Functional Specification

Jason Henderson: 19309916

Conor Joyce: 19425804

Date: 17/11/2022

0. Table of contents

1. Introduction

1.1 Overview

Continens is a system that provides an easy-to-use web interface for deploying and managing Docker containers. Users will have the ability to create new containers, manage existing containers, connect to them in-browser and expose a container publicly that can be accessed externally (e.g. a web server or database container).

The barrier to entry for using Docker and Docker containers is somewhat high, and is a hard concept to grasp for new developers. Continens aims to fix this by providing a simple web UI that can perform all the functions necessary to deploy and manage Docker containers.

Continens will have a backend that interacts with the Docker API on the host system to perform tasks such as container creation and management, while the frontend for Continens will simply send HTTP requests to the Continens backend. This design model consists of multiple layers of abstraction, with the goal to make tasks as easy as possible for the end user.

Continens will be primarily written in Go and JavaScript, and will also involve a lot of infrastructure setup and design work. The backend will be a Go REST API and the frontend will be a web application written in SvelteKit, which makes writing a web application with all the modern best practices in mind much easier.

The name Continens is a rough translation of the word “container” to Latin. We chose this as it highlights the main aspect of the project and also sounds quite cool!

1.2 Business Context

n/a

1.3 Glossary

Docker - an open source software platform to create, deploy and manage virtualized application containers on a common operating system.

Docker Containers - A Docker container is a virtualized runtime environment used in application development. It is used to create, run and deploy applications that are isolated from the underlying hardware.

Docker Images - A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker.

Database - A database is an organised collection of structured information, or data, typically stored electronically in a computer system.

Front-end - Denoting the part of a computer system or application with which the user interacts directly.

Back-end - The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

REST API - A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

SvelteKit - SvelteKit is a framework for building web applications of all sizes, with a beautiful development experience and flexible filesystem-based routing.

WeTTY - Terminal access in browser over http/https.

Traefik - Traefik is a leading modern reverse proxy and load balancer that makes deploying microservices easy.

Reverse Proxy - A reverse proxy is the application that sits in front of back-end applications and forwards client requests to those applications.

Nomad -

Kubernetes - Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

Unit Testing - Unit testing is a software testing method by which individual units of source code.

2. General Description

2.1 Product / System Functions

- **Creating a container**
  A user can create a container via the web UI. This container will either use an uploaded Dockerfile, docker-compose file, dockerhub link or a GitHub link
- **Connecting to a container**
  A user can select a given container in the web UI and connect to it via SSH within their browser. This process will open a WeTTY instance in a new tab that connects to the selected container
- **Managing a container**
  A user can change settings relating to their container via the web UI, such as the image, volumes, environment variables, etc.
- **Deleting a container**
  A user can delete any one of their containers via a button in the web UI.
- **Exposing a container publicly**
  A user can edit a setting in the web UI to expose one of their containers publicly at their personalised url, e.g. user.continens.xyz

2.2 User Characteristics and Objectives

Continens is aimed at both experienced and inexperienced users of Docker. In providing users with functionalities with using Docker we aim for this to be practical in that, a new user who has no experience can learn how Docker works by using Continens and getting familiar with commands. Whereas a frequent experienced Docker user could make use of Continens to spin up containers quickly through ease of use all displayed on a web UI rather than having to interact with Docker manually themselves. The web-app will be designed to be as user friendly as possible.

2.3 Operational Scenarios

- **User logs on to Continens to create a container**
  The user can create a container on continens through the web UI.
- **User wants to host their website on Continens**
  The User can start an nginx container (or any other web server), ssh into that container and pull in their website's source code. Then they can set that container to be exposed publicly on port 80 and they will be able to access their site at username.continens.xyz
- **User wants to delete their web server container from Continens**
  User can select the given container in the web UI, go to the settings for that container and press the delete button.
- **User wants to spin up a database and expose it publicly**
  User can start a MySQL container (for example) and in their Continens settings, set the container to be exposed publicly. It will then be accessible from username.continens.xyz:3306 (where 3306 is the default port for MySQL).
- **User wants to start a container that is built from a Dockerfile stored on GitHub**
  User can select the option to input a Dockerfile, where they can paste the link to their git repository, e.g. https://github.com/username/repo. The repository must contain a Dockerfile, else Continens will display an error message. Continens will then build a docker image with the given Dockerfile before starting a container based on the generated image.

