



# Docker Implementation

## Railway Administration and Information Logical System

*RAILS for Model Railroads*

David Bristow  
Version 2.0.1  
August 5, 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Microservices Design Components . . . . .	2
1.1.1	IoT Components . . . . .	3
1.1.2	DS Components . . . . .	4
1.1.3	SPA Components . . . . .	4
1.1.4	Reverse Proxy . . . . .	4
1.2	Docker . . . . .	4
1.3	Docker Compose . . . . .	6
1.4	RAILS Docker Implementation . . . . .	7
<b>2</b>	<b>Setup and Run</b>	<b>8</b>
2.1	Docker Installation . . . . .	8
2.2	Docker Compose Installation . . . . .	8
2.3	Broker Configuration . . . . .	8
2.4	Create the RAILS Docker Environment . . . . .	9
2.5	Run the RAILS Docker Containers . . . . .	9
2.6	Stop and Remove the RAILS Docker Containers . . . . .	10
<b>A</b>	<b>YAML File</b>	<b>11</b>
<b>B</b>	<b>Example Setup and Run Commands</b>	<b>17</b>
B.1	Docker in a Windows Environment . . . . .	17
B.2	Docker in a Linux Environment . . . . .	24
B.3	Docker in a MacOS Environment . . . . .	28
<b>C</b>	<b>Docker Commands</b>	<b>31</b>
	<b>Glossary</b>	<b>33</b>

# Chapter 1

## Introduction

Railway Administration and Information Logical System (RAILS) is a software model and implementation of an automated system to assist the model railroader achieve realism in the operation of a model railroad. The model then drives the development of hardware and or software.

### 1.1 Microservices Design Components

Figure 1.1 shows the microservices components that make up the design of RAILS.

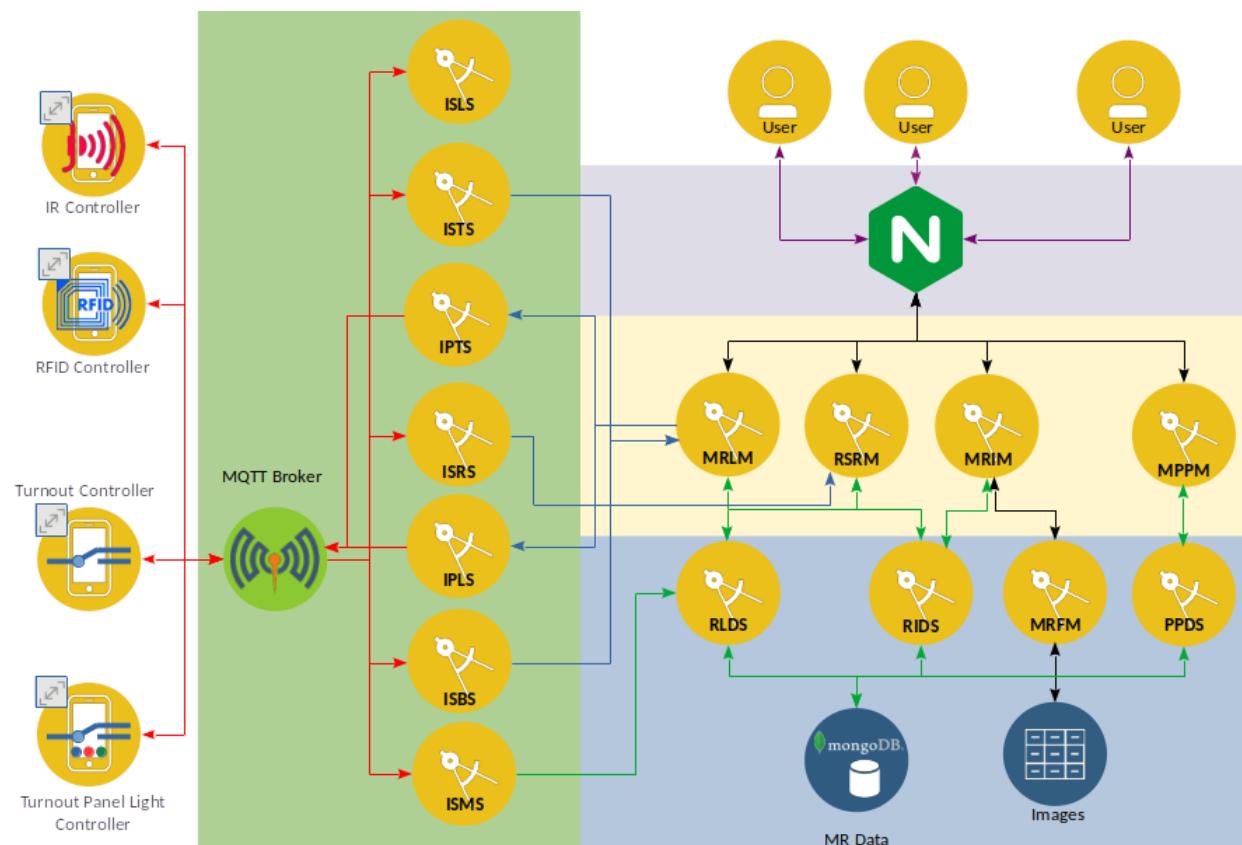


Figure 1.1: Microservices Component Architecture

The microservices design components are divided into three sets:

- Internet of Things (IoT) components, which are highlighted with the light green colored background in Figure 1.1.
- Data Services (DS) components, which are highlighted with the light blue colored background in Figure 1.1.
- Single Page Application (SPA) components, which are highlighted with the buff colored background in Figure 1.1.

### 1.1.1 IoT Components

The components of the IoT set of design components are subdivided into:

- Micro Controllers using the Message Queuing Telemetry Transport (MQTT) protocol:
  - Radio Frequency Identification (RFID) Controller processes RFID tags obtained from a RFID reader and then publishes the value
  - Turnout Controller subscribes to turnout commands then to act on the command to cause the turnout to move. It then publishes the state of the turnout
  - infrared (IR) Controller (in planning) processes IR sensors and publishes their values
- MQTT Broker is the heart of any publish/subscribe protocol, is responsible for receiving messages, posting to designated topics and sending messages to clients subscribing to topics.
- The subscribers and publishers bridge the MQTT elements with the Graphic User Interface (GUI) applications:
  - IoT Publisher Turnout Panel Light Services (IPLS) publishes turnout panel light commands a Turnout Panel Controller
  - IoT Publisher Turnout Services (IPTS) publishes turnout commands to a Turnout Controller
  - IoT Subscriber Turnout Panel Button Services (ISBS) subscribes to push button events and pushes them via a web-socket to the MRLM component
  - IoT Subscriber Location Services (ISLS) (in planning) IoT subscribes to topics that provide location information i.e., IR Sensors and RFID sensors
  - IoT Subscriber Micro-controller Services (ISMS) subscribes to micros and adds or updates micros collection in RAILS. It also subscribes to micro heartbeats.
  - IoT Subscriber RFID Services (ISRS) subscribes to RFID tags and pushes them via a web-socket to the Rollingstock RFID Manager (RSRM) component
  - IoT Subscriber Turnout Services (ISTS) subscribes to turnout switch closures and pushes them via a web-socket to the Model Railroad Layout Manager (MRLM) component

### 1.1.2 DS Components

DS consist of all the components that handle and or store the model railroad data:

- MR Data – the document repository, MongoDB, to store complete collections of items such as rolling stock, industries (producers and consumers), track elements, turnouts, projects, purchases, etc.
- Railroad Inventory Data Services (RIDS) provides Representational State Transfer (REST) access to railroad inventory documents
- Plans and Purchases Data Services (PPDS) provides REST access to model railroad projects and purchases documents
- Railroad Layout Data Services (RLDS) provides REST access to model railroad layout documents
- Model Railroad File Manager (MRFM) provides the user the ability to upload image files for the use by the Model Railroad Inventory Manager (MRIM) component
- Images is the file store for the images uploaded by MRFM component and used by the MRIM component

### 1.1.3 SPA Components

GUI applications that provide user access to RAILS:

- RSRM, this SPA allows a user to match a RFID value to a rolling stock road name and number
- MRIM, this SPA allows a user to create, update and delete model railroad assets, such as rolling stock
- Model Projects and Purchase Manager (MPPM), this SPA allows a user to enter information about their projects and purchases
- MRLM, this SPA allows a user to enter information about their layout and control elements of it

### 1.1.4 Reverse Proxy

A server that sits in front of the Single Page Applications (SPAs), which allows users to access the SPAs from any browser on any Personal Computer (PC) in the network.

## 1.2 Docker

In the context of microservices, containers are used to package the code, libraries, and configuration files needed for a microservice to run as a single, executable unit. This makes it easy to deploy and run microservices in different environments, such as on-premises servers or in the cloud. By packaging all the dependencies for a microservice in a container, it can be guaranteed to run the same way regardless of the environment it's deployed in. This helps to reduce issues caused by

differences between development, testing, and production environments. Containers also make it easy to scale up or down the number of instances of a microservice running, and to update or rollback a microservice without affecting other services.

Docker is a platform that makes it easy to create, deploy, and run applications in containers. It provides a [command-line interface \(CLI\)](#) and a set of [Application Program Interfaces \(APIs\)](#) that make it simple to work with containers. Docker in conjunction with microservices is used to package and deploy each service in its own container. This allows each service to be managed, deployed, and scaled independently of other services. The Docker platform provides a number of tools to help manage and orchestrate the containers that make up a microservice-based application, such as Docker Compose and Docker Swarm.

Docker is an open-source platform that enables developers to automate the deployment, scaling, and management of applications using containerization. Containers are lightweight, isolated environments that encapsulate an application and all its dependencies, including libraries, frameworks, and other runtime components. Docker allows the user to package an application into a container image, which can then be run consistently across different environments, such as development, testing, and production.

