



Docker Implementation

Railway Administration and Information Logical System

RAILS for Model Railroads

David Bristow
Version 2.2.0
February 5, 2026

Contents

1	Introduction	2
1.1	Microservices Design Components	2
1.1.1	IoT Components	3
1.1.2	DS Components	3
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	6
2	Setup and Run	8
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	10
2.6	Stop and Remove the RAILS Docker Containers	10
A	YAML File	11
B	NGINX Configuration	16
C	Example Setup and Run Commands	21
C.1	Docker in a Windows Environment	21
C.2	Docker in a Linux Environment	27
C.3	Docker in a MacOS Environment	29
D	Docker Commands	34
Acronyms		35

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.

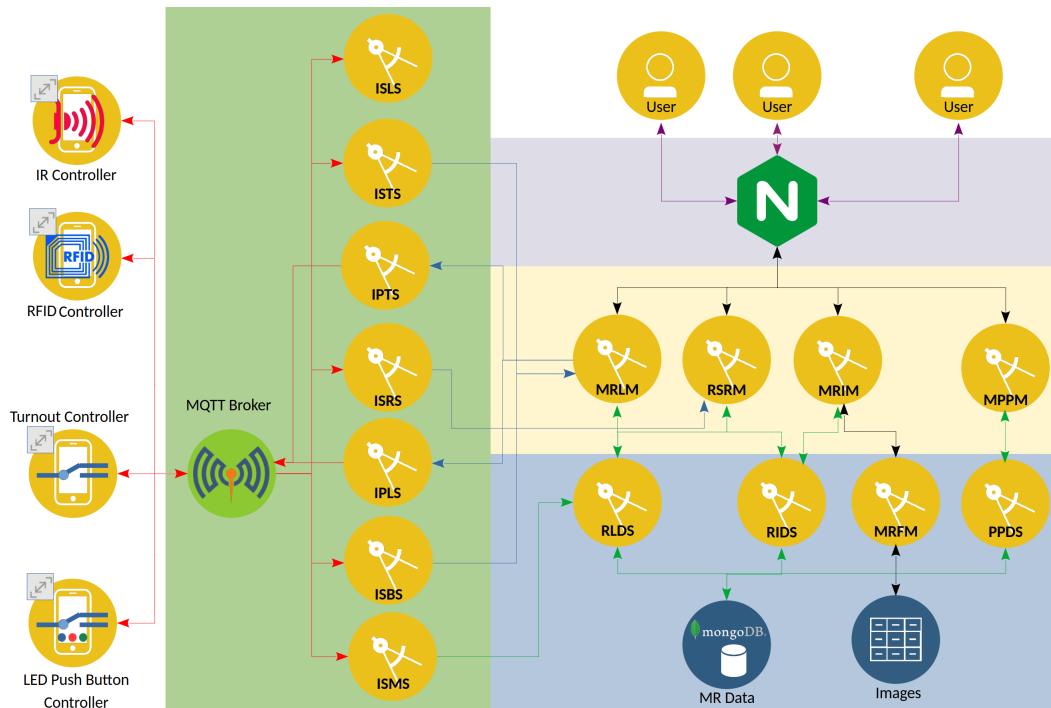


Figure 1.1: Microservices Component Architecture

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.
- a reverse proxy, which is highlighted with the mauve 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 yet another markup language ([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.yaml 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.yaml 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 conatinerized 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
----- SPAs -----				
dbristow/mppm	Model Projects and Purchase Manager	3008	4.0.9	2026-02-02
dbristow/mrim	Model Railroad Inventory Manager	3001	5.3.5	2026-02-02
dbristow/mrlm	Model Railroad Layout Manager	3004	4.0.9	2026-02-02
dbristow/rsrm	Rollingstock RFID Manager	3002	6.0.9	2026-02-02
----- Data Services -----				
dbristow/mrfm	Model Railroad File Manager	3030	2.6.0	2026-02-05
dbristow/ppsds	Plans and Purchases Data Services	3007	3.0.7	2026-02-02
dbristow/rids	Railroad Inventory Data Services	3000	3.0.7	2026-02-02
dbristow/rlds	Railroad Layout Data Services	3006	3.0.9	2026-02-02
----- IoT Services -----				
dbristow/ipls	IoT Publisher Turnout Panel Light Services	3013	1.1.19	2026-02-02
dbristow/ipts	IoT Publisher Turnout Services	3011	2.1.19	2026-02-02
dbristow/isbs	IoT Subscriber Turnout Panel Button Services	3012	2.0.6	2026-02-02
dbristow/isms	IoT Subscriber Micro-controller Services		3.0.18	2026-02-02
dbristow/isrs	IoT Subscriber RFID Services	3005	2.0.6	2026-02-02
dbristow/ists	IoT Subscriber Turnout Services	3010	2.0.6	2026-02-02

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 instructions¹ 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 folder/directory "RAILS" and open a terminal from that folder/directory (all docker commands should be run in that terminal).
2. Create a Docker volume for the Mosquitto broker.

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

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