2.4 Constraints

- **Speed**
  Speed can be viewed as a constraint because we can't guarantee each users requests will consist of working with either small or large amounts of data, therefore we will need to implement a responsive intuitive working system.
- **Networking**
  Networking will be a large aspect of the project that will likely be hard to configure and set up. This involves dealing with requests going back and forth from the frontend to the backend while using the site, and also allowing users to publicly expose their containers.
- **Hardware**
  Continens requires a large amount of infrastructure to function correctly and depends on hardware running 24/7 to host the API and run user's containers. This may demand a lot of resources.
- **Scalability**
  We plan on deploying Continens across a cluster of hosts to help alleviate concerns about resources as mentioned above. This will require more complex infrastructure also but will give us a lot more flexibility and availability in the end.

3. Functional Requirements

Container Management

Description	In order for Continens to function we need to incorporate container management for the users, for example creating/deleting a container.
Criticality	This is a critical requirement for Continens as it this functionality is what the project is built upon and what we offer our users.
Technical issues	Technical issues could consist of issues interacting with the Docker API, programmatic errors or user errors in a similar manor
Dependencies with other requirements	n/a

User Authentication

Description	Continens must have user authentication and isolation between each user's data
Criticality	This is very critical to the project, as each user must be separated from each other for privacy and security reasons.
Technical issues	Will require implementation on both the backend and the frontend to function correctly.
Dependencies with other requirements	n/a

Networking

Description	Continens must have the ability to route requests appropriately based on user or container. Also the ability to expose a given container on a per user basis.
Criticality	This is very critical because a non-working networking stack will result in a messy and unintuitive application.
Technical issues	Lots of configuring and testing involved here
Dependencies with other requirements	n/a

Reliability

Description	Reliability is key to Continens as it shares the same offer to all its users in having a clean easy to use system with little to no issues.
Criticality	This is critical because we want every user's experience to be the same when using Continens.
Technical issues	The backend API needs to be stable and well designed to ensure the above.
Dependencies with other requirements	• Containerx Management • Networking