Docker provides the following features:

- Docker containers are portable and can run on any system that supports Docker, regardless of the underlying operating system or infrastructure. This eliminates the "works on my machine" problem and ensures consistent behavior across different environments.
- Containers provide a high level of isolation, ensuring that applications and their dependencies are encapsulated and do not interfere with each other. This isolation improves security, as well as prevents conflicts between different software components.
- Docker containers are lightweight and share the host system's operating system kernel. This means they require fewer resources compared to traditional [virtual machines \(VMs\)](#). Multiple containers can run on the same host without significant performance overhead.
- Docker makes it easy to scale applications horizontally by running multiple instances of containers across different hosts or a cluster of machines. This enables efficient utilization of resources and helps handle increased workload demands.
- Docker allows versioning of container images, making it easier to track changes and roll back to a previous version if needed. This simplifies the deployment and update process, reducing the risk of application downtime.
- Docker has a large and active community, resulting in a vast ecosystem of pre-built container images available from Docker Hub and other registries. These images can be easily pulled and used as a base for building and deploying applications, saving development time.

The key aspects of Docker are:

- **Docker Engine:** The core component of Docker, responsible for running and managing containers. It includes the Docker daemon, which runs on the host system, and the Docker CLI, which provides a command-line interface for interacting with the daemon.

- **Docker Image:** A read-only template that contains the application code, runtime, libraries, and other dependencies required to run a container. Images are used to create containers and can be shared and reused across different environments.
- **Docker Container:** A runnable instance of a Docker image. Containers are isolated environments that run applications and their dependencies, providing consistency and portability across different systems.
- **Dockerfile:** A text file that contains instructions for building a Docker image. It specifies the base image, environment variables, commands to run, and other configuration settings needed to create the image.
- **Docker Hub:** A cloud-based registry service provided by Docker for sharing and distributing container images. It hosts a vast collection of public and private images that can be used as a base for building and deploying applications.'

For additional information on Docker see the [official website](#).

### 1.3 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the services, networks, and volumes required for a multi-container application, making it easy to manage complex deployments. Docker Compose allows developers to define a multi-container application in a single file, then use the [CLI](#) to create and start all the services defined in the file with a single command. This simplifies the process of managing the deployment and scaling of microservices, as well as the configuration of the networks and volumes needed to run the application. Docker Compose also provides a number of commands for managing the lifecycle of the application, such as starting, stopping, and scaling the services, as well as viewing the status of the running containers. This makes it easy to develop, test, and deploy multi-container applications, and to manage the dependencies between the different services that make up the application. Key features of Docker Compose include:

- **Multi-container Applications** It helps you define and manage applications composed of multiple interacting Docker containers. Each container can represent a different service within your application.
- **Simplified Configuration** By using a docker-compose.yml file, you specify the configuration for each service, including the image to use, ports, volumes, networks, and environment variables. This centralizes configuration and makes it easier to manage complex applications.
- **Streamlined Workflow** Docker Compose provides commands to easily build, run, stop, and scale your multi-container application. You can manage all your services with a single command instead of managing individual Docker commands for each container.
- **Declarative Syntax** The docker-compose.yml file uses a declarative syntax, meaning you specify the desired state of your application (services, configurations), and Docker Compose takes care of creating and managing the containers to achieve that state.
- **Dependency Management** Docker Compose automatically handles dependencies between services. If one service depends on another, Compose ensures the dependent service starts only after the required service is up and running.

For additional information on Docker Compose see the [official website](#).

## 1.4 RAILS Docker Implementation

Docker based components are the partial realization of RAILS software model of an automated system. These components are the micro-services that are containerized and typically run on a PC (Linux, Mac, or Windows) or Single Board Computer running Linux.

Table 1.1 lists the Docker images available from the [Docker Hub repository](#). These images are the microservices depicted in Figure 1.1.

Table 1.1: Docker Images Table

Image	Name	Port	Version	Date
<hr/> ----- SPAs -----				
dbristow/mppm	<b>Model Projects and Purchase Manager</b>	3008	4.0.0	2025-07-30
dbristow/mrim	<b>Model Railroad Inventory Manager</b>	3001	5.0.0	2025-07-30
dbristow/mrlm	<b>Model Railroad Layout Manager</b>	3004	3.3.2	2025-07-30
dbristow/rsrm	<b>Rollingstock RFID Manager</b>	3002	6.0.0	2025-07-30
<hr/> ----- Data Services -----				
dbristow/mrfm	<b>Model Railroad File Manager</b>	3030	2.4.6	2025-07-22
dbristow/ppds	<b>Plans and Purchases Data Services</b>	3007	2.3.19	2025-07-30
dbristow/rids	<b>Railroad Inventory Data Services</b>	3000	2.3.19	2025-07-30
dbristow/rlds	<b>Railroad Layout Data Services</b>	3006	2.1.39	2025-07-30
<hr/> ----- IoT Services -----				
dbristow/ipls	<b>IoT Publisher Turnout Panel Light Services</b>	3013	1.1.15	2025-07-22
dbristow/ipts	<b>IoT Publisher Turnout Services</b>	3011	2.1.15	2025-07-22
dbristow/isbs	<b>IoT Subscriber Turnout Panel Button Services</b>	3012	1.1.18	2025-07-22
dbristow/isms	<b>IoT Subscriber Micro-controller Services</b>		3.0.13	2025-07-23
dbristow/isrs	<b>IoT Subscriber RFID Services</b>	3005	2.0.0	2025-07-30
dbristow/ists	<b>IoT Subscriber Turnout Services</b>	3010	1.4.15	2025-07-22

The RAILS Docker implementation is a multi-container application that uses Docker Compose to define and run the services required for the RAILS application. The Docker Compose file specifies the services, networks, and volumes needed to run the RAILS application, making it easy to manage the deployment and scaling of the microservices.

# Chapter 2

## Setup and Run

### 2.1 Docker Installation

The RAILS SPAs are implemented as Docker containers. The Docker containers are built and pushed to Docker Hub. The Docker containers are then pulled from Docker Hub and run on the host machine. The host machine must have Docker installed. Docker is a platform for developing, shipping, and running applications in containers. Docker can be installed on Windows, macOS, and Linux. The installation instructions for Docker can be found at <https://docs.docker.com/get-docker/>.

### 2.2 Docker Compose Installation

Docker Compose is a tool for defining and running multi-container Docker applications. Docker Compose uses a YAML file to configure the application's services. The installation instructions<sup>1</sup> for Docker Compose can be found at <https://docs.docker.com/compose/install/>. The Docker Compose file for the RAILS SPAs is shown in Appendix A.

### 2.3 Broker Configuration

The RAILS SPAs use the Mosquitto broker to communicate with each other. The Mosquitto broker is an open-source message broker that implements the MQTT protocol. The Mosquitto broker must be configured to allow the RAILS SPAs to communicate with each other. The Mosquitto broker configuration file must be edited to allow the RAILS SPAs to communicate with each other. The configuration file is found in the virtual storage whose volume name is "mosquitto". To establish and modify the configuration file the following steps are taken:

1. Create a Docker volume for the Mosquitto broker.

```
docker volume create --name mosquitto
```

2. Run a Docker container for the first time the Mosquitto broker, which will create the initial configuration file.

---

<sup>1</sup>Note that for Microsoft Windows the installation of the GUI Docker Desktop automatically installs Docker Compose.

```
docker run -it --name myMqttBrkr -p 1883:1883 -p 9001:9001 --rm  
-v mosquitto:/mosquitto -d eclipse-mosquitto
```

3. Stop the broker container.

```
docker stop myMqttBrkr
```

4. Locate the configuration file in the "mosquitto" volume.

```
docker inspect mosquitto
```

5. Edit the configuration file modifying or adding the following lines:

```
listener 1883  
allow_anonymous true  
socket_domain ipv4
```

## 2.4 Create the RAILS Docker Environment

The RAILS SPAs are implemented as Docker containers that require an environment to run in. To create that environment, the following steps are taken:

1. Create a Docker volume for the Rails database.

```
docker volume create --name myRailsDb
```

2. Create a Docker volume for the Rails images.

```
docker volume create --name myRailsImages
```

3. Create a Docker network for the Rails containers to communicate over.

```
docker network create myRailsNet
```

## 2.5 Run the RAILS Docker Containers

The Docker containers are pulled from Docker Hub and run on the host machine. To run all of the RAILS Docker containers, either create the YAML file from Appendix A or retrieve the YAML file from the RAILS GitHub repository found at [my GitHub repository](#). The YAML file is used to run the RAILS Docker containers. to run all four RAILS Docker containers, the following command in the directory where the YAML file is located:

```
docker-compose up -d
```

If the RAILS Docker SPA containers are to be run individually, the following commands are used:

```
docker-compose up -d myMrim  
docker-compose up -d myRsrm  
docker-compose up -d myMppm  
docker-compose up -d myMrlm
```

Point a browser, on the host machine, to the following URLs to access the RAILS SPAs:

- MRIM SPA - <http://localhost/mrim> (or `http://< MACHINE_IP >/mrim/`)
- RSRM SPA - <http://localhost/rsrm> (or `http://< MACHINE_IP >/rsrm/`)
- MPPM SPA - <http://localhost/mppm> (or `http://< MACHINE_IP >/mppm/`)
- MRLM SPA - <http://localhost/mrlm> (or `http://< MACHINE_IP >/mrlm/`)

Where `< MACHINE_IP >` is the IP address of the host machine running the Docker containers. The RAILS SPAs can be accessed from any browser on any Personal Computer (PC) in the network.

## 2.6 Stop and Remove the RAILS Docker Containers

To stop and remove the RAILS Docker containers, the following command is used:

```
docker-compose down
```

See Appendix B for screenshots of the Docker commands used to create the Docker environment and run the RAILS SPAs.

