



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	2
1.1	Purpose	2
1.2	Product Scope	2
1.3	Definitions, acronyms and abbreviations	2
1.4	UML Domain Model	3
2	Functional Requirements	3
2.1	Users	3
2.2	Subsystems	4
2.2.1	Authentication	4
2.2.2	WebHooks and Docker Events	5
2.2.3	Stack API Extension for Docker	5
2.2.4	API Proxy for Docker	8
2.2.5	Frontend Docks API service	8
2.2.6	Docker Management Functions	8
2.3	Specific Requirements	8
2.3.1	WebHooks and Docker Events	8
2.3.2	Authentication	9
2.3.3	Stack API Extension for Docker	9
2.3.4	API proxy for Docker	9
2.3.5	Docker Management Functions	9
2.3.6	Frontend Docks API service	10
3	System	10
3.1	Interfaces	10
3.1.1	Software Interfaces	10
3.2	System Configuration	10
3.3	Architectural Styles	11

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 and managing a Docker swarm
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)”
Swarm	A tool to schedule and clump docker nodes into a single virtual machine which is easier to use and maintain.
Stack	A group of Docker services that make up an application.
Service	A collection of Docker containers of the same images.
Nodes	Any virtual or physical machines that run Docker and are part of a swarm.

1.4 UML Domain Model

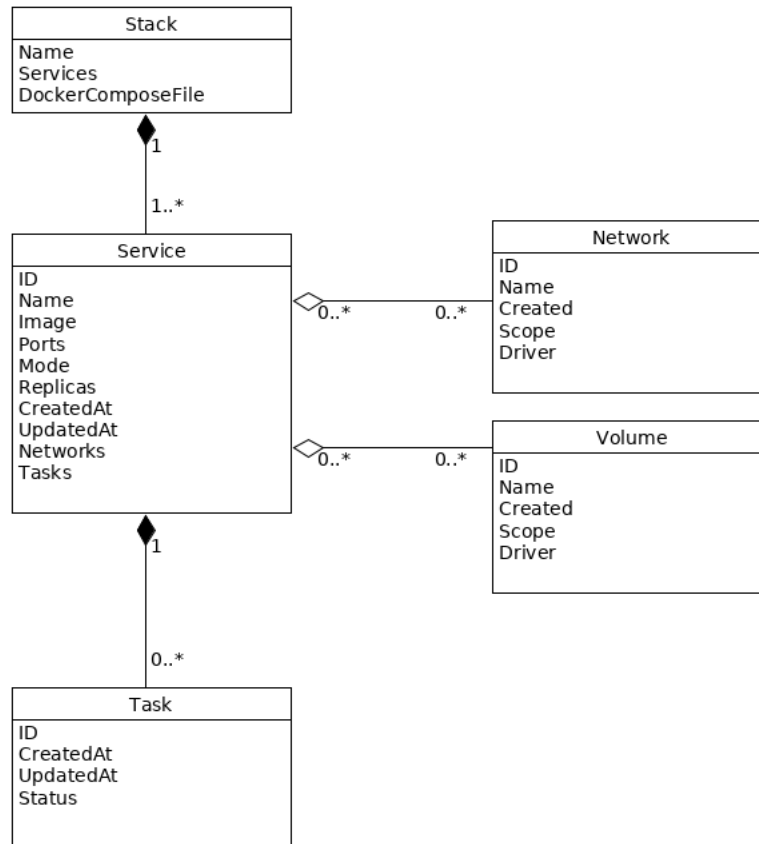


Figure 1: UML Domain Diagram for the Docks System

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

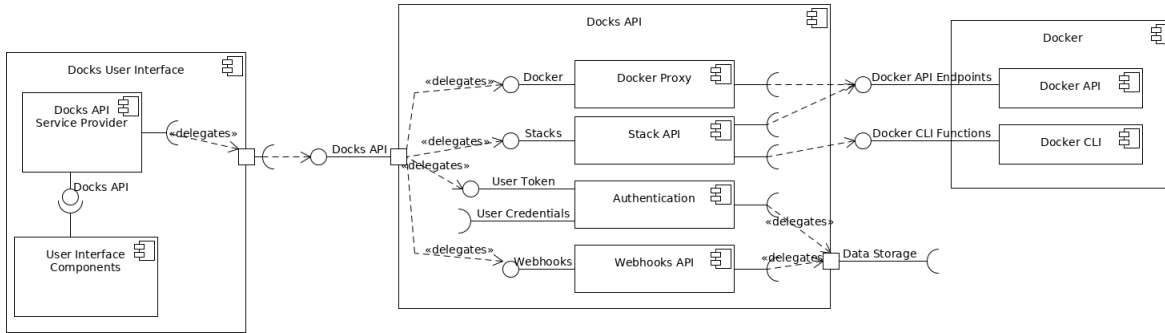


Figure 2: Component Diagram for [Docks](#)