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

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

4. Stop the broker container.

```
docker stop myMqttBrkr
```

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

```
docker inspect mqtt
```

6. 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](#). Additionally create the “nginx.conf” file from Appendix B or retrieve it from the RAILS GitHub repository. Be sure that both files are put into the “RAILS” folder/directory. To fetch the images, create the containers, and start the containers running execute the following command in the terminal:

```
docker compose up -d
```

This command will pull the Docker images from Docker Hub and run the containers in detached mode. The RAILS SPAs will be running in the background, and the terminal will return to the command prompt. The RAILS SPAs will be running on the host machine. 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 C 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, 2020-2026
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     volumes:
25       # Mount your custom nginx.conf into the container
26       - ./nginx.conf:/etc/nginx/nginx.conf:ro
27     networks:
28       - myRailsNet # Connect Nginx to your internal network
29     depends_on:
30       # Nginx should start after all services it proxies to
31       - myMongo
32       - myPpds
33       - myMppm
34       - myRids
35       - myMrfm
36       - myMrim
37       - myMqttBrkr
```

```

38      - myRlds
39      - myIsrs
40      - myIsms
41      - myRsrm
42      - myIsts
43      - myIsbs
44      - myIpst
45      - myIpls
46      - myMrlm
47
48 myMongo:
49   image: 'mongo'
50   container_name: myMongo
51   networks:
52     - myRailsNet
53   volumes:
54     - myRailsDb:/data/db
55
56 myPpds:
57   image: 'dbristow/ppds'
58   container_name: myPpds
59   networks:
60     - myRailsNet
61   depends_on:
62     - myMongo
63
64 myMppm:
65   image: 'dbristow/mppm'
66   container_name: myMppm
67   networks:
68     - myRailsNet # Add to network so Nginx can find it
69   depends_on:
70     - myPpds
71
72 myRids:
73   image: 'dbristow/rids'
74   container_name: myRids
75   networks:
76     - myRailsNet
77   depends_on:
78     - myMongo
79
80 myMrfm:
81   image: 'dbristow/mrfm'
82   # Environment variables for myMrfm's *backend* if it has one.
83   # Frontend VITE_ variables are handled at image build time.
84   environment:
85     # These might be for internal communication *within* myMrfm's backend,
86     # or if myMrfm's backend needs to call other services.
87     # If myMrfm is purely a static file server, these might not be needed.
88     - MYMRIM_URI1=http://myMrim:8080 # Use service name for internal Docker
       network

```