## Appendix A

### YAML File

```
1 # docker-compose.yaml
2 # &copy; David Bristow, 2020-2025
3 # VERSION 2.0.0
4 # LICENSE
5 # The code in this repository is licensed under the Apache License, Version
6 # 2.0 (the "License");
7 # you may not use this file except in compliance with the License.
8 # You may obtain a copy of the License at
9 #
10 # http://www.apache.org/licenses/LICENSE-2.0
11 #
12 # Unless required by applicable law or agreed to in writing, software
13 # distributed under the License is distributed on an "AS IS" BASIS,
14 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 # See the License for the specific language governing permissions and
16 # limitations under the License.
17
18 services:
19   nginx:
20     image: nginx:latest # Use the official Nginx image
21     container_name: myNginxProxy
22     ports:
23       - '80:80' # Expose Nginx on host port 80 (for HTTP)
24       # - '443:443' # Uncomment and configure SSL for HTTPS in nginx.conf
25     volumes:
26       # Mount your custom nginx.conf into the container
27       - ./nginx.conf:/etc/nginx/nginx.conf:ro
28       # If you have separate site configs in conf.d, you'd mount a directory:
29       # - ./nginx_conf.d:/etc/nginx/conf.d:ro
30     networks:
31       - myRailsNet # Connect Nginx to your internal network
32     depends_on:
33       # Nginx should start after all services it proxies to
34       - myMongo
35       - myPpds
36       - myMppm
37       - myRids
```

```
38      - myMrfm
39      - myMrim
40      - myMqttBrkr
41      - myRlds
42      - myIsrs
43      - myIsms
44      - myRsrm
45      - myIsts
46      - myIsbs
47      - myIpst
48      - myIpls
49      - myMrlm
50
51 myMongo:
52   image: 'mongo'
53   container_name: myMongo
54   networks:
55     - myRailsNet
56   # No direct host port exposure needed if only Nginx accesses it
57   # ports:
58   #   - '27017:27017' # Remove this if only Nginx accesses it
59   volumes:
60     - myRailsDb:/data/db
61
62 myPpds:
63   image: 'dbristow/ppds'
64   container_name: myPpds
65   networks:
66     - myRailsNet
67   # No direct host port exposure needed if only Nginx accesses it
68   # ports:
69   #   - '3007:3007' # Remove this
70   depends_on:
71     - myMongo
72
73 myMppm:
74   image: 'dbristow/mppm'
75   container_name: myMppm
76   # No direct host port exposure needed if only Nginx accesses it
77   # ports:
78   #   - '3008:8080' # Remove this
79   networks:
80     - myRailsNet # Add to network so Nginx can find it
81   depends_on:
82     - myPpds
83
84 myRids:
85   image: 'dbristow/rids'
86   container_name: myRids
87   networks:
88     - myRailsNet
89   # No direct host port exposure needed if only Nginx accesses it
90   # ports:
```

```

91      # - '3000:3000' # Remove this
92      depends_on:
93        - myMongo
94
95  myMrfm:
96    image: 'dbristow/mrfm'
97    # Environment variables for myMrfm's *backend* if it has one.
98    # Frontend VITE_ variables are handled at image build time.
99    environment:
100      # These might be for internal communication *within* myMrfm's backend,
101      # or if myMrfm's backend needs to call other services.
102      # If myMrfm is purely a static file server, these might not be needed.
103      - MYMRIM_URI1=http://myMrim:8080 # Use service name for internal Docker
104        network
105      - MYMRIM_URI2=http://myMrim:8080 # Use service name for internal Docker
106        network
107    container_name: myMrfm
108    # No direct host port exposure needed if only Nginx accesses it
109    # ports:
110      # - '3030:3030' # Remove this
111    volumes:
112      - myRailsImages:/usr/djb/src/uploads
113    networks:
114      - myRailsNet # Add to network so Nginx can find it
115
116  myMrim:
117    image: 'dbristow/mrim'
118    container_name: myMrim
119    networks:
120      - myRailsNet
121    # No direct host port exposure needed if only Nginx accesses it
122    # ports:
123      # - '3001:8080' # Remove this
124    depends_on:
125      - myRids
126      - myMrfm
127
128  myMqttBrkr:
129    image: 'eclipse-mosquitto'
130    container_name: myMqttBrkr
131    networks:
132      - myRailsNet
133    # Keep MQTT ports exposed if external clients need to connect directly
134    ports:
135      - '1883:1883'
136      - '9001:9001'
137    volumes:
138      - mosquitto:/mosquitto
139
140  myRlds:
141    image: 'dbristow/rlds'

```

```
142     - myRailsNet
143     # No direct host port exposure needed if only Nginx accesses it
144     # ports:
145     #   - '3006:3006' # Remove this
146     depends_on:
147       - myMongo
148
149 myIsrs:
150   image: 'dbristow/isrs'
151   container_name: myIsrs
152   networks:
153     - myRailsNet
154   # No direct host port exposure needed if only Nginx accesses it
155   # ports:
156   #   - '3005:3005' # Remove this
157   depends_on:
158     - myMqttBrkr
159
160 myIsms:
161   image: 'dbristow/isms'
162   container_name: myIsms
163   networks:
164     - myRailsNet
165   depends_on:
166     - myMqttBrkr
167     - myRlds
168
169 myRsrm:
170   image: 'dbristow/rsrm'
171   container_name: myRsrm
172   networks:
173     - myRailsNet
174   # No direct host port exposure needed if only Nginx accesses it
175   # ports:
176   #   - '3002:8080' # Remove this
177   depends_on:
178     - myRids
179     - myRlds
180     - myIsrs
181     - myIsms
182
183 myIsts:
184   image: 'dbristow/ists'
185   container_name: myIsts
186   networks:
187     - myRailsNet
188   # No direct host port exposure needed if only Nginx accesses it
189   # ports:
190   #   - '3010:3010' # Remove this
191   depends_on:
192     - myMqttBrkr
193
194 myIsbs:
```

```
195     image: 'dbristow/isbs'
196     container_name: myIsbs
197     networks:
198       - myRailsNet
199     # No direct host port exposure needed if only Nginx accesses it
200     # ports:
201     #   - '3012:3012' # Remove this
202     depends_on:
203       - myMqttBrkr
204
205 myIpts:
206   image: 'dbristow/ipts'
207   container_name: myIpts
208   networks:
209     - myRailsNet
210   # No direct host port exposure needed if only Nginx accesses it
211   # ports:
212   #   - '3011:3011' # Remove this
213   depends_on:
214     - myMqttBrkr
215
216 myIpls:
217   image: 'dbristow/ipls'
218   container_name: myIpls
219   networks:
220     - myRailsNet
221   # No direct host port exposure needed if only Nginx accesses it
222   # ports:
223   #   - '3013:3013' # Remove this
224   depends_on:
225     - myMqttBrkr
226
227 myMrilm:
228   image: 'dbristow/mrilm'
229   container_name: myMrilm
230   networks:
231     - myRailsNet
232   # No direct host port exposure needed if only Nginx accesses it
233   # ports:
234   #   - '3004:8080' # Remove this
235   depends_on:
236     - myRlds
237     - myIsts
238     - myIsbs
239     - myIpts
240     - myIpls
241
242 networks:
243   myRailsNet:
244     # It's good practice to define the network as internal for security,
245     # unless you explicitly need containers outside this compose file to join
246     # it.
```

```
246     # If `myRailsNet` is truly external (created manually with `docker network
247     # create`),
248     # then keep `external: true`. Otherwise, use `driver: bridge`.
249     external: true # Keep if you've already created it with `docker network
250     # create myRailsNet`
251     # driver: bridge # Use this if you want Docker Compose to manage its
252     # creation
253
254 volumes:
255   myRailsDb:
256     external: true
257   myRailsImages:
258     external: true
259   mosquitto:
260     external: true
```

Listing A.1: docker compose file

## Appendix B

# Example Setup and Run Commands

The following screenshots show the Docker commands used to create the Docker environment and run the RAILS Docker containers.

### B.1 Docker in a Windows Environment

Docker on Windows 11 relies on a component called [Windows Subsystem for Linux 2 \(WSL 2\)](#) to provide a Linux-like environment for running Docker containers. Docker Desktop installer provides two options:

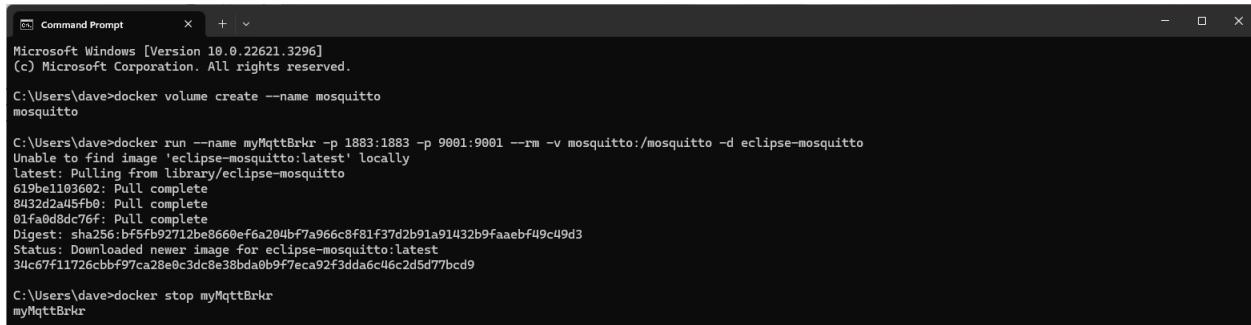
- Use [WSL 2 backend](#): This is the recommended approach for most users. It leverages WSL 2 to create a Linux environment for Docker to run containers.
- Use Hyper-V backend: This is an alternative that uses Hyper-V virtualization technology. However, [WSL 2](#) is generally considered more performant and resource-efficient.

This setup is based on using WSL as it:

- offers better performance for running Linux containers compared to Hyper-V.
- provides a more native Linux experience for Docker, ensuring compatibility with most Linux container images.
- uses the Windows kernel directly, leading to more efficient resource usage.

During Docker Desktop installation with the [WSL 2](#) backend, Docker will automatically configure and set up [WSL 2](#) if it's not already installed. It will also create a default Linux distribution (usually Ubuntu) within [WSL 2](#). Once [WSL 2](#) is set up, Docker Desktop installs the Docker daemon within the [WSL 2](#) environment. This daemon is the core process that manages Docker containers. Docker Desktop provides a Docker [CLI](#) for Windows. This [CLI](#) interacts with the Docker daemon running in the [WSL 2](#) environment through a remote API. So, when you run Docker commands on Windows, they are translated and sent to the Docker daemon in [WSL 2](#) for execution.