[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. Their specific requirements will be enumerated

2.2.1 Authentication

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

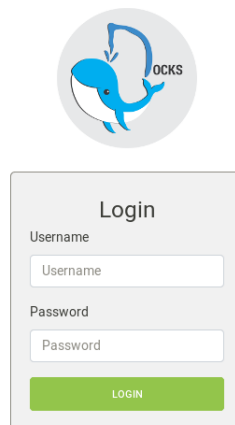


Figure 3: Screenshot of User Login

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 data to the stored WebHooks.

The screenshot shows the 'Webhooks' page in the Docks application. The left sidebar contains a navigation menu with categories: Applications (Tasks, Services, Stacks), Infrastructure (Nodes, Networks), Data (Volumes), Users (Users), and Webhooks (Webhooks). The main content area is titled 'Webhooks' and contains a form for creating or editing a webhook. The form includes a 'Name' field, a 'Webhook URL' field, and several checkboxes for selecting the scope of the webhook: Volumes, Network, Service, Node, and Image.

Figure 4: Screenshot of WebHooks

2.2.3 Stack API Extension for Docker

The Stack API Extension for Docker is part of the [Docks](#) API. Since Docker lacks an API for managing Stacks, it has to be implemented by the [Docks](#) API. This will enable [Docks](#) UI and third party services to view and manage Docker stacks.

The screenshot shows the 'Stacks' page in the Docks application. The left sidebar is the same as in Figure 4. The main content area is titled 'Stacks' and features a search bar, a 'DEPLOY STACK' button, and a table of existing stacks. The table has two columns: 'Stack Name' and 'Services Count'. It lists two stacks: 'docks' with 2 services and 'docks-api' with 1 service. A footer bar indicates '0 selected / 2 total'.

Stack Name	Services Count
docks	2
docks-api	1

Figure 5: Screenshot of stacks

Docks

LOGOUT

Applications

Tasks

Services

Stacks

 Infrastructure

Nodes

Networks

 Data

Volumes

 Users

Users

 Webhooks

Webhooks

Search Nodes

Q

Name	Engine version	IP address	Availability	Role	State	CPUs	Memory
trashPanda	17.05.0-ce	192.168.101.247	active	manager	ready	4	3.76 GB
1 total							

Figure 6: Screenshot of Nodes

Docks

LOGOUT

Search Volumes

Q

CREATE NEW VOLUME

Name	Driver
7279c3ac138b85072ce78011e133a005453db981623bcd7f1fba1fb893e4e10d	local
docksapl_docks_data	local
fa25bb0f7794c92a84c5f37fe7700d0bd00d4d57d3b2772983769daf3996d267	local
247ad0364a13fb101a14bc9ccb70372821b30be4507fcc3cf6ad1dae457c805	local
45c03e2d1bbfd76997c678fb7f7613672295bd3b9f7ead013a1303f72f011a63	local
92fbd08c8006466e3e5c9c12d0d1f94941d441ba093eaa23ff3f044bb93b3512	local
d36adad35e7f7a467e13123095ea9924344a06d944075b1a23aad93236119c47	local
031fc8761a0b54351aa000e5532606175097e2972437ff1555c28a80aea5f174	local
45e5041a18b938740c8ad72daa30ba4f0c7f0936b69ea46145668f8e91222f4b	local

Figure 7: Screenshot of Volumes

Docks

LOGOUT

Applications

Tasks

Services

Stacks

Infrastructure

Nodes

Networks

Data

Volumes

Users

Webhooks

Search Services

Q

CREATE SERVICE

Name	Image	Mode	Replicas	Ports
docks-api_db	postgres:latest	vip	1	-
docks_api	tripleparity/docks-api:latest	vip	1	8080
docks_db	postgres:latest	vip	1	-

0 selected / 3 total

Figure 8: Screenshot of Services

Docks

LOGOUT

Applications

Tasks

Services

Stacks

Infrastructure

Nodes

Networks

Data

Volumes

Users

Webhooks

Users

CREATE USER

List of users

Filter username...