```
89      - MYMRIM_URI2=http://myMrim:8080 # Use service name for internal Docker
90          network
91      - ALLOWED_CONTAINER_ORIGIN=http://myMrim
92      container_name: myMrfm
93      volumes:
94          - myRailsImages:/usr/djb/src/uploads
95      networks:
96          - myRailsNet # Add to network so Nginx can find it
97
98 myMrim:
99     image: 'dbristow/mrim'
100    container_name: myMrim
101    networks:
102        - myRailsNet
103    depends_on:
104        - myRids
105        - myMrfm
106
107 myMqttBrkr:
108     image: 'eclipse-mosquitto'
109    container_name: myMqttBrkr
110    networks:
111        - myRailsNet
112    # Keep MQTT ports exposed if external clients need to connect directly
113    ports:
114        - '1883:1883'
115        - '9001:9001'
116    volumes:
117        - mosquitto:/mosquitto
118
119 myRlds:
120     image: 'dbristow/rlds'
121    container_name: myRlds
122    networks:
123        - myRailsNet
124    depends_on:
125        - myMongo
126
127 myIsrs:
128     image: 'dbristow/isrs'
129    container_name: myIsrs
130    networks:
131        - myRailsNet
132    depends_on:
133        - myMqttBrkr
134
135 myIsms:
136     image: 'dbristow/isms'
137    container_name: myIsms
138    networks:
139        - myRailsNet
140    depends_on:
141        - myMqttBrkr
```

```
141     - myRlds
142
143 myRsrm:
144   image: 'dbristow/rsrm'
145   container_name: myRsrm
146   networks:
147     - myRailsNet
148   depends_on:
149     - myRids
150     - myRlds
151     - myIsrs
152     - myIsms
153
154 myIsts:
155   image: 'dbristow/ists'
156   container_name: myIsts
157   networks:
158     - myRailsNet
159   depends_on:
160     - myMqttBrkr
161
162 myIsbs:
163   image: 'dbristow/isbs'
164   container_name: myIsbs
165   networks:
166     - myRailsNet
167   depends_on:
168     - myMqttBrkr
169
170 myIpts:
171   image: 'dbristow/ipts'
172   container_name: myIpts
173   networks:
174     - myRailsNet
175   depends_on:
176     - myMqttBrkr
177
178 myIpls:
179   image: 'dbristow/ipls'
180   container_name: myIpls
181   networks:
182     - myRailsNet
183   depends_on:
184     - myMqttBrkr
185
186 myMrilm:
187   image: 'dbristow/mrilm'
188   container_name: myMrilm
189   networks:
190     - myRailsNet
191   depends_on:
192     - myRlds
193     - myIsts
```

```
194      - myIsbs
195      - myIpts
196      - myIpls
197
198 networks:
199   myRailsNet:
200     # It's good practice to define the network as internal for security,
201     # unless you explicitly need containers outside this compose file to join
202     # it.
203     # If `myRailsNet` is truly external (created manually with `docker network
204     # create`),
205     # then keep `external: true`. Otherwise, use `driver: bridge`.
206     # external: true # Keep if you've already created it with `docker network
207     # create myRailsNet`
208     # driver: bridge # Use this if you want Docker Compose to manage its
209     # creation
210
211 volumes:
212   myRailsDb:
213     external: true
214   myRailsImages:
215     external: true
216   mosquitto:
217     external: true
```

Listing A.1: docker compose file

Appendix B

NGINX Configuration

```
1 # nginx.conf (for the Nginx Reverse Proxy container)
2
3 worker_processes auto;
4
5 events {
6     worker_connections 1024;
7 }
8
9 http {
10     include mime.types;
11     default_type application/octet-stream;
12
13     sendfile on;
14     keepalive_timeout 65;
15
16     # Log format for better debugging
17     log_format combined_proxy '$remote_addr-$remote_user[$time_local]-
18                             "'.$request"$status$body_bytes_sent"
19                             "'.$http_referer"$http_user_agent"
20                             "'.$http_x_forwarded_for"$upstream_addr';
21
22     access_log /var/log/nginx/access.log combined_proxy;
23     error_log /var/log/nginx/error.log warn; # Set to info or debug for more
24         verbose logging
25
26     server {
27         listen 80;
28         server_name localhost; # Use your public domain name here in
29             production (e.g., yourmodelrailroad.com)
30
31         # --- Frontend Applications (SPAs) ---
32         # Each SPA will be served from its own base path, e.g., /mrim/, /mppm/
33         # The rewrite rule removes the path prefix before forwarding to the
34             SPA's internal server.
35
36         # myMrim Frontend
37         location /mrim/ {
38             rewrite ^/mrim/(.*)$ /$1 break; # Remove /mrim/ prefix
```