In essence, Docker on Windows 11 leverages [WSL 2](#) to create a Linux-like environment for running Docker containers. This approach provides a powerful and performant solution for containerization on Windows, while still offering a familiar Windows experience for interacting with Docker.



```

Command Prompt
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dave>docker volume create --name mosquitto
mosquitto

C:\Users\dave>docker run --name myMqttBrkr -p 1883:1883 -p 9001:9001 --rm -v mosquitto:/mosquitto -d eclipse-mosquitto
Unable to find image 'eclipse-mosquitto:latest' locally
latest: Pulling from library/eclipse-mosquitto
619be1103602: Pull complete
8432d2a45fb0: Pull complete
01fa0d8dc76f: Pull complete
Digest: sha256:bff5fb92712be8660ef6a204bf7a966c8f81f37d2b91a91432b9faaebf49c49d3
Status: Downloaded newer image for eclipse-mosquitto:latest
34c67f11726:cbf97ca28e0c3dc8e38bda0b9f7eca92f3dd46c2d5d77bcd9

C:\Users\dave>docker stop myMqttBrkr
myMqttBrkr

```

Figure B.1: Intial Windows Docker Commands

Docker Desktop with WSL 2 backend stores Docker images and volumes on the Windows file system, accessible from both Windows and WSL 2. This allows for easy sharing of container data between the two environments. However, it simpler to use Docker Desktop GUI to locate and edit the mosquito configuration file. The following screenshots show the location of he configuration file and the editing of the file.

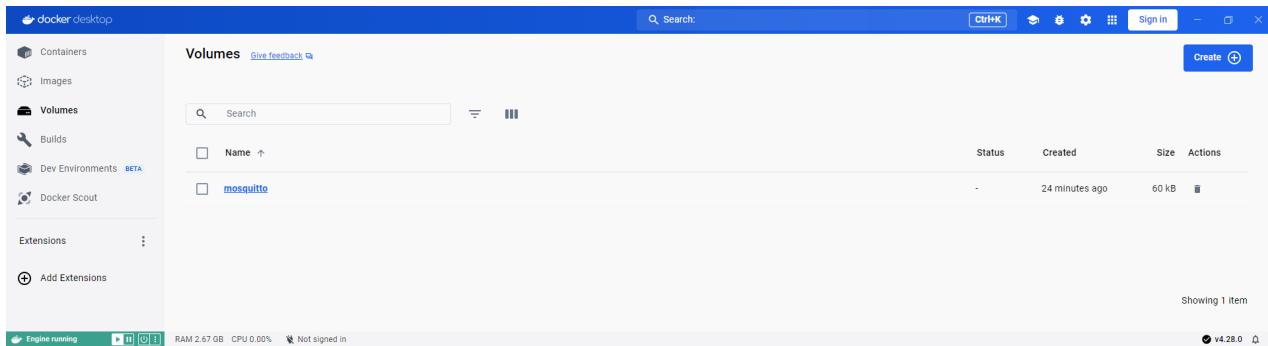


Figure B.2: Docker Desktop Volumes

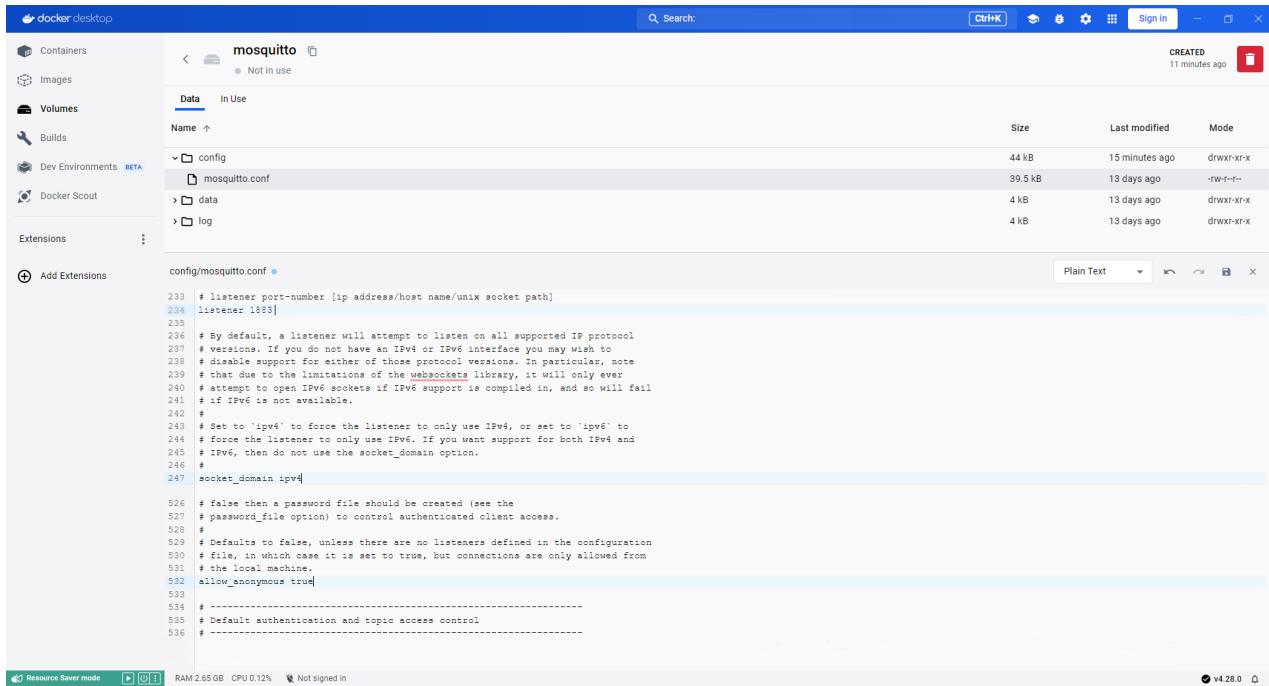


Figure B.3: Mosquitto Configuration File

The following screenshots show the Docker commands used to create the Docker environment and run all of the RAILS Docker containers.

```
C:\Users\Dave>docker volume create --name myRailsDb
myRailsDb

C:\Users\Dave>docker volume create --name myRailsImages
myRailsImages

C:\Users\Dave>docker network create myRailsNet
3bcae2531f0526b71d4c6ccf310c11b7bf76a37be46779a5cf50018b636e8568

C:\Users\Dave>docker-compose up -d
[*] Running 77/25
  ✓ myMppm 9 layers [██████████]  0B/0B    Pulled          19.9s
  ✓ myIsrs 4 layers [████]      0B/0B    Pulled          58.5s
  ✓ myIpls 4 layers [████]      0B/0B    Pulled          55.6s
  ✓ myRids 4 layers [████]      0B/0B    Pulled          35.6s
  ✓ myRsrm 1 layers [ ]        0B/0B    Pulled          33.3s
  ✓ myRlds 4 layers [████]      0B/0B    Pulled          55.6s
  ✓ myPpds 4 layers [████]      0B/0B    Pulled          58.3s
  ✓ myIsbs 5 layers [██████]    0B/0B    Pulled          35.8s
  ✓ myMongo 8 layers [███████]   0B/0B    Pulled          78.0s
  ✓ myIsts 4 layers [████]      0B/0B    Pulled          57.3s
  ✓ myMrM 1 layers [ ]        0B/0B    Pulled          31.1s
  ✓ myMrM 1 layers [ ]        0B/0B    Pulled          34.9s
  ✓ myIpTs 4 layers [████]      0B/0B    Pulled          56.7s
  ✓ myIsMs 4 layers [████]      0B/0B    Pulled          35.5s
  ✓ myMrFm 5 layers [██████]   0B/0B    Pulled          41.2s

[*] Running 16/17
- Network dave_default Created
✓ Container myMrFm Started
✓ Container myMqttBrkr Started
✓ Container myMongo Started
✓ Container myPpds Started
✓ Container myRids Started
✓ Container myRlds Started
✓ Container myIsbs Started
✓ Container myIsrs Started
✓ Container myIpTs Started
✓ Container myIsts Started
✓ Container myIpls Started
✓ Container myMrM Started
✓ Container myIsMs Started
✓ Container myMppm Started
✓ Container myMrM Started
✓ Container myRsrm Started
C:\Users\Dave>
```

Figure B.4: Windows Docker Commands Continued

The following screenshots confirm the Docker containers are running.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2be5d44ef597	dbristow/rsrm	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3002->8880/tcp	myRsrm
0fdcd79e62dd	dbristow/mrM	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3004->8880/tcp	myMrM
ae295173ab17	dbristow/mppm	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3008->8880/tcp	myMppm
533a956c7e6b	dbristow/isms	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3009->8880/tcp	myIsMs
b3739d23e7a7	dbristow/mrM	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3001->8880/tcp	myMrM
b7c095bdeedb	dbristow/ipTs	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3013->3013/tcp	myIpTs
09fde488686a	dbristow/ists	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3010->3010/tcp	myIsts
dd5b99fba84b2	dbristow/rlds	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3006->3006/tcp	myRlds
d38fd3eb5c4b	dbristow/isrs	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3005->3005/tcp	myIsrs
d1f299be5893	dbristow/ipts	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3011->3011/tcp	myIpTs
0d2595b3923a	dbristow/isbs	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3012->3012/tcp	myIsbs
f5ecbfef94320	dbristow/rids	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3009->3009/tcp	myRids
afc913da859c	dbristow/ppds	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3007->3007/tcp	myPpds
54ed8b23822d	eclipse-mosquitto	"/docker-entrypoint..."	About a minute ago	Up About a minute	0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->9001/tcp	myMqttBrkr
8ubj13766d740	dbristow/mrFm	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3030->3030/tcp	myMrFm
3dc5a910ld7c	mongo	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:27017->27017/tcp	myMongo