Username	Operations
admin	<div>EDIT</div> <div>ENABLE TWO-FACTOR</div> <div>DELETE</div>

1 total

Figure 9: Screenshot of Users

Name	Driver
ingress	overlay
docks_default	overlay
docks-api_default	overlay
bridge	bridge
docker_gwbridge	bridge
appichapsapi_default	bridge
appichaps-network-private	bridge
host	host
none	null

9 total

Figure 10: Screenshot of Network

2.2.4 API Proxy for Docker

The API proxy for Docker is part of the [Docks](#) API. It provides authenticated access to the Docker API through the [Docks](#) API. The Docker API is not exposed directly to the world, rather authenticated users may send requests to the [Docks](#) API to be transparently forwarded to the private Docker API.

2.2.5 Frontend [Docks](#) API service

The [Docks](#) API service is part of the frontend ([Docks](#) UI). It acts as the interface between the graphical frontend components and the [Docks](#) HTTP API. This layer of abstraction means that the network logic is hidden from components that need to interact with the [Docks](#) API.

2.2.6 Docker Management Functions

Docker will be managed from the [Docks](#) UI, through the authenticated [Docks](#) API. The user should be able to perform common operations from [Docks](#) UI such as deploying an application and viewing its state.

2.3 Specific Requirements

2.3.1 WebHooks and Docker Events

R1.1. The system shall allow a user to add new outgoing WebHooks

R1.2. The system shall display a list of added WebHooks

R1.3. The system shall allow a user to remove a WebHook

R1.4. The system shall allow a user to specify the type of events to send to the WebHook

R1.5. The system shall monitor all Docker events and send the relevant event data to the respective WebHook

R1.6. The system shall send a Slack notification to WebHooks that should receive node events

2.3.2 Authentication

- R2.1. The system shall allow an authorized user to interact with the Docks API
- R2.2. The system shall provide the ability to use two factor authentication as described in RFC 6238
- R2.3. The system shall provide a global administrative account role without restrictions
- R2.4. The system shall provide the following user management features to be used by administrative accounts
 - R2.4.1. Create new administrative user
 - R2.4.2. Remove user
 - R2.4.3. Update user password
 - R2.4.4. Enable and disable two factor authentication

2.3.3 Stack API Extension for Docker

- R3.1. The system shall provide the ability to deploy new stacks given the stack name and docker-compose file
- R3.2. The system shall provide the ability to return deployed stacks along with the number of services in each stack
- R3.3. The system shall provide the ability to update a stack
- R3.4. The system shall provide the ability to remove a stack
- R3.5. The system shall not allow a stack to be created if it already exists
- R3.6. The system shall not allow a stack to be updated if it does not exist
- R3.7. The system shall provide the ability to return the services that are part of a given stack

2.3.4 API proxy for Docker

- R4.1 The system shall only forward requests to the Docker API if the request was made by an authenticated user
- R4.2 The system shall not modify content forwarded from the Docker API to the user
- R4.3 The system shall not modify requests forwarded from the user to the Docker API
- R4.4 The system shall forward error messages from the Docker API to the user

2.3.5 Docker Management Functions

- R5.1. The system shall display all nodes
- R5.2. The system shall display all stacks
- R5.3. The system shall display all services
- R5.4. The system shall display all tasks
- R5.5. The system shall display all networks
- R5.6. The system shall display all volumes
- R5.7. The system shall allow a user to upload a docker-compose file to deploy a Stack
- R5.8. The system shall allow a user to remove a stack from the swarm
- R5.9. The system shall display the tasks that are running in a service
- R5.10. The system shall allow a user to view the log of a service

- R5.11. The system shall allow a user to update a stack using a docker-compose file
- R5.12. The system shall allow a user to delete a volume
- R5.13. The system shall allow a user to delete a network

2.3.6 Frontend Docks API service

- R6.1. The system shall provide the interface for all requirements stated in the "Docker Management Functions" section above
- R6.2. The system shall provide meaningful error message to the user

3 System

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.