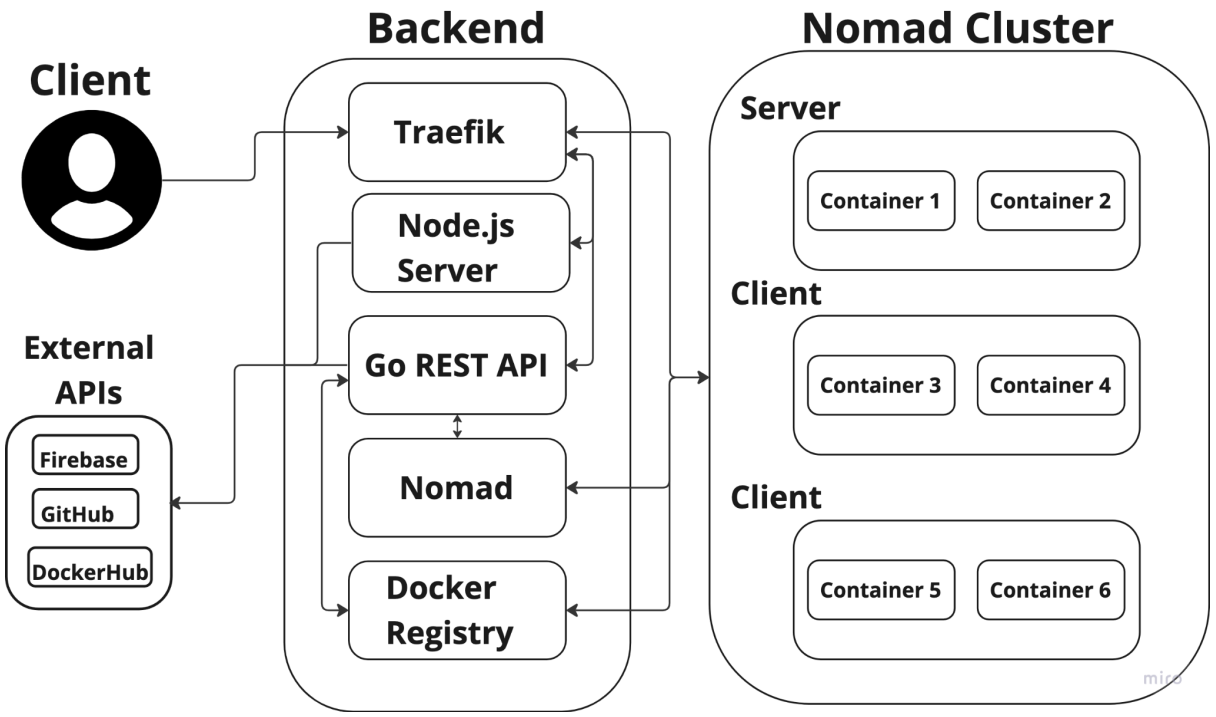
Good UX

Description	Continens should have a good, intuitive UX (User Experience) in the web UI.
Criticality	Since our goal is to create an easy to use interface for docker containers, a good UX is very important.
Technical issues	n/a
Dependencies with other requirements	• Container Management • User Authentication • Networking

Docker Images

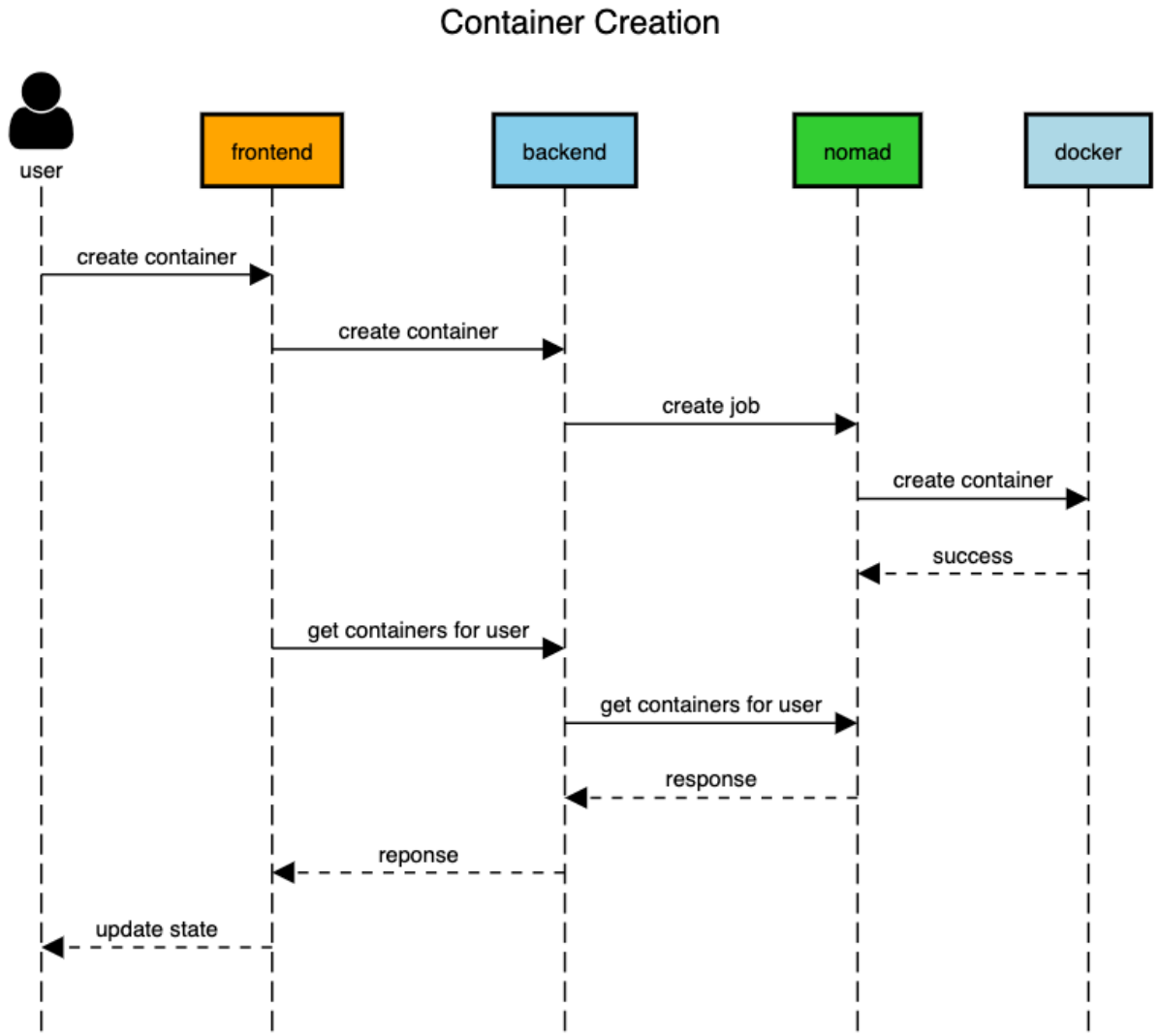
Description	Continens should have the ability for users to upload their own Dockerfiles, pull an image from Dockerhub or supply a link to a github repository that contains a Dockerfile.
Criticality	Pulling images from dockerhub is essential but should “just work” by default. Enabling use of Dockerfiles is a great feature but not as criticial
Technical issues	Using arbitrary Dockerfiles requires us to spin up our own local docker registry to store the images on, as nomad cannot just take a Dockerfile and distribute it to each node. We will need to build the image in our backend and push the image to our local docker registry, from which each nomad node can pull the image and run it.
Dependencies with other requirements	• Container Management • Networking