Figure B.5: Windows Docker PS

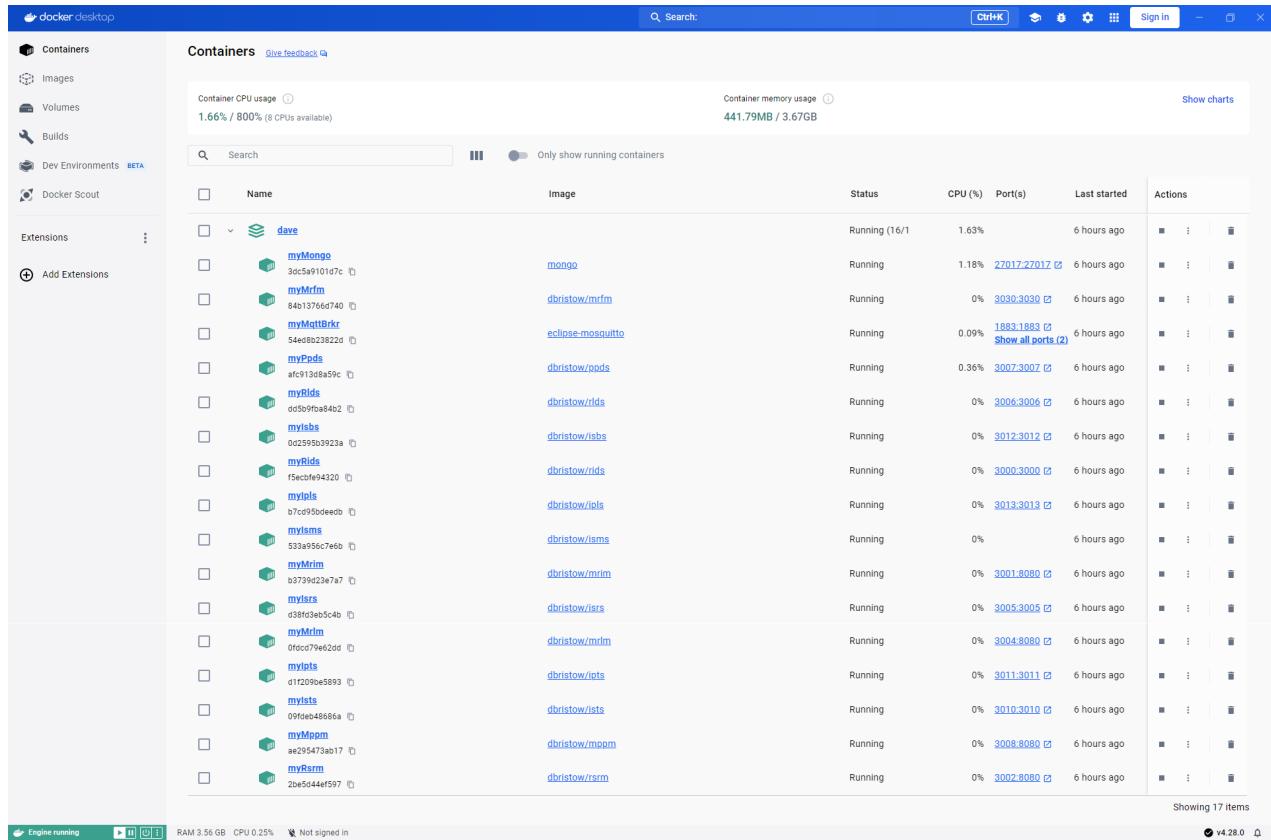


Figure B.6: Windows Docker Desktop Containers

Once the Docker containers are running, the RAILS SPAs can be accessed from a web browser (Chrome). The following screenshots show the RAILS SPAs running in a web browser.

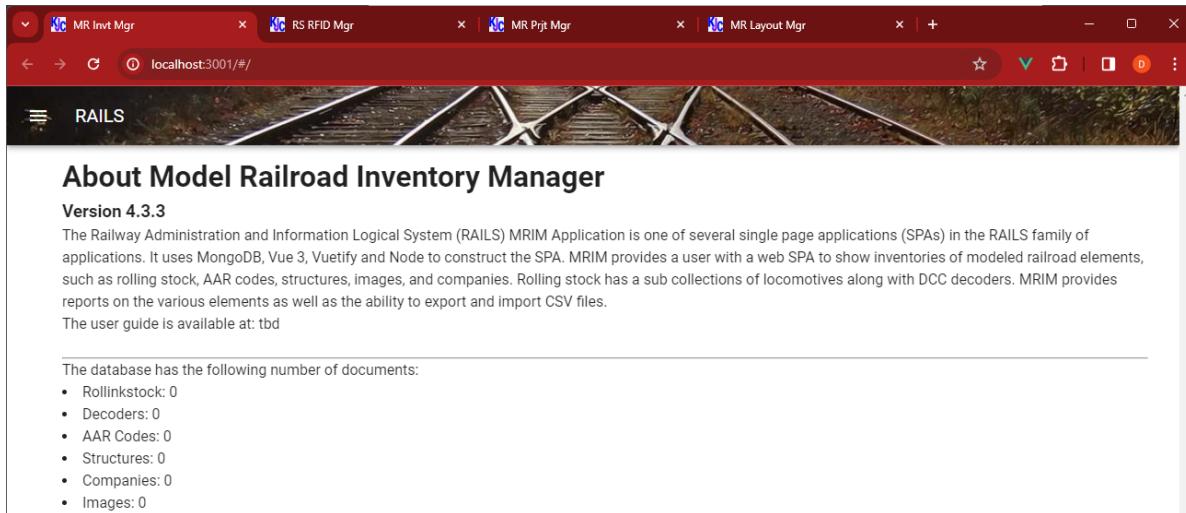


Figure B.7: RAILS MRIM Running

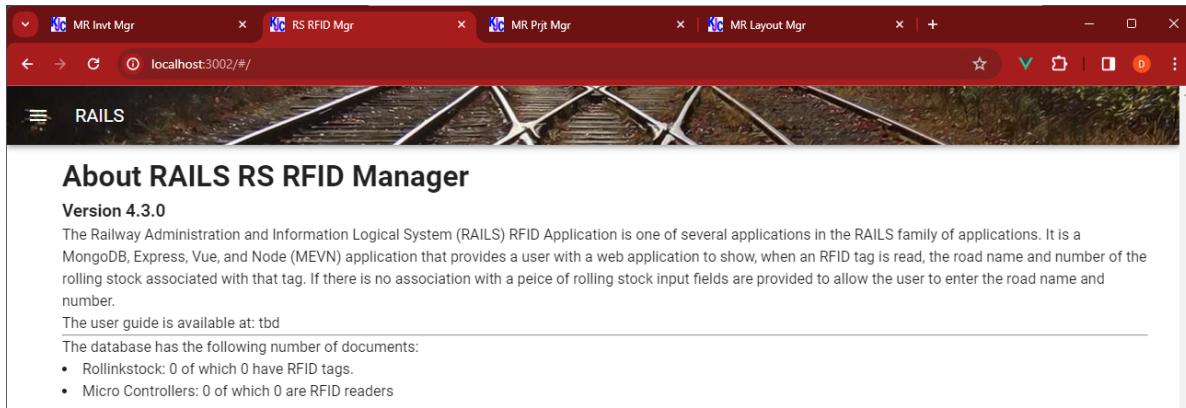


Figure B.8: RAILS RSRM Running

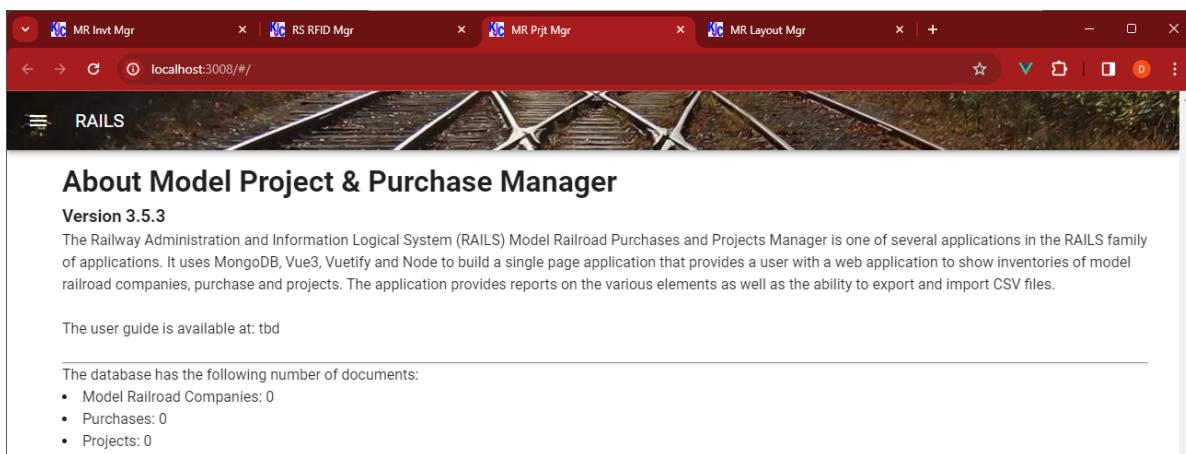


Figure B.9: RAILS MPPM Running

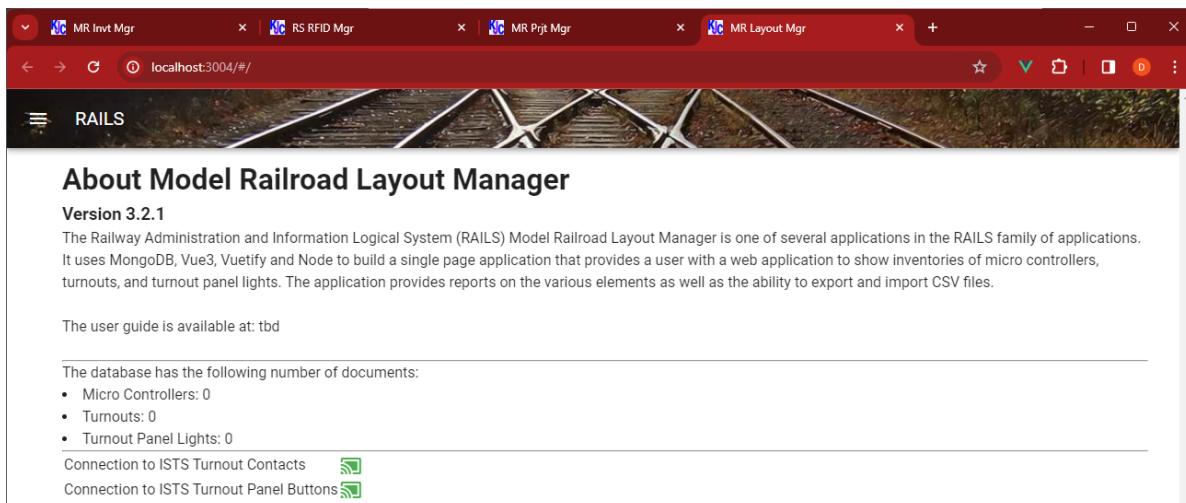


Figure B.10: RAILS MRLM Running

The following screenshot shows the stopping and removal of the RAILS Docker containers in a Windows environment.