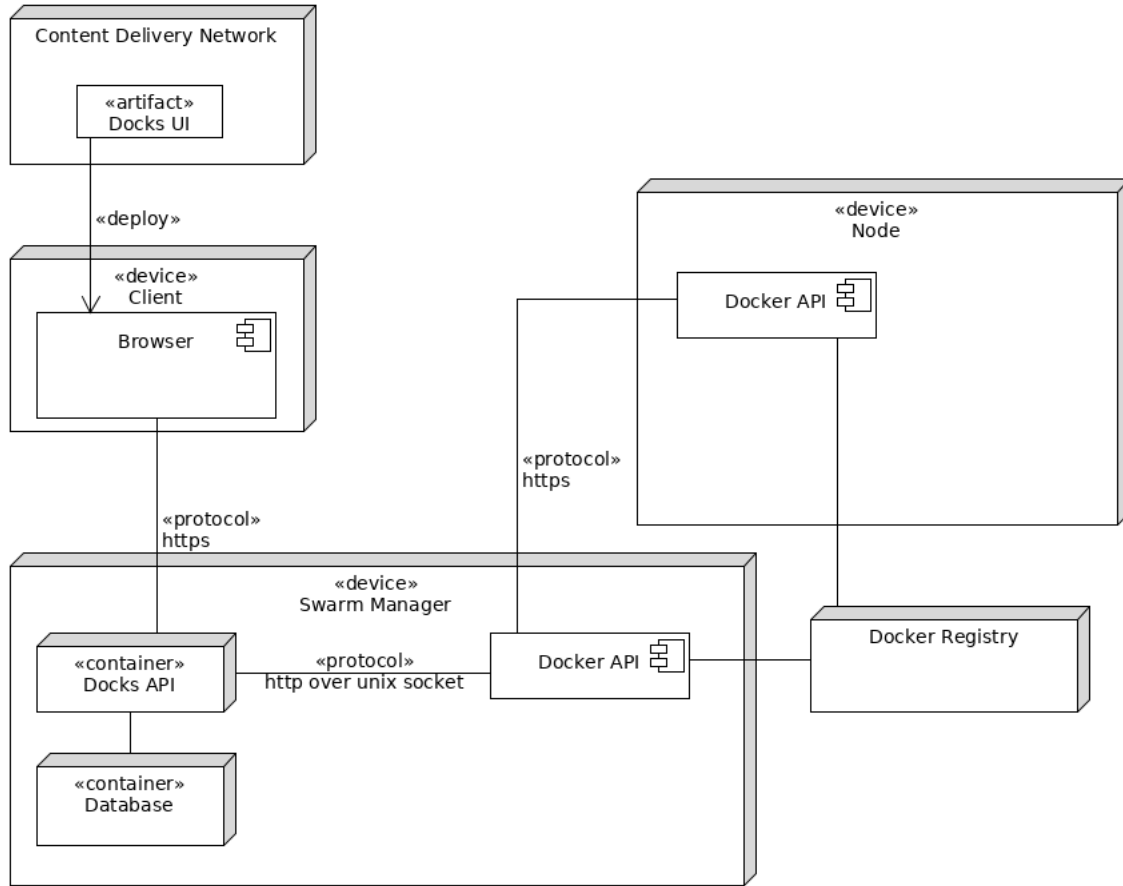


Figure 11: UML Deployment Diagram for the Docks System

3.3 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.

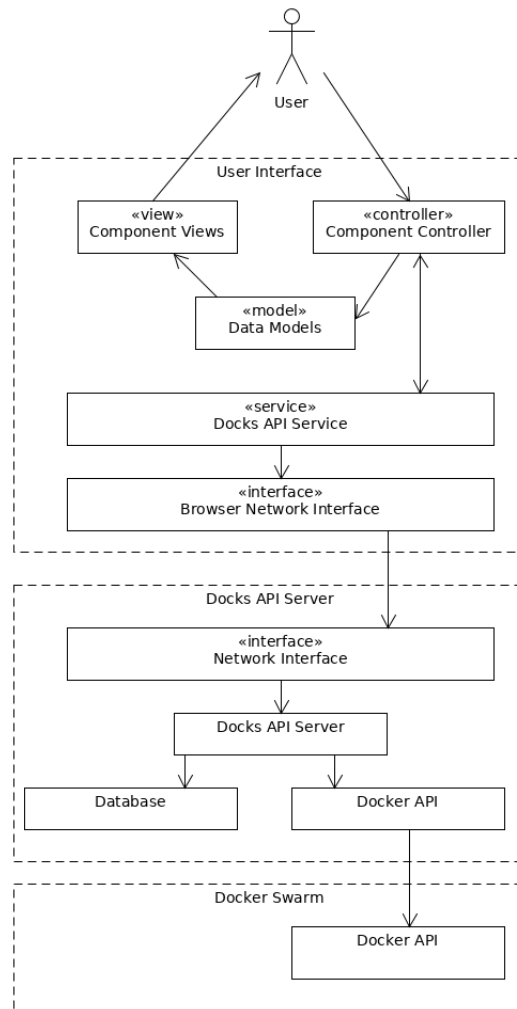


Figure 12: MVC and 3-Tier Architecture diagram for the Docks system