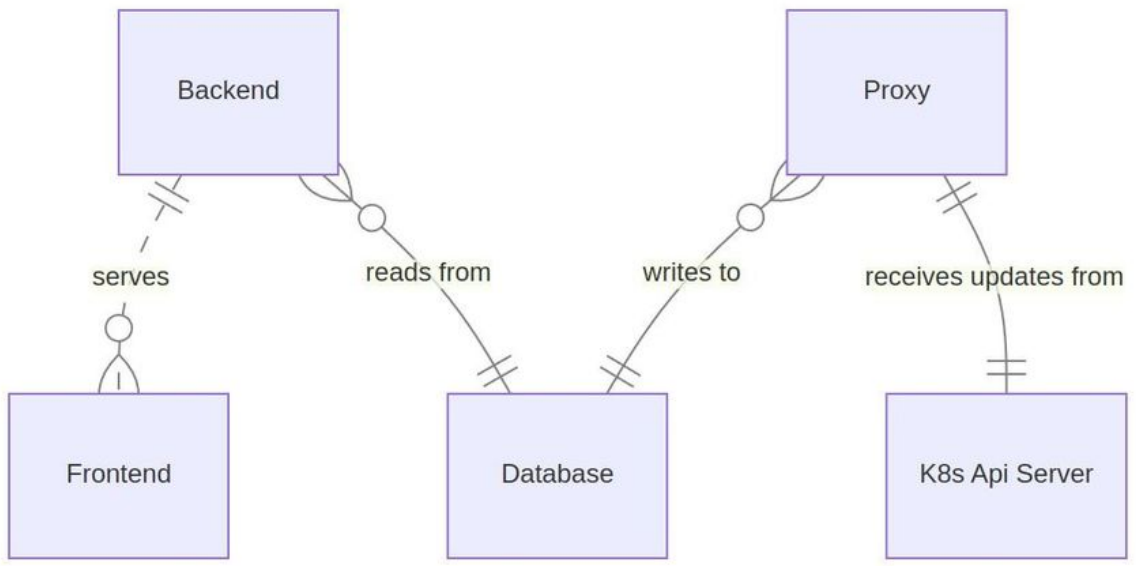
Technical Documentation

Architecture

Our Architecture is comprised of the following components:

1. **The Proxy:** Communicates with the Kubernetes API, listens for changes in the Kubernetes cluster and updates the database. It is written in `Golang`.
2. **The Database:** Stores all states and changes of Kubernetes components in the cluster. As a database we use `PostgreSQL`.
3. **The Explorer:** Frontend web application with backend that fetches data from the database and displays it in a user friendly manner in any local webbrowser. Here the user can navigate through the different parts of the cluster and look at health information, status information and event changelogs. It is written in `Next.js`and uses `Tailwind CSS` and the UI elements of `Flowbite`.

A visualization of this architecture can be found below.



Technology Stack summary

To increase reproducibility, we have commited to using `Docker` and `Docker-Compose` to set up our software. By building in the container, developers are not required to install any of the development tools (with the exception of a K8s API provider) locally.

Additionally, we are very mindful of polling vs event driven updates and are evaluating each part of our stack by its ability to receive, transform and emits events efficiently.

