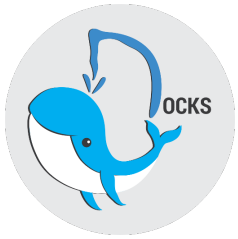

Requirements and Design for Docks

Team: TripleParity
Client: Compiax

Team Members
Francois Mentz
Connor Armand du Plooy
Raymond De Vos
Evert Geldenhuys
Anna-Mari Helberg
Paul Wood

Contents

1 System Overview

1.1 Purpose

Docker is a tool designed to make it easier to create, deploy and run applications using lightweight virtualization. It provides a command line interface (CLI) and RESTful API. Maintaining and deploying applications often involve multiple people. Providing multiple people access to the **Docker** CLI requires Secure Shell access (SSH) as root to the server running **Docker** . If the server is secure it will only provide SSH access using public/private keys, which reduces convenience and restricts access to devices that are SSH capable and holds a private key. The **Docker** API lacks functions which are provided by the CLI, so it cannot be used on its own.

The purpose of **Docks** is to provide a secure web user interface for using **Docker** .

1.2 Product Scope

Docks will provide functionality in three areas

1. Provide a secure* web user interface that enables using the essential* functions exposed by the **Docker** API and CLI.
2. Provide a secure API to allow third party services to integrate with **Docks** and use **Docker**
3. Send real time notifications to system administrators via Slack about important events

1.3 Definitions, acronyms and abbreviations

Docks	A system to provide a web user interface and API for using **Docker**
Docker	Tool designed to make it easier to create, deploy and run applications using lightweight virtualization
Image	"A container is launched by running an image. An image is an executable package that includes everything needed to run an application–the code, a runtime, libraries, environment variables, and configuration files."
Container	"A container is a runtime instance of an image–what the image becomes in memory when executed (that is, an image with state, or a user process)"

1.4 UML Domain Model

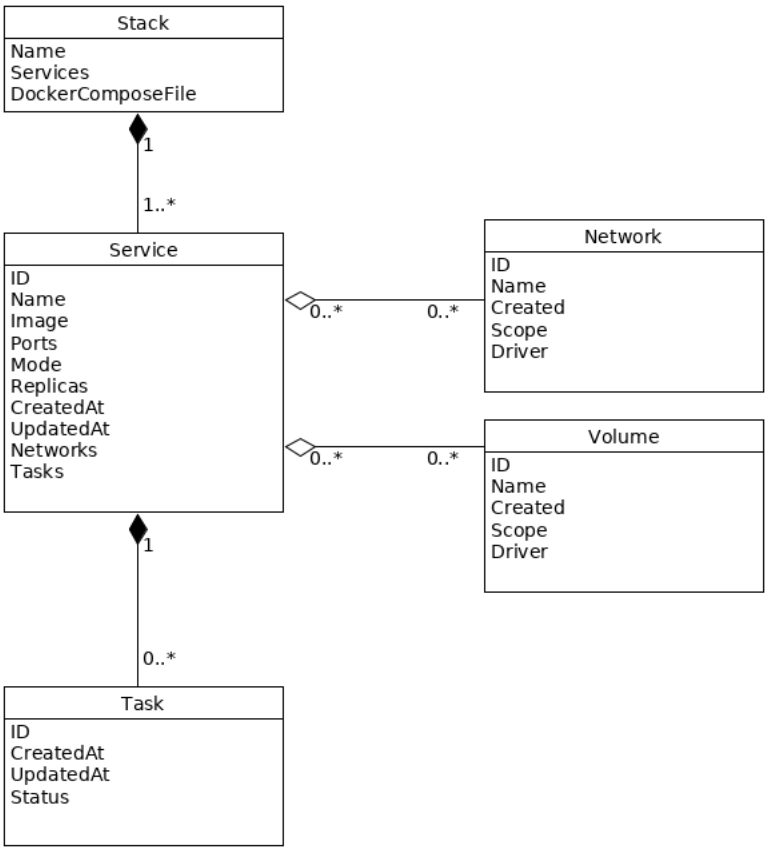

Figure 1: UML Domain Diagram for the Docks System

2 Functional Requirements

2.1 Users

Docks will appeal mainly to software developers and system administrators with fundamental understanding of Docker . Docks will allow them to deploy new applications with Docker as well as manage existing applications. With the web user interface they will be able to update and troubleshoot applications using any web browser. They will also be able to give access to the web user interface to other administrators for assisting in management. It is assumed this category of users will have knowledge on configuring applications to be deployed with Docker and the ability to troubleshoot networks and applications.

Docks will also appeal to users that are interested in learning how to use Docker . With Docks they can deploy pre-configured applications and develop an understanding of the features provided by Docker using the web user interface. It is assumed these users know the basic Docker terminology and can learn from the Docker documentation.