4. System Architecture

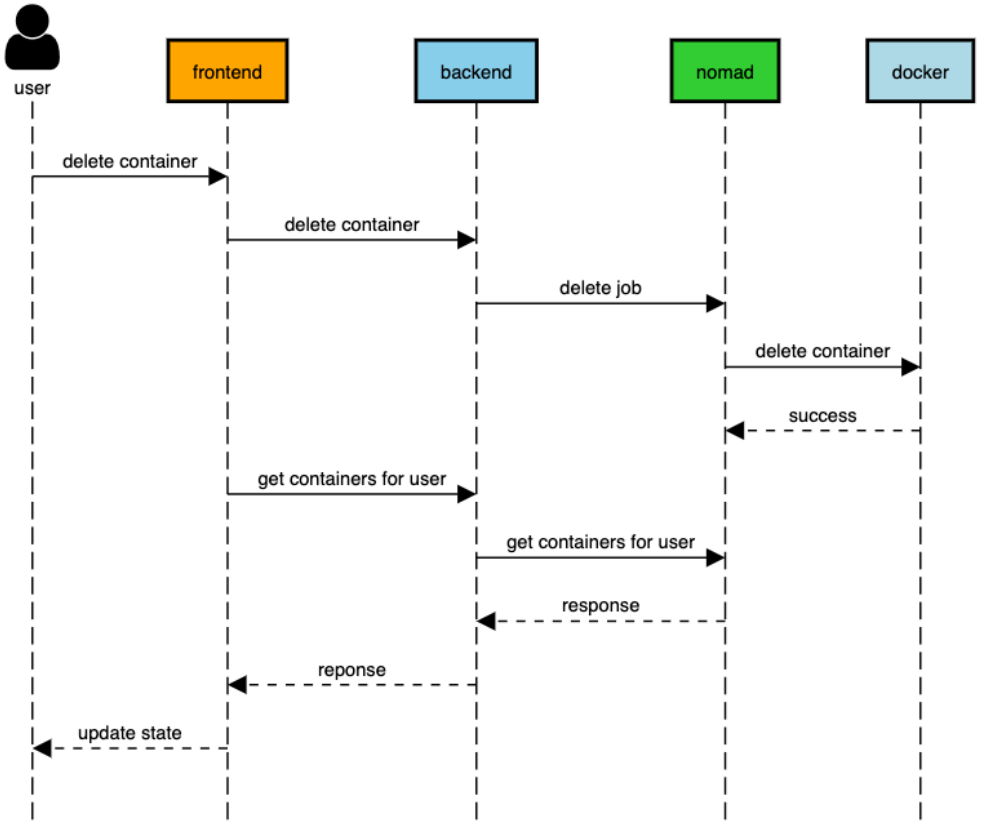


5. High-Level Design

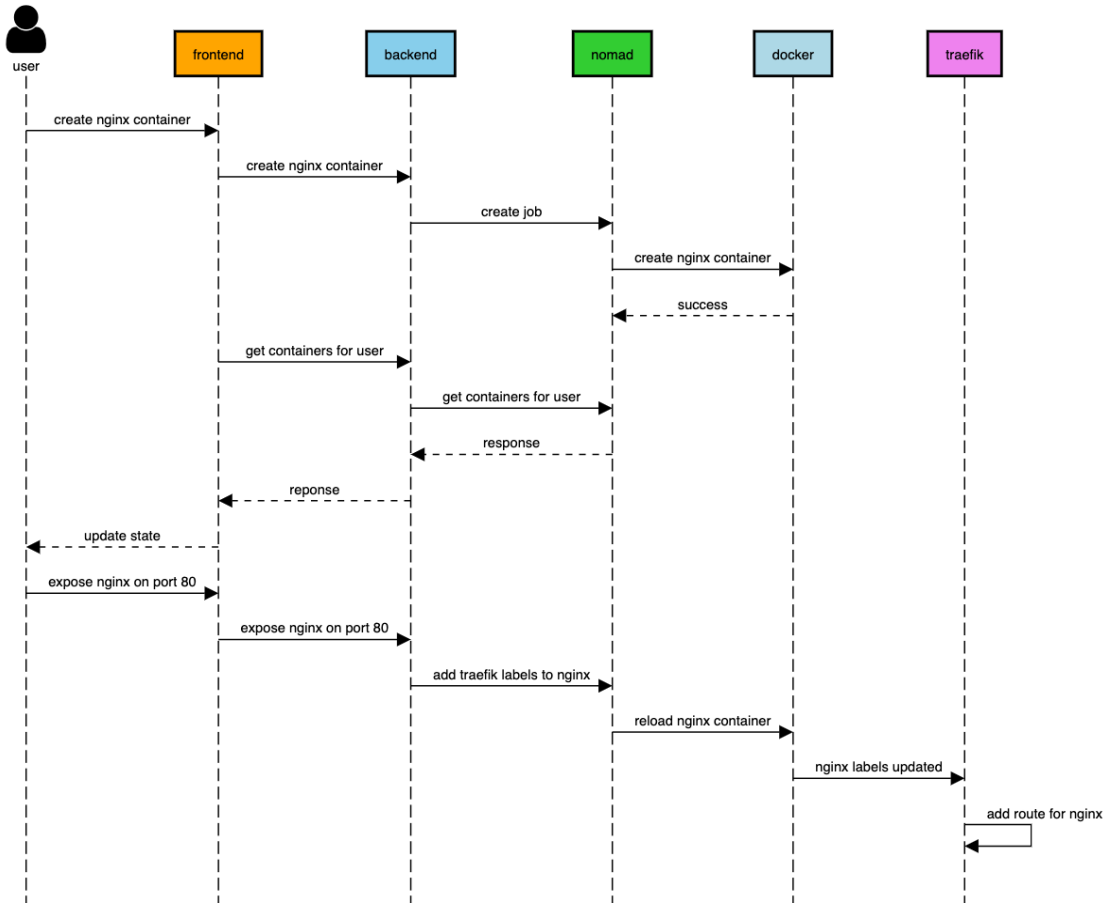
5.1 Sequence Diagrams



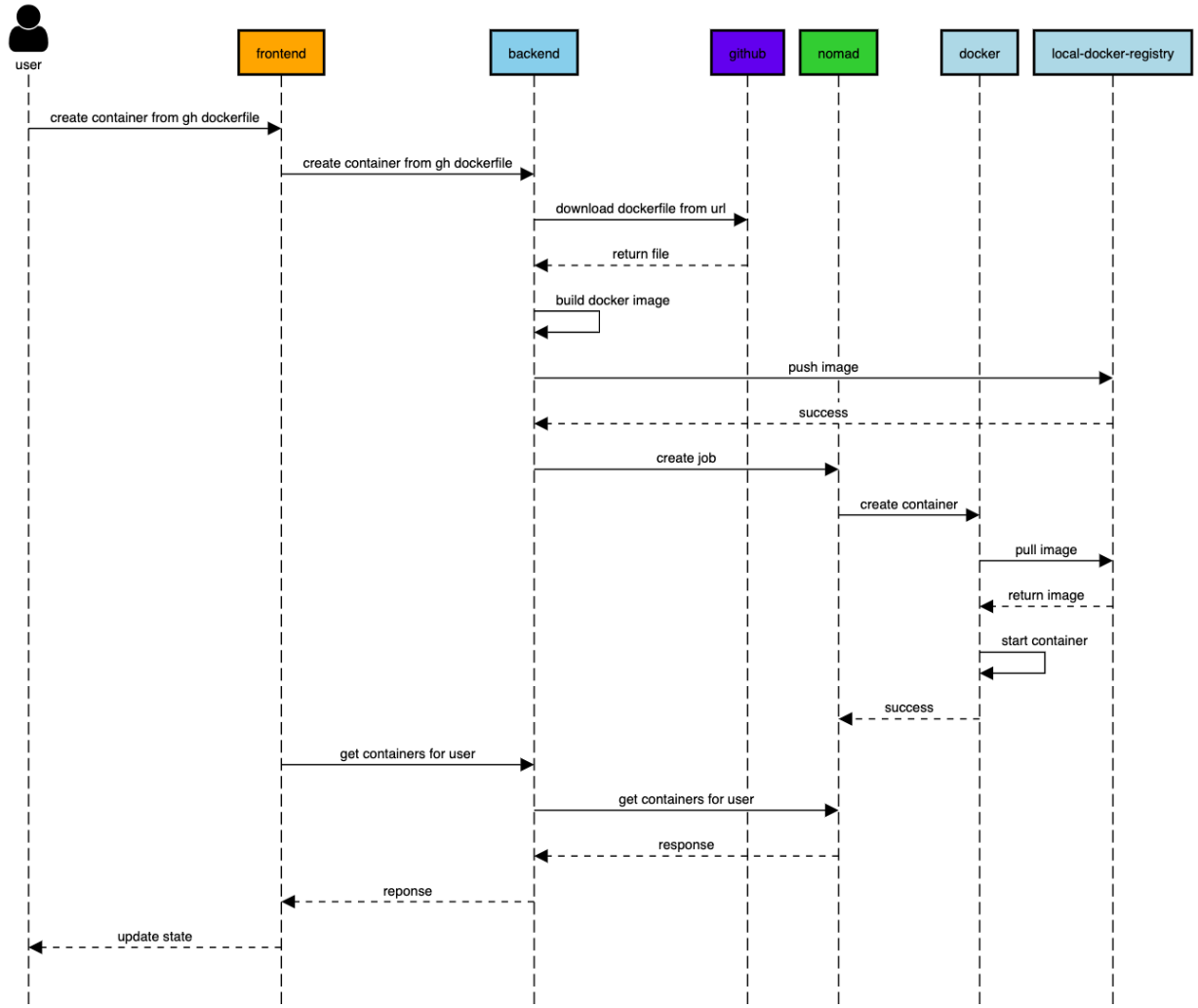
Container Deletion