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the command "C:\Users\Dave>docker-compose down" being run. The output lists 17 containers, each with a status of "Removed" and a duration. The containers are listed in two columns:

Container	Duration
myMpm	1.5s
myMrIm	1.6s
myMrUm	1.2s
myRsrm	1.4s
myIpts	11.1s
myIsts	12.0s
myIpIs	11.8s
myIsbs	12.3s
myIsms	12.9s
myIsrs	12.6s
myPpds	12.3s
myMrFm	12.5s
myRlds	12.1s
myRld	10.5s
myMqttBrkr	0.6s
myMongo	0.7s
Network dave_default	0.2s

Figure B.11: Windows Docker-Compose Down

## B.2 Docker in a Linux Environment

The following screenshot shows the intial Docker commands used to create the Docker environment in a Linux environment. The yellow highlighted text represents the edits of the mosquitto configuration file using the vi editor.

```

root@kingston:/SoftwareDev/RAILS/Docker Based# docker volume create --name mosquitto
mosquitto
root@kingston:/SoftwareDev/RAILS/Docker Based# docker run -it --name myMqttBrkr -p 1883:1883 -p
9001:9001 --rm -v mosquitto:/mosquitto -d eclipse-mosquitto
919d31e8f767810ac4dfd0ce2152d8c7f1d246434d735453fc12771538461ddc
root@kingston:/SoftwareDev/RAILS/Docker Based# docker stop myMqttBrkr
myMqttBrkr
root@kingston:/SoftwareDev/RAILS/Docker Based# docker inspect mosquitto
[
  {
    "CreatedAt": "2024-03-30T09:37:18-06:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/mosquitto/_data",
    "Name": "mosq",
    "Options": null,
    "Scope": "local"
  }
]
root@kingston:/SoftwareDev/RAILS/Docker Based# cd /var/lib/docker/volumes/mos/_data/config
root@kingston:/var/lib/docker/volumes/mos/_data/config# ls
mosquitto.conf
root@kingston:/var/lib/docker/volumes/mos/_data/config# vi mosquitto.conf
# =====
# Listeners
# =====
#
# On systems that support Unix Domain Sockets, it is also possible
# to create a # Unix socket rather than opening a TCP socket. In
# this case, the port number should be set to 0 and a unix socket
# path must be provided, e.g.
# listener 0 /tmp/mosquitto.sock
#
# listener port-number [ip address/host name/unix socket path]
listener 1883

# Set to `ipv4` to force the listener to only use IPv4, or set to `ipv6` to
# force the listener to only use IPv6. If you want support for both IPv4 and
# IPv6, then do not use the socket_domain option.
#
socket_domain ipv4

# =====
# Security
# =====
# Boolean value that determines whether clients that connect
# without providing a username are allowed to connect. If set to
# false then a password file should be created (see the
# password_file option) to control authenticated client access.
#
# Defaults to false, unless there are no listeners defined in the configuration
# file, in which case it is set to true, but connections are only allowed from
# the local machine.
allow_anonymous true

root@kingston:/SoftwareDev/RAILS/Docker Based# docker volume create --name myRailsDb
myRailsDb
root@kingston:/SoftwareDev/RAILS/Docker Based# docker volume create --name myRailsImages
myRailsImages
root@kingston:/SoftwareDev/RAILS/Docker Based# docker network create myRailsNet
cd6b2f49e1024dab42a7f28e85e00539f920fdfcfa8637fa8d6183bc42dc9e4ac
root@kingston:/SoftwareDev/RAILS/Docker Based#

```

Figure B.12: Initial Linux Docker Commands

The following screenshots show the Docker command used to pull images and run the RAILS Docker containers in a Linux environment. The pink elipses represent the multiple layers of the images being pulled from the Docker Hub.

```
root@kingston:/home/dbristow/ModelRailroad/SoftwareDev/RAILS/Docker Based# docker-compose up -d
Creating network "dockerbased_default" with the default driver
Pulling myMongo (mongo:)...  

latest: Pulling from library/mongo  

bcccd10f490ab: Pull complete  

...  

Digest: sha256:0e145625e78b94224d16222ff2609c4621ff6e2c390300e4e6bf698305596792  

Status: Downloaded newer image for mongo:latest  

Pulling myPpds (dbristow/ppds:)...  

latest: Pulling from dbristow/ppds  

4abcf2066143: Pull complete  

...  

Digest: sha256:7c699148e723b9357c3b40f8b549d8bd4ef563abde95fc30f973de0152bab7e6  

Status: Downloaded newer image for dbristow/ppds:latest  

Pulling myMppm (dbristow/mppm:)...  

latest: Pulling from dbristow/mppm  

4abcf2066143: Pull complete  

...  

Digest: sha256:f9066c1a7648df2a06a905afc0ced88a3a5ce77aa6250d1f1e98776f32541ff1  

Status: Downloaded newer image for dbristow/mppm:latest  

Pulling myRids (dbristow/rids:)...  

latest: Pulling from dbristow/rids  

4abcf2066143: Pull complete  

1c6d0e08288a: Pull complete  

...  

Digest: sha256:2aae5205370b400ec6c1f4b5199c25a1ab874a19a8ddc3bfc204f1ac09a87c84  

Status: Downloaded newer image for dbristow/rids:latest  

Pulling myMrfm (dbristow/mrfm:)...  

latest: Pulling from dbristow/mrfm  

4abcf2066143: Pull complete  

...  

Digest: sha256:36c5ac34ca1413f6c536eed430c14039e6cd883d46d56beba95415b87a821058  

Status: Downloaded newer image for dbristow/mrfm:latest  

Pulling myMrIm (dbristow/mrim:)...  

latest: Pulling from dbristow/mrim  

4abcf2066143: Pull complete  

...  

Digest: sha256:a35873cd333901742cc4049bef09d18d94863e16ecdfa28a1d1936af099f2cfb  

Status: Downloaded newer image for dbristow/mrim:latest  

Pulling myMqttBrkr (eclipse-mosquitto:)...  

latest: Pulling from library/eclipse-mosquitto  

619be1103602: Pull complete  

...  

Digest: sha256:bf5fb92712be8660ef6a204bf7a966c8f81f37d2b91a91432b9faaebf49c49d3  

Status: Downloaded newer image for eclipse-mosquitto:latest  

Pulling myRlds (dbristow/rlds:)...  

latest: Pulling from dbristow/rlds  

4abcf2066143: Pull complete  

...  

Digest: sha256:754f7e013595f14ff07adeaf9c46840697c58b76174166c0048e55026cab17eb  

Status: Downloaded newer image for dbristow/rlds:latest  

Pulling myIsrs (dbristow/isrs:)...  

latest: Pulling from dbristow/isrs  

4abcf2066143: Pull complete  

...  

Digest: sha256:1ab23967f11f839dfdc6c6b5987a1adb3157669d3833d93a561c5dbc9e85a871
```

Figure B.13: Linux Docker-Compose Up

```

Status: Downloaded newer image for dbristow/isrs:latest
Pulling myIsms (dbristow/isms)... 
latest: Pulling from dbristow/isms
4abcf2066143: Pull complete
...
Digest: sha256:63fba8fa14a261fc57b1a0ea61621f3ebcb2e10be52a7345245e3571a64a9689
Status: Downloaded newer image for dbristow/isms:latest
Pulling myRsm (dbristow/rsrm)... 
latest: Pulling from dbristow/rsrm
4abcf2066143: Pull complete
...
Digest: sha256:a2d72d9e1531cfe6e64e0191916c98abb30bd343e18fe534a6905d37db6facb5
Status: Downloaded newer image for dbristow/rsrm:latest
Pulling myIsts (dbristow/ists)... 
latest: Pulling from dbristow/ists
4abcf2066143: Pull complete
...
Digest: sha256:1f0e8ed6ba03d4df4e64f16070efd750d4bd130cf5168eea7a685a023344f93
Status: Downloaded newer image for dbristow/ists:latest
Pulling myIsbs (dbristow/isbs)... 
latest: Pulling from dbristow/isbs
4abcf2066143: Pull complete
...
Digest: sha256:fa35ee864bce4624ef2aea0d3aeae7d7772e6fad8166bacdc304a7c66a
Status: Downloaded newer image for dbristow/isbs:latest
Pulling myIpts (dbristow/ipts)... 
latest: Pulling from dbristow/ipts
4abcf2066143: Pull complete
...
Digest: sha256:7686f27a3b6165c0bafef1425e2f80b2a735615959014e8d45419d0dc34e4148
Status: Downloaded newer image for dbristow/ipts:latest
Pulling myIpls (dbristow/ipls)... 
latest: Pulling from dbristow/ipls
4abcf2066143: Pull complete
...
Digest: sha256:aafe3e13bbdcfef9ce382b2c079513637d057890f82b28adb1f0c2cb3ce47564
Status: Downloaded newer image for dbristow/ipls:latest
Pulling myMrM (dbristow/mrlm)... 
latest: Pulling from dbristow/mrlm
4abcf2066143: Pull complete
...
Digest: sha256:1b3d821ad904e68906f0af9d0614e9158beb14bd43e7e9a5b065a9637384b65b
Status: Downloaded newer image for dbristow/mrlm:latest
Creating myMqttBrkr ... done
Creating myMongo ... done
Creating myMrfm ... done
Creating myIpls ... done
Creating myIpts ... done
Creating myIsrs ... done
Creating myIsts ... done
Creating myIsbs ... done
Creating myPpds ... done
Creating myRlds ... done
Creating myRids ... done
Creating myMrM ... done
Creating myMpPm ... done
Creating myIsms ... done
Creating myMrM ... done
Creating myRsrm ... done
root@kingston:/SoftwareDev/RAILS/Docker Based#

```

Figure B.14: Linux Docker-Compose Up Continued

The following screenshot shows a RAILS SPAs running in a web browser (Chrome) in a Linux environment. The remaining SPAs can be accessed in the same manner.