2.2 Subsystems

Docks consists of two projects with distinct purposes: Docks UI which is the web user interface running in the web browser and Docks API which is the server running on a Manager Node*.

Docks UI is responsible for displaying information about applications running in Docker , and to provide a convenient interface for sending information to Docker via the Docks API.

Docks API is responsible for providing authenticated access to the **Docker** API. It also extends the **Docker** API by providing the ability to deploy Stacks* and monitor **Docker** events for sending notifications.

Across these two projects exist a number of subsystems

2.2.1 Authentication

The authentication subsystem is responsible for authenticating and authorizing users as well as managing (create, edit, delete) user accounts.

2.2.2 Webhooks and Docker Events

This subsystem is responsible for managing (create, edit, delete) webhooks. It interfaces with the Docker API to listen for events and send relevant date to the stored webhooks. The

The Docks system consists of two main subsystems. The Docks API Server and the Docks UI.

- All management and view operations require authentication and the correct authorization
- Fine-grain permission control for user accounts
- Account roles with different permissions
- View Docker Nodes that are part of the Swarm
- View Services that are running in the Swarm
- View Tasks running in the Swarm
- Start and stop services
- Remove services
- Deploy a Stack to the Swarm using a docker-compose file
- Deploy a Service to the Swarm
- View Networks in the Swarm
- View Volumes in the Swarm
- Deploy images from a private repository

2.2.3 Docks API Server

The Docks API Server acts as a middleman between the UI and the Docker API running on the Manager Node. The Docks API also adds a layer of authentication and session management over the Docker API.

2.2.4 Docks UI

Docks UI is the web interface for managing the Docker Swarm through the Docks API. It consumes the API provided by the Docks API server.

2.3 Specific Requirements

- R1.1 The system shall allow an authorized user to interact with the Docks API
- R1.2 The system shall allow a user to perform actions only when they are authorized to do so.
- R1.3 The system shall provide a global administrative account role without restrictions
- R1.4 The system shall provide Teams
- R1.5 The system shall provide the ability to add Users to Teams

- R1.6 The system shall only allow a User to manage resources that are part of their team.
- R1.7 The system shall provide the following account roles within Teams: (Role (Permissions))
  - R1.7.1 Team Leader
  - R1.7.2 Super User
  - R1.7.3 Normal User
  - R1.7.4 Guest
- R1.8 Roles shall have the following permissions:
  - R1.8.1 Team Leader
    - Add users to team
    - Bind Mount
    - Deploy Resources
  - R1.8.2 Super User
    - Bind Mount
    - Deploy Resources
  - R1.8.3 Normal User
    - Deploy Resources
  - R1.8.4 Guest
    - Read only access. No access to Inspect
- R1.9 The system shall provide the following user management features to be used by administrative accounts
  - R1.9.1 Create new user
  - R1.9.2 Remove user
  - R1.9.3 Update user password
  - R1.9.4 Set permissions as stated in R1.3

2.3.1 Docker Swarm Management Functions

- R2.1 The system shall display all nodes in the swarm.
- R2.2 The system shall display all services running in the swarm.
- R2.3 The system shall display the following attributes about a Task:
  - R2.3.1 Container Name (if set)
  - R2.3.2 Uptime
  - R2.3.3 Container ID
  - R2.3.4 State of the container (running/stopped)
  - R2.3.5 Image used to create the task
- R2.4 The system shall allow a user to stop a running service
- R2.5 The system shall allow a user to start a stopped service
- R2.6 The system shall allow a user to upload a docker-compose file to deploy a Stack
- R2.7 The system shall allow a user to remove a stopped service
- R2.8 The system shall allow a user to destroy a Stack

- R2.9 The system shall display the networks on a Node
- R2.10 The system shall display the volumes on a Node
- R2.11 The system shall be able to deploy images from a private repository
- R2.12 The system shall display the following attributes about a Service
  - R2.12.1 Service Name
  - R2.12.2 Stack (if deployed from Stack)
  - R2.12.3 Tasks running in the service
  - R2.12.4 State of the Service
  - R2.12.5 Image used to create the Service

3 Architecture

3.1 Interfaces

3.1.1 Software Interfaces

Since the frontend cannot securely interface with the Docker API, an intermediate interface will be developed (Docks API). The Docks API will communicate between the frontend (Docks-UI) and the Docker API. The Docks API will provide a simplified interface for interacting with the Docker API.

3.2 System Configuration

The Deployment diagram shows the architecture from the device perspective.