```
36         proxy_pass http://myMrim:8080; # myMrim service, internal port
37             8080
38         proxy_set_header Host $host;
39         proxy_set_header X-Real-IP $remote_addr;
40         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
41         proxy_set_header X-Forwarded-Proto $scheme;
42     }
43
44     # myMppm Frontend
45     location /mppm/ {
46         rewrite ^/mppm/(.*)$ /$1 break;
47         proxy_pass http://myMppm:8080; # myMppm service, internal port
48             8080
49         proxy_set_header Host $host;
50         proxy_set_header X-Real-IP $remote_addr;
51         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
52         proxy_set_header X-Forwarded-Proto $scheme;
53     }
54
55     # myMrilm Frontend
56     location /mrilm/ {
57         rewrite ^/mrilm/(.*)$ /$1 break;
58         proxy_pass http://myMrilm:8080; # myMrilm service, internal port
59             8080
60         proxy_set_header Host $host;
61         proxy_set_header X-Real-IP $remote_addr;
62         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
63         proxy_set_header X-Forwarded-Proto $scheme;
64     }
65
66     # myRsrm Frontend
67     location /rsrm/ {
68         rewrite ^/rsrm/(.*)$ /$1 break;
69         proxy_pass http://myRsrm:8080; # myRsrm service, internal port
70             8080
71         proxy_set_header Host $host;
72         proxy_set_header X-Real-IP $remote_addr;
73         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
74         proxy_set_header X-Forwarded-Proto $scheme;
75     }
76
77     # --- Backend API Services ---
78     # myRids API (e.g., accessible at http://yourdomain.com/api/rids/
79     # rslistall)
80     location /api/rids/ {
81         rewrite ^/api/rids/(.*)$ /$1 break; # Remove /api/rids/ prefix
82         proxy_pass http://myRids:3000; # myRids service, internal port
83             3000
84         proxy_set_header Host $host;
85         proxy_set_header X-Real-IP $remote_addr;
86         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
87         proxy_set_header X-Forwarded-Proto $scheme;
88     }
89 }
```

```

83
84      # myRlds API (e.g., accessible at http://yourdomain.com/api/rlds/
85      # some_endpoint)
86      location /api/rlds/ {
87          rewrite ^/api/rlds/(.*)$ /$1 break; # Remove /api/rlds/ prefix
88          proxy_pass http://myRlds:3006; # myRlds service, internal port
89          proxy_set_header Host $host;
90          proxy_set_header X-Real-IP $remote_addr;
91          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
92          proxy_set_header X-Forwarded-Proto $scheme;
93      }
94
95      # myPpds API (e.g., accessible at http://yourdomain.com/api/ppds/
96      # some_endpoint)
97      location /api/ppds/ {
98          rewrite ^/api/ppds/(.*)$ /$1 break; # Remove /api/ppds/ prefix
99          proxy_pass http://myPpds:3007; # myPpds service, internal port
100         proxy_set_header Host $host;
101         proxy_set_header X-Real-IP $remote_addr;
102         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
103         proxy_set_header X-Forwarded-Proto $scheme;
104     }
105
106     # myMrfm Image Server (e.g., accessible at http://yourdomain.com/
107     # images/mikado.png)
108     location /images/ {
109         rewrite ^/images/(.*)$ /$1 break; # Remove /images/ prefix
110         proxy_pass http://myMrfm:3030; # myMrfm service, internal port
111         proxy_set_header Host $host;
112         proxy_set_header X-Real-IP $remote_addr;
113         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
114         proxy_set_header X-Forwarded-Proto $scheme;
115
116     # --- MQTT-Interfacing API Services ---
117     # myIsrs API3005
118     location /api/isrs/ {
119         rewrite ^/api/isrs/(.*)$ /$1 break;
120         proxy_pass http://myIsrs:3005;
121
122         proxy_http_version 1.1;
123         proxy_set_header Upgrade $http_upgrade;
124         proxy_set_header Connection "upgrade";
125         proxy_set_header Host $host;
126         proxy_set_header X-Real-IP $remote_addr;
127         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
128         proxy_set_header X-Forwarded-Proto $scheme;
129     }

```