Folder Structure

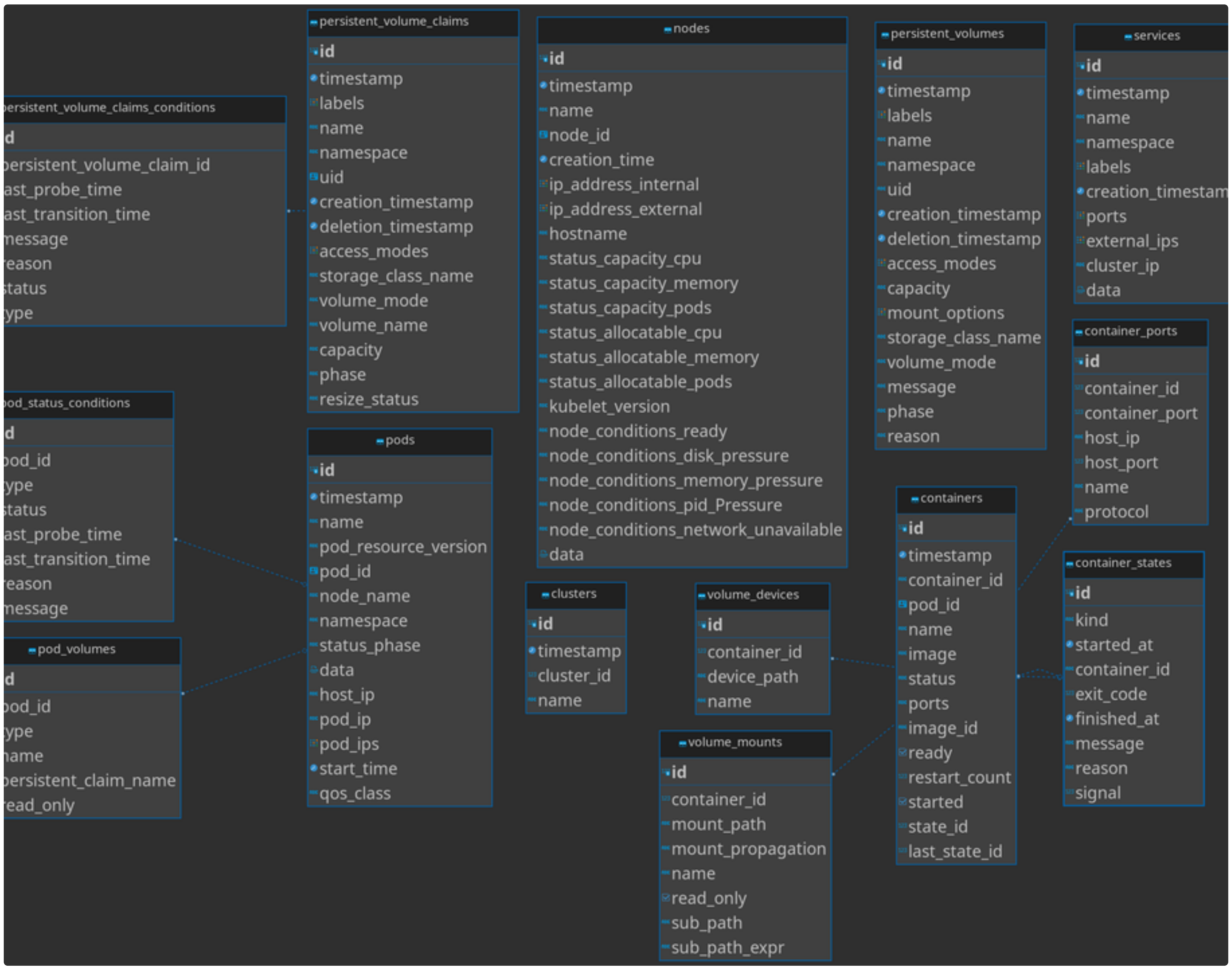
The three main folders of the projects

- DB
  - The database layout and the associated test data are stored there
  - Great Entrypoint
- Explorer - The Frontend
  - `package.json` - contains all relevant commands and dependencies for the development of the Explorer

    - `src/app` - The app router, each of the sub directories is equivalent to a route, so `src/app/containers/page.tsx` can be reached under `localhost:3000/container`
    - `cypress` - Our integration tests
    - `Dockerfile` - Shows how to build the application
- Proxy
    - `cmd/proxy/main.go` - The entry point to the application
    - `internal` - The actual implementation parts
    - `.kube` - The config folder where the Kubernetes credentials need to be for the docker-compose to work
    - `Dockerfile` - Shows how to build the application
- docker-compose.yaml
    - Allows for easy build and setup

Database Schema

See `DB/kubernetes-schema.sql` for the details


persistent_volume_claims
id
timestamp
labels
name
namespace
uid
creation_timestamp
deletion_timestamp
access_modes
storage_class_name
volume_mode
volume_name
capacity
phase
resize_status
nodes
id
timestamp
name
node_id
creation_time
ip_address_internal
ip_address_external
hostname
status_capacity_cpu
status_capacity_memory
status_capacity_pods
status_allocatable_cpu
status_allocatable_memory
status_allocatable_pods
kubelet_version
node_conditions_ready
node_conditions_disk_pressure
node_conditions_memory_pressure
node_conditions_pid_Pressure
node_conditions_network_unavailable
data
persistent_volumes
id
timestamp
labels
name
namespace
uid
creation_timestamp
deletion_timestamp
access_modes
capacity
mount_options
storage_class_name
volume_mode
message
phase
reason
services
id
timestamp
name
namespace
labels
creation_timestam
ports
external_ips
cluster_ip
data
ersistent_volume_claims_conditions
d
ersistent_volume_claim_id
ast_probe_time
ast_transition_time
message
eason
tatus
ype
od_status_conditions
d
od_id
ype
tatus
ast_probe_time
ast_transition_time
eason
message
pod_volumes
d
od_id
ype
ame
ersistent_claim_name
ead_only
pods
id
timestamp
name
pod_resource_version
pod_id
node_name
namespace
status_phase
data
host_ip
pod_ip
pod_ips
start_time
qos_class
clusters
id
timestamp
cluster_id
name
volume_devices
id
container_id
device_path
name
volume_mounts
id
container_id
mount_path
mount_propagation
name
read_only
sub_path
sub_path_expr
containers
id
timestamp
container_id
pod_id
name
image
status
ports
image_id
ready
restart_count
started
state_id
last_state_id
container_ports
id
container_id
container_port
host_ip
host_port
name
protocol
container_states
id
kind
started_at
container_id
exit_code
finished_at
message
reason
signal