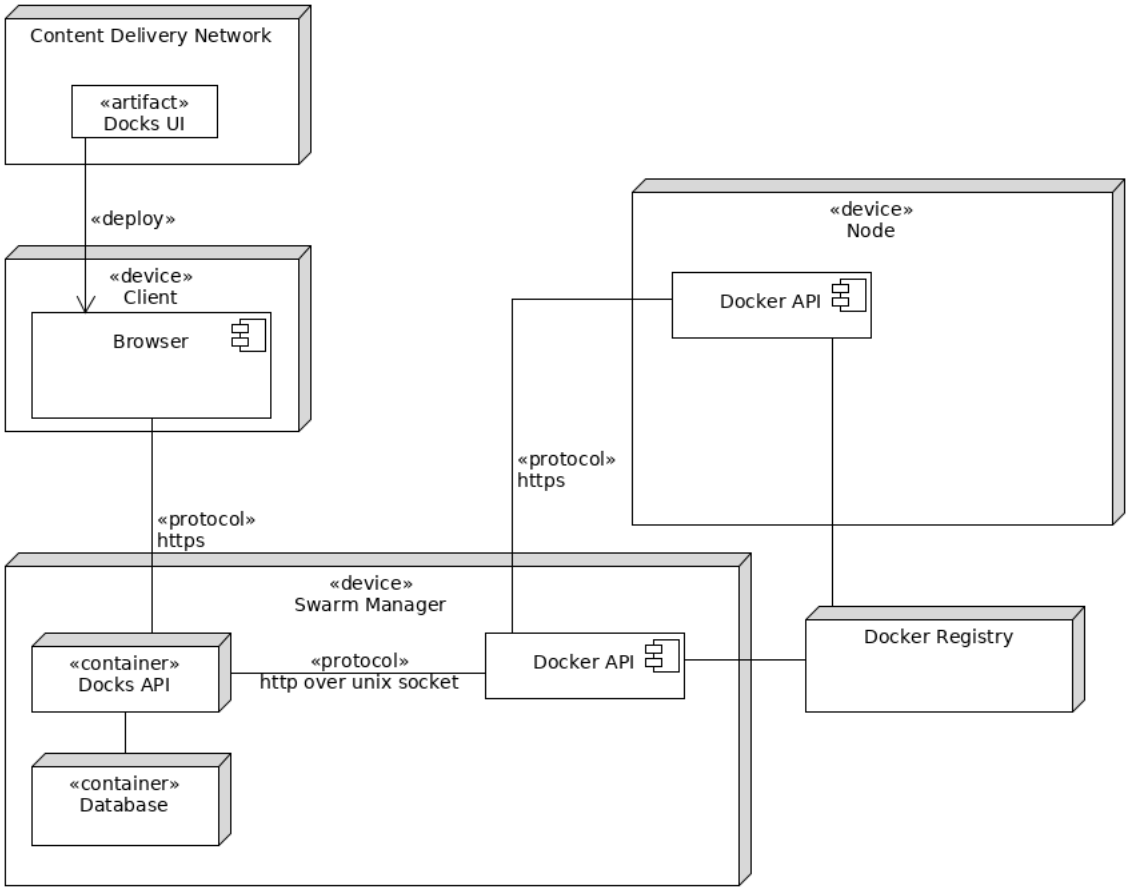

Figure 2: UML Deployment Diagram for the Docks System

3.3 ERD Diagram

The Deployment diagram shows the database architecture for the Docks API system

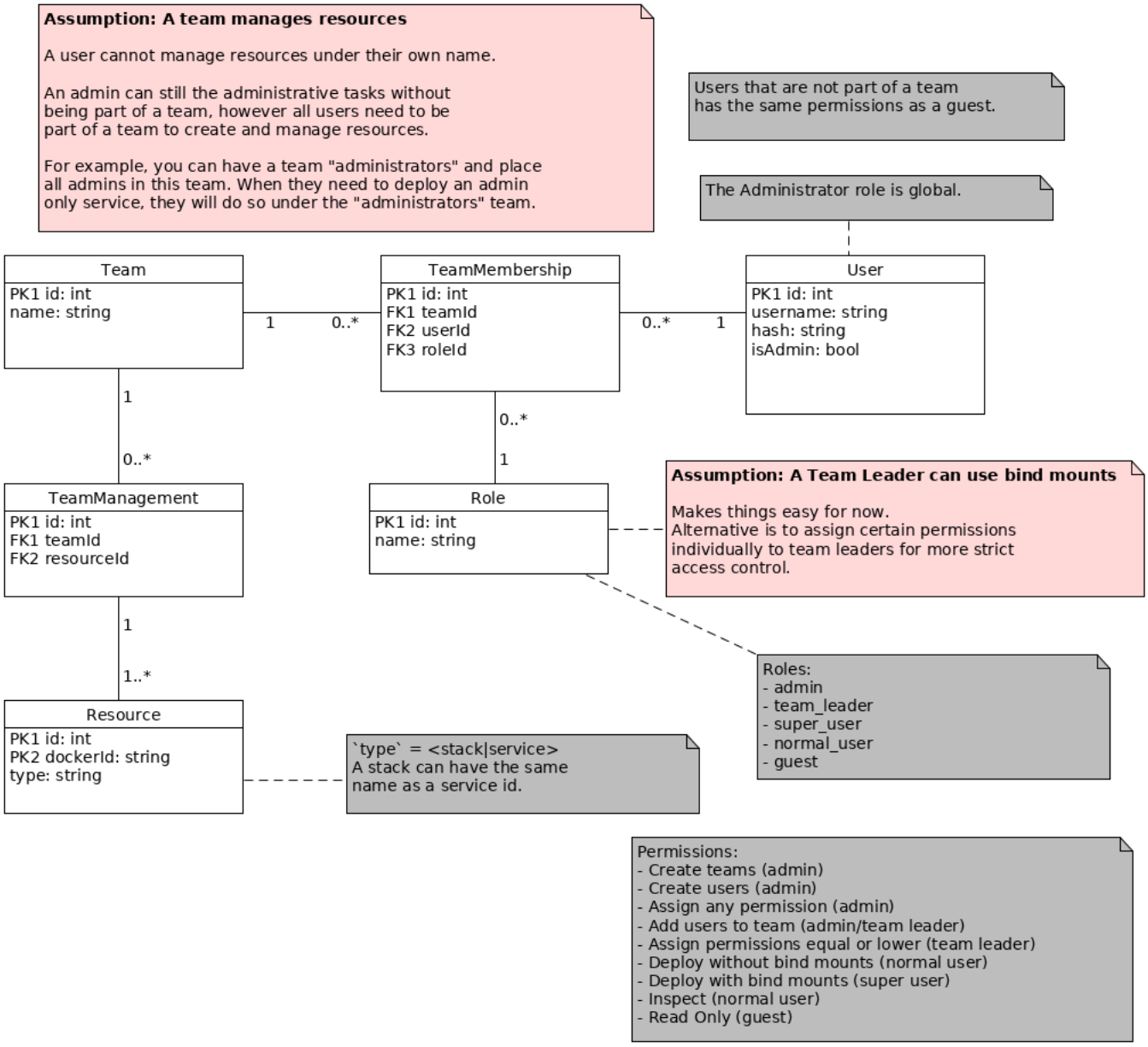

Figure 3: ERD diagram for the Docks API database

3.4 Architectural Styles

The User Interface uses the Model View Controller architecture. Nodes and Containers have a data model. The user interacts with the view to manipulate