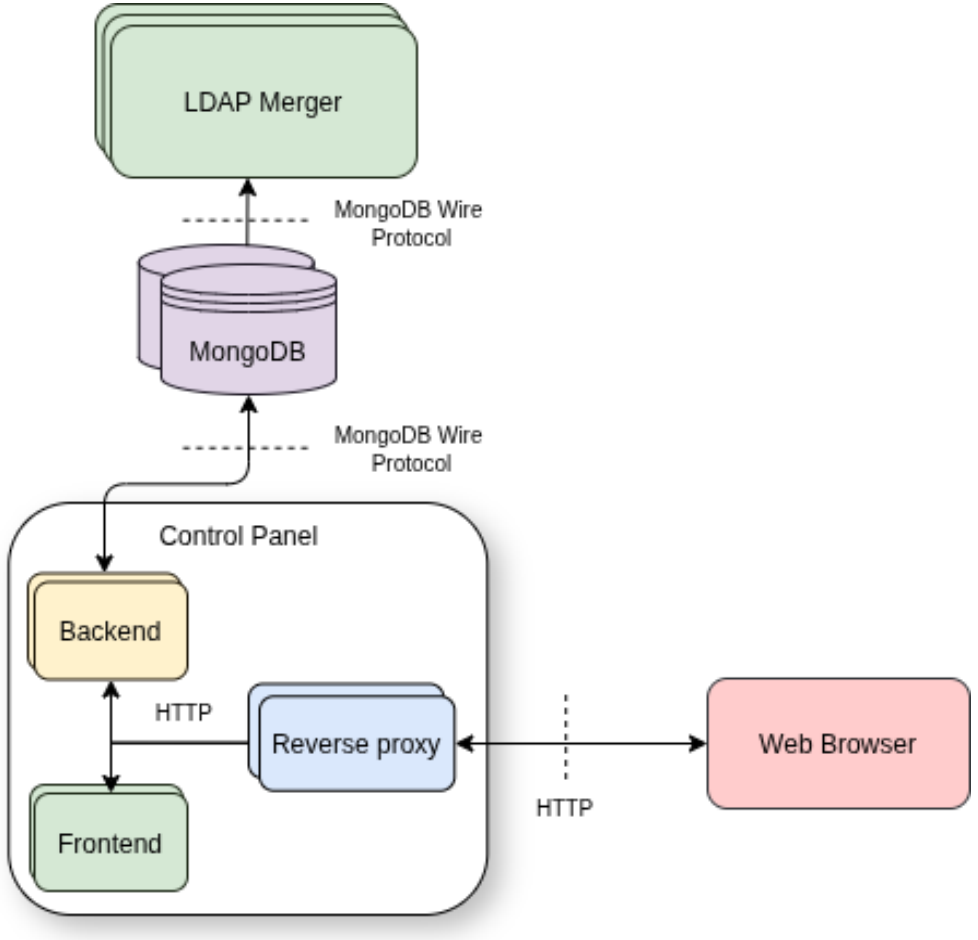
PERSISTENCE

- Based on MongoDB
  - Eventual consistency
  - Scalable and replicated
- Stores the configuration for the merger
  - The configuration is loaded on each incoming connection
  - If the database is down, no new connections can be handled

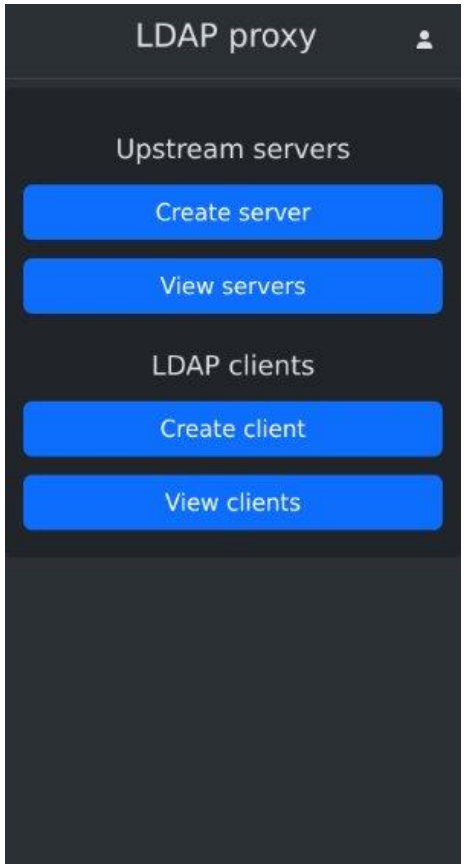


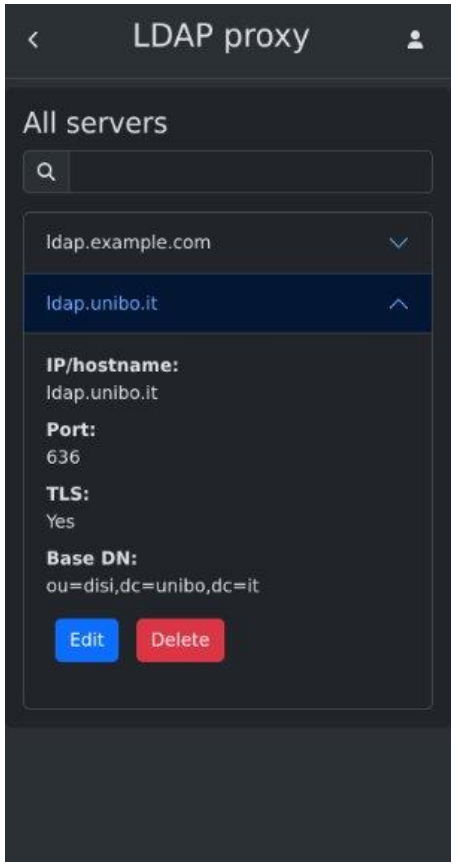
CONTROL PANEL

- Lightweight (and standalone) Web UI
- Manages
  - Backend servers
  - LDAP clients
  - Admin users
- Saves the configuration in the DB



CONTROL PANEL





DEPLOYMENT

- Each component is a separate container
  - It is independently scalable
- This project was deployed and tested on Kubernetes
  - Built-in load balancing
  - Built-in networking
  - Abstraction over physical location of each system function

kubernetes

QUESTIONS?