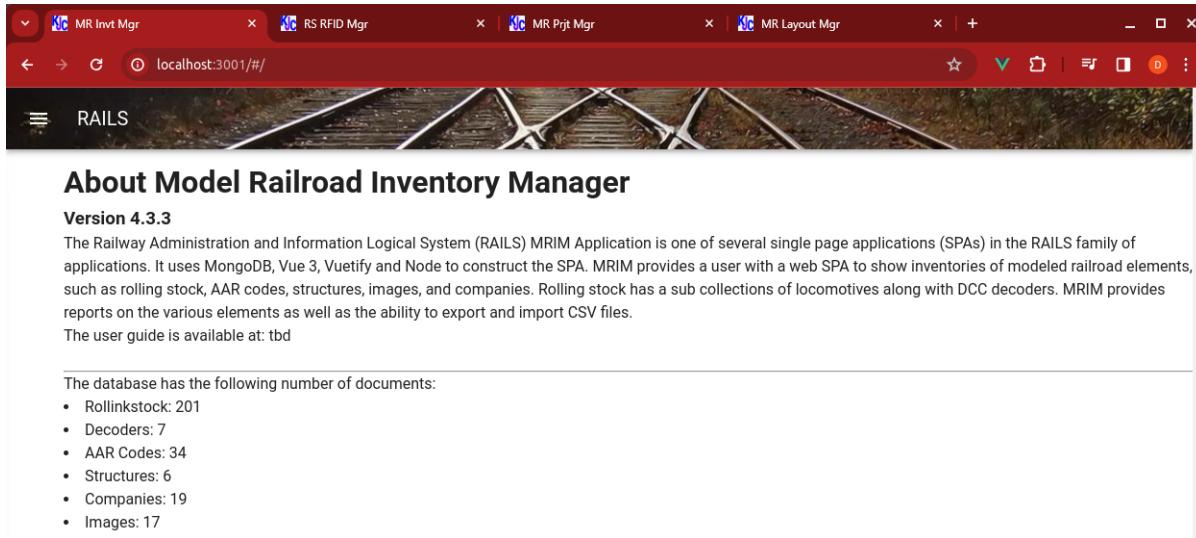


Figure B.15: RAILS MRIM Running

The following screenshot shows the stopping and removal of the RAILS Docker containers in a Linux environment.

```
root@kingston:/SoftwareDev/RAILS/Docker Based# docker-compose down
Stopping myRsrn    ... done
Stopping myIsms    ... done
Stopping myMrlm    ... done
Stopping myMppm    ... done
Stopping myMrim    ... done
Stopping myRids    ... done
Stopping myRlds    ... done
Stopping myPpds    ... done
Stopping myIsbs    ... done
Stopping myIsrs    ... done
Stopping myIsts    ... done
Stopping myIppts   ... done
Stopping myIppls   ... done
Stopping myMrfm    ... done
Stopping myMqttBrkr ... done
Stopping myMongo   ... done
Removing myRsrn   ... done
Removing myIsms   ... done
Removing myMrlm   ... done
Removing myMppm   ... done
Removing myMrim   ... done
Removing myRids   ... done
Removing myRlds   ... done
Removing myPpds   ... done
Removing myIsbs   ... done
Removing myIsrs   ... done
Removing myIsts   ... done
Removing myIppts  ... done
Removing myIppls  ... done
Removing myMrfm   ... done
Removing myMqttBrkr ... done
Removing myMongo   ... done
Network myRailsNet is external, skipping
Removing network dockerbased_default
root@kingston:/SoftwareDev/RAILS/Docker Based# |
```

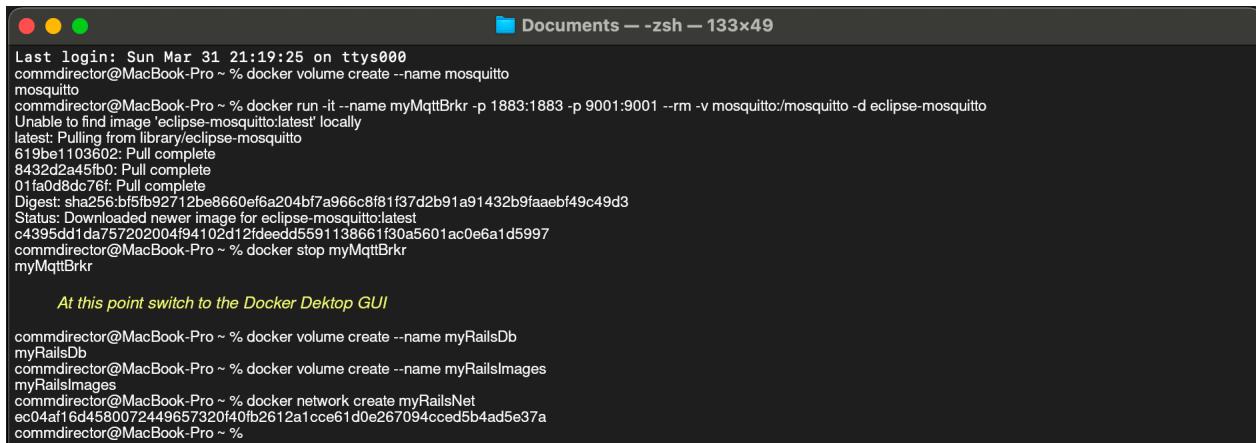
Figure B.16: Linux Docker-Compose Down

### B.3 Docker in a MacOS Environment

Docker itself doesn't run directly on MacOS 10.14 (Mojave). This is because Docker relies on a Linux kernel for containerization, and MacOS uses a different kernel. Similar to Windows there is a Docker Desktop software package that provides a complete Docker experience on MacOS. It includes:

- Docker Desktop installs and runs a lightweight Linux VM in the background. This VM typically uses LinuxKit, a minimal Linux distribution optimized for running Docker.
- The Docker engine runs within the Linux VM. This engine is responsible for managing Docker images, containers, and networks.
- Docker Desktop provides a CLI for interacting with the Docker engine running in the VM. You can use familiar Docker commands to build, run, and manage containers.
- Docker Desktop offers an optional GUI that allows you to manage containers visually.

The following screenshot shows the initial Docker commands used to create the Docker environment in a MacOS environment. Like Windows it is simpler to use Docker Desktop GUI to locate and edit the mosquito configuration file.



```
Last login: Sun Mar 31 21:19:25 on ttys000
commdirector@MacBook-Pro ~ % docker volume create --name mosquito
mosquito
commdirector@MacBook-Pro ~ % docker run -it --name myMqtBrkr -p 1883:1883 -p 9001:9001 --rm -v mosquito:/mosquitto -d eclipse-mosquitto
Unable to find image 'eclipse-mosquitto:latest' locally
latest: Pulling from library/eclipse-mosquitto
619be1103602: Pull complete
8432d2a45fb0: Pull complete
01fa0ddc76f: Pull complete
Digest: sha256:b5fb92712be8660ef6a204bf7a966c8f81f37d2b91a91432b9faaebf49c49d3
Status: Downloaded newer image for eclipse-mosquitto:latest
c4395dd1da757202004f94102d12fdeedd5591138661f30a5601ac0e6a1d5997
commdirector@MacBook-Pro ~ % docker stop myMqtBrkr
myMqtBrkr

At this point switch to the Docker Desktop GUI

commdirector@MacBook-Pro ~ % docker volume create --name myRailsDb
myRailsDb
commdirector@MacBook-Pro ~ % docker volume create --name myRailsImages
myRailsImages
commdirector@MacBook-Pro ~ % docker network create myRailsNet
ec04af16d4580072449657320f40fb2612a1cce61d0e267094cced5b4ad5e37a
commdirector@MacBook-Pro ~ %
```

Figure B.17: Initial MacOS Docker Commands

The following screenshots show the Docker command used to pull images and run the RAILS Docker containers in a MacOS environment.

```
Last login: Sun Mar 31 21:19:25 on ttys000
[commdirector@MacBook-Pro Documents % docker-compose up -d
[+] Running 77/15
✓ myMrim 1 layers [ ]    0B/0B    Pulled          15.8s
✓ myMrlm 9 layers [██████]  0B/0B    Pulled          11.9s
✓ myRids 4 layers [███]   0B/0B    Pulled          12.6s
✓ myIsbs 4 layers [███]   0B/0B    Pulled          21.6s
✓ myPpds 5 layers [███]   0B/0B    Pulled          13.4s
✓ myMrfm 5 layers [███]   0B/0B    Pulled          18.6s
✓ myIsms 4 layers [███]   0B/0B    Pulled          24.5s
✓ myIpts 4 layers [███]   0B/0B    Pulled          15.6s
✓ myMpmp 1 layers [ ]    0B/0B    Pulled          12.1s
✓ myIpds 4 layers [███]   0B/0B    Pulled          24.4s
✓ myRsrm 1 layers [ ]    0B/0B    Pulled          12.0s
✓ myIsrs 4 layers [███]   0B/0B    Pulled          18.8s
✓ myRlds 4 layers [███]   0B/0B    Pulled          13.4s
✓ myMongo 8 layers [██████] 0B/0B    Pulled          49.5s
✓ myLists 4 layers [███]   0B/0B    Pulled          24.9s
[+] Running 16/17
: Network documents_default Created          4.8s
✓ Container myMqttBrkr Started           1.4s
✓ Container myMrfm Started           1.3s
✓ Container myMongo Started           1.4s
✓ Container myRids Started           3.0s
✓ Container myPpds Started           3.2s
✓ Container myRlds Started           1.9s
✓ Container myIsts Started           2.3s
✓ Container myIpds Started           2.4s
✓ Container myIsrs Started           2.2s
✓ Container myIpts Started           2.2s
✓ Container myIsbs Started           1.8s
✓ Container myMrim Started           4.3s
✓ Container myIsms Started           2.8s
✓ Container myMpmp Started           4.3s
✓ Container myMrlm Started           3.6s
✓ Container myRsrm Started           4.1s
commdirector@MacBook-Pro Documents %
```

Figure B.18: MacOS Docker-Compose Up

The following screenshot shows a RAILS SPAs running in a web browser (Safari) in a MacOS environment. The remaining SPAs can be accessed in the same manner.