the data model. The view is updated when the data model retrieves data using an N-Tier architecture. The 3-Tier architecture can be seen by the actor interacting with the view, the request is then delegated to the models, which in turn communicate with other objects to retrieve and set the required data from the Docker API server and Docker Swarm.

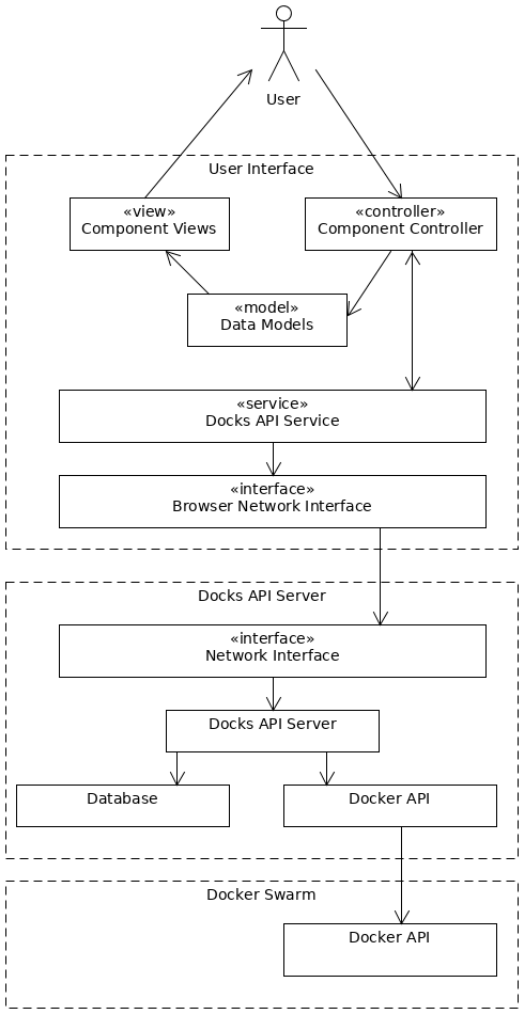

Figure 4: MVC and 3-Tier Architecture diagram for the Docks system