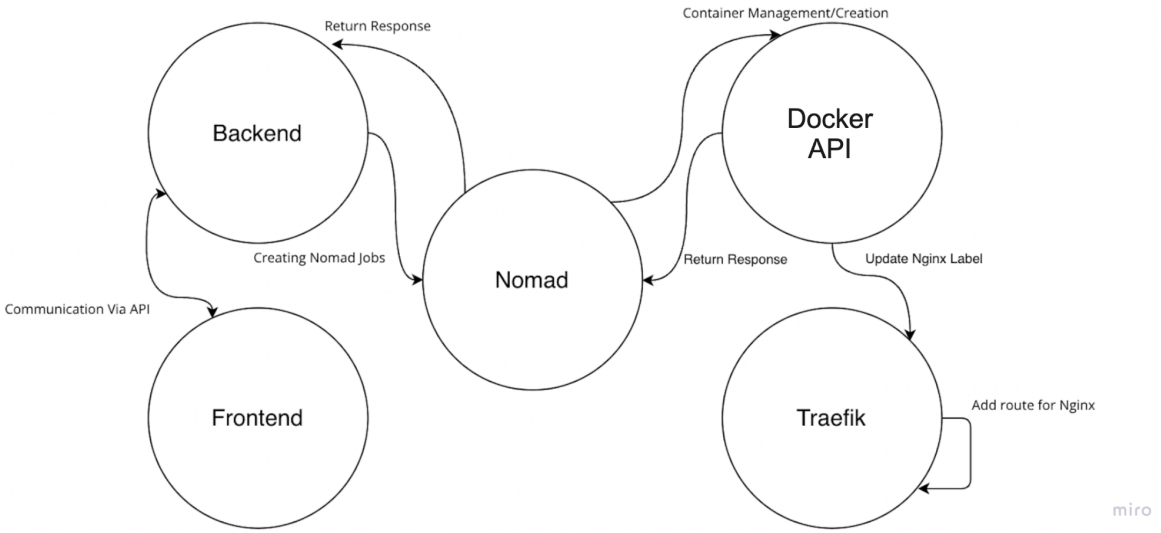
Create and Expose a Web Server



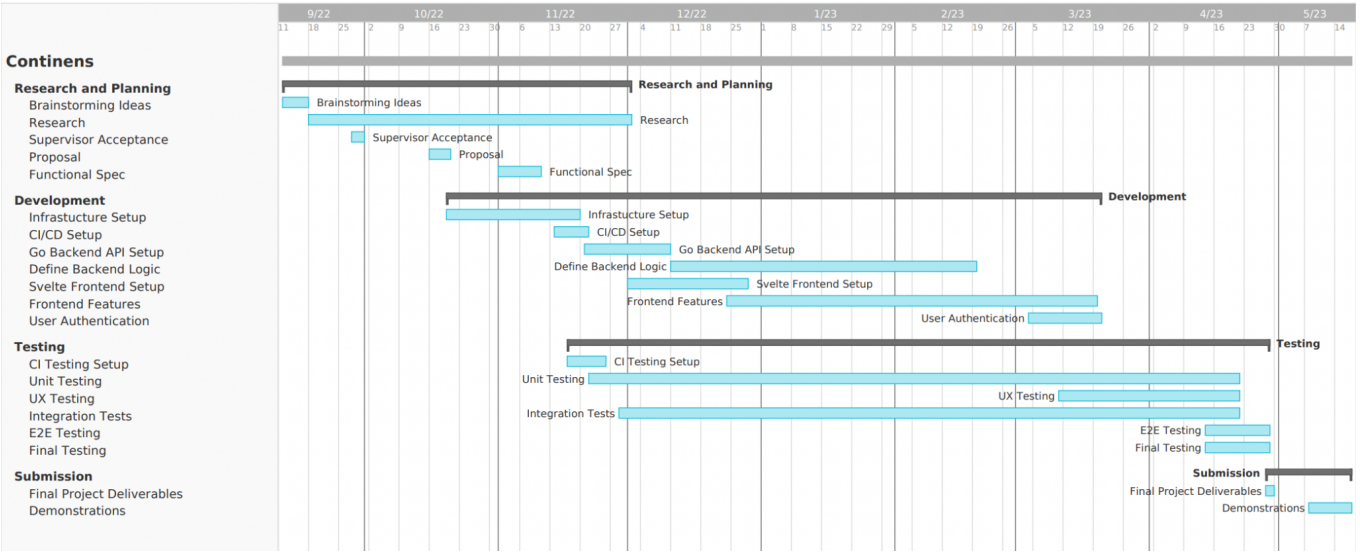
Create Container From GitHub Dockerfile



5.2 Data Flow Diagram



6. Preliminary Schedule



7. Appendices

Below specifies other useful information for understanding the requirements.

- Nomad API
- Deploying a Docker Registry
- Traefik Docker Configuration
- Traefik Nomad Configuration
- SvelteKit