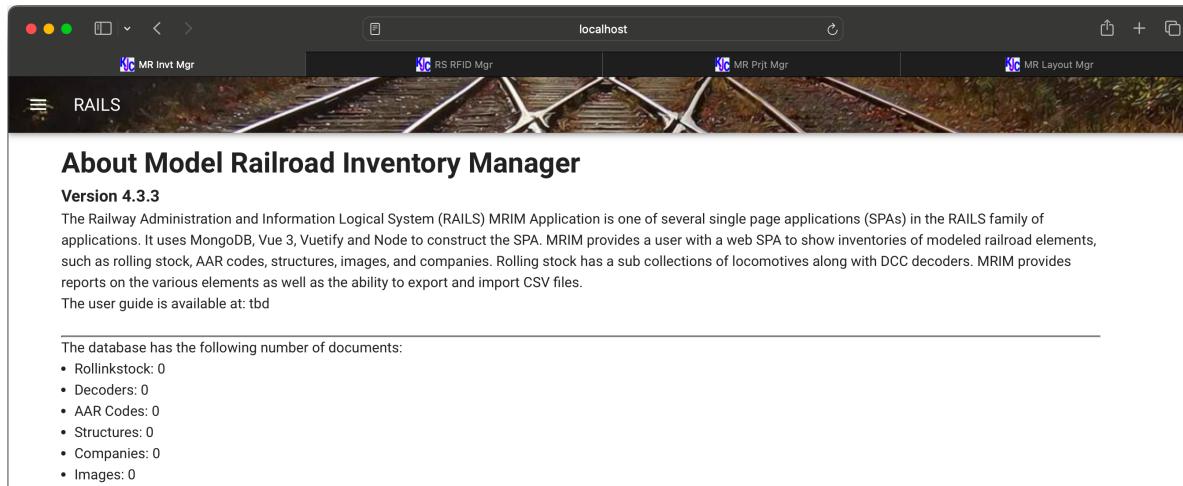
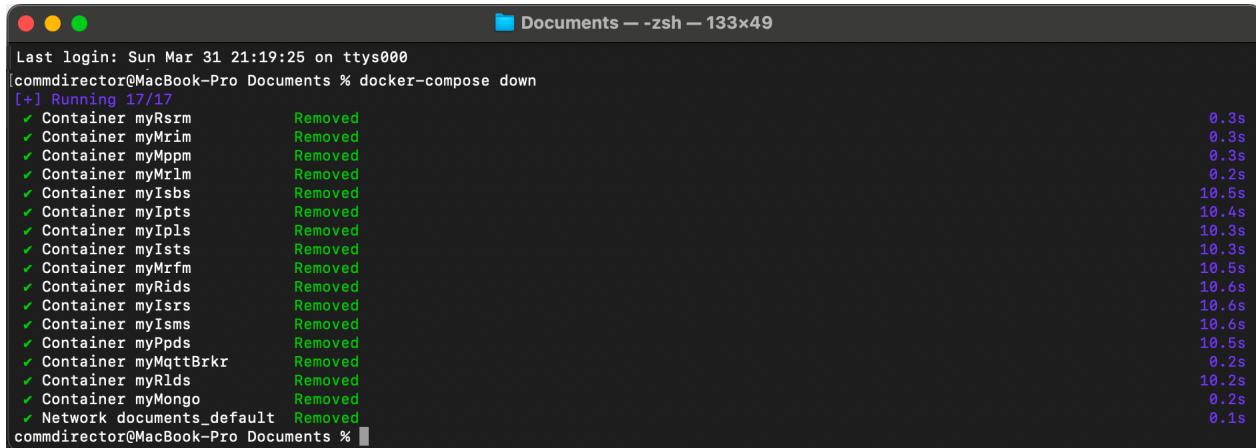


Figure B.19: RAILS MRIM Running

The following screenshot shows the stopping and removal of the RAILS Docker containers in a Linux environment.



```
Last login: Sun Mar 31 21:19:25 on ttys000
[commdirector@MacBook-Pro Documents % docker-compose down
[+] Running 17/17
✓ Container myRsrm      Removed          0.3s
✓ Container myMrim      Removed          0.3s
✓ Container myMppm      Removed          0.3s
✓ Container myMrIm      Removed          0.2s
✓ Container myIsbs      Removed          10.5s
✓ Container myIpTs      Removed          10.4s
✓ Container myIpLs      Removed          10.3s
✓ Container myIsts      Removed          10.3s
✓ Container myMrfm      Removed          10.5s
✓ Container myRids      Removed          10.6s
✓ Container myIsrs      Removed          10.6s
✓ Container myIsms      Removed          10.6s
✓ Container myPpds      Removed          10.5s
✓ Container myMqttBrkr  Removed          0.2s
✓ Container myRlds      Removed          10.2s
✓ Container myMongo     Removed          0.2s
✓ Network documents_default Removed          0.1s
commdirector@MacBook-Pro Documents % ]
```

Figure B.20: MacOS Docker-Compose Down

## Appendix C

# Docker Commands

The following Docker commands may be useful when working with Docker images and containers related to the RAILS SPAs.

- `docker ps` - List running containers.
- `docker ps -a` - List all containers including those that have been stopped.
- `docker images` - List images.
- `docker run -d --name <container-name> <image-name>` - Run a container from an image.
- `docker stop <container-name>` - Stop a running container.
- `docker rm <container-name>` - Remove a container.
- `docker rmi <image-name>` - Remove an image.
- `docker inspect <container-name> | <volume-name>` - Display detailed information about a container or volume.
- `docker-compose up -d` - Start the services defined in the Docker Compose file.
- `docker-compose down` - Stop the services defined in the Docker Compose file.

# Glossary

**Application Program Interface (API)** acts as a middleman between different software applications.

It defines a set of rules and specifications that allow applications to talk to each other and exchange data or functionality. 5

**command-line interface (CLI)** is a text-based way to interact with a computer program by typing commands and seeing text-based responses. 5, 6, 17, 28

**Data Services (DS)** is a class of RAILS components that provides an API permitting access to the MongoDB that stores the documents used by RAILS. 3, 4

**Graphic User Interface (GUI)** is the visual way you interact with a computer program using elements like windows, icons, menus, and buttons, instead of typing commands. GUIs are the standard way most people interact with computers today. 3, 4, 8, 18, 28

**infrared (IR)** Infrared radiation is a type of electromagnetic radiation that lies just beyond the red end of the visible light spectrum. It's invisible to the human eye, but we can feel it as heat. 3

**Internet of Things (IoT)** The Internet of Things refers to a vast network of physical devices embedded with sensors, software, and other technologies that allows them to connect and exchange data with other devices and systems over the internet or other communication networks. 3

**IoT Publisher Turnout Panel Light Services (IPLS)** . 3

**IoT Publisher Turnout Services (IPTS)** . 3

**IoT Subscriber Location Services (ISLS)** . 3

**IoT Subscriber Micro-controller Services (ISMS)** . 3

**IoT Subscriber RFID Services (ISRS)** . 3

**IoT Subscriber Turnout Panel Button Services (ISBS)** . 3

**IoT Subscriber Turnout Services (ISTS)** . 3

**Message Queuing Telemetry Transport (MQTT)** MQTT is a lightweight messaging protocol designed for machine-to-machine (M2M) communication in resource-constrained environments, like those found in the Internet of Things (IoT). 3, 8

**Model Projects and Purchase Manager (MPPM)** . 4, 10

**Model Railroad File Manager (MRFM)** . 4

**Model Railroad Inventory Manager (MRIM)** . 4, 10

**Model Railroad Layout Manager (MRLM)** . 3, 4, 10

**Plans and Purchases Data Services (PPDS)** . 4

**Radio Frequency Identification (RFID)** is technology that uses radio waves to wirelessly identify and track objects. 3, 4

**Railroad Inventory Data Services (RIDS)** . 4

**Railroad Layout Data Services (RLDS)** . 4

**Railway Administration and Information Logical System (RAILS)** . 2, 3, 4, 7, 8, 9, 10, 17, 19, 21, 23, 25, 26, 27, 28, 29, 31

**Representational State Transfer (REST)** is a set of architectural principles for designing web services. It's not a protocol itself (like HTTP), but rather a guideline for creating web services. 4

**Rollingstock RFID Manager (RSRM)** . 3, 4, 10

**Single Page Application (SPA)** is a web application that loads a single HTML page in the user's browser and dynamically updates the content based on user interaction, without reloading the entire page. This creates a more fluid and responsive user experience, similar to what you might expect from a native mobile app. 3, 4, 8, 9, 10, 21, 26, 29, 31

**virtual machine (VM)** is a software program that acts like a physical computer. It creates a virtualized environment that can run its own operating system (OS) and applications, just like a physical computer would. 5, 28

**Windows Subsystem for Linux 2 (WSL 2)** is a feature introduced by Microsoft that allows you to run Linux distributions directly on Windows 10 and 11, alongside your existing Windows environment. 17, 18