```
130      # myIpls API and Socket.IO internal port 3013
131      location /api/ipls/ {
132          rewrite ^/api/ipls/(.*)$ /$1 break;
133          proxy_pass http://myIpls:3013;
134
135          proxy_http_version 1.1;
136          proxy_set_header Upgrade $http_upgrade;
137          proxy_set_header Connection "upgrade";
138          proxy_set_header Host $host;
139          proxy_set_header X-Real-IP $remote_addr;
140          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
141          proxy_set_header X-Forwarded-Proto $scheme;
142      }
143
144      # myIpts API and Socket.IO internal port 3011
145      location /api/ipts/ {
146          rewrite ^/api/ipts/(.*)$ /$1 break;
147          proxy_pass http://myIpts:3011;
148
149          proxy_http_version 1.1;
150          proxy_set_header Upgrade $http_upgrade;
151          proxy_set_header Connection "upgrade";
152          proxy_set_header Host $host;
153          proxy_set_header X-Real-IP $remote_addr;
154          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
155          proxy_set_header X-Forwarded-Proto $scheme;
156      }
157
158      # myIsbs API and Socket.IO internal port 3012
159      location /api/isbs/ {
160          rewrite ^/api/isbs/(.*)$ /$1 break;
161          proxy_pass http://myIsbs:3012;
162
163          proxy_http_version 1.1;
164          proxy_set_header Upgrade $http_upgrade;
165          proxy_set_header Connection "upgrade";
166          proxy_set_header Host $host;
167          proxy_set_header X-Real-IP $remote_addr;
168          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
169          proxy_set_header X-Forwarded-Proto $scheme;
170      }
171
172      # myIsms API and Socket.IO internal port
173      location /api/isms/ {
174          rewrite ^/api/isms/(.*)$ /$1 break;
175          proxy_pass http://myIsms:8080; # Assuming internal port 8080 (
176              common for Node/Java)
176          proxy_set_header Host $host;
177          proxy_set_header X-Real-IP $remote_addr;
178          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
179          proxy_set_header X-Forwarded-Proto $scheme;
180      }
181
```

```
182     # myIsts API and Socket.IO internal port 3010
183     location /api/ists/ {
184         rewrite ^/api/ists/(.*)$ /$1 break;
185         proxy_pass http://myIsts:3010;
186
187         proxy_http_version 1.1;
188         proxy_set_header Upgrade $http_upgrade;
189         proxy_set_header Connection "upgrade";
190         proxy_set_header Host $host;
191         proxy_set_header X-Real-IP $remote_addr;
192         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
193         proxy_set_header X-Forwarded-Proto $scheme;
194     }
195
196     # --- Default/Root Fallback ---
197     # If you have a primary SPA that should load at the root (e.g.,
198     # yourmodelrailroad.com/),
199     # you would put its proxy_pass here. Otherwise, this serves a default
200     # Nginx page.
201     location / {
202         root /usr/share/nginx/html; # Default Nginx welcome page or static
203         content
204         index index.html index.htm;
205         try_files $uri $uri/ =404;
206     }
207
208     # Error pages (optional)
209     error_page 500 502 503 504 /50x.html;
210     location = /50x.html {
211         root /usr/share/nginx/html;
212     }
213 }
```

Listing B.1: nginx configuration file

Appendix C

Example Setup and Run Commands

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

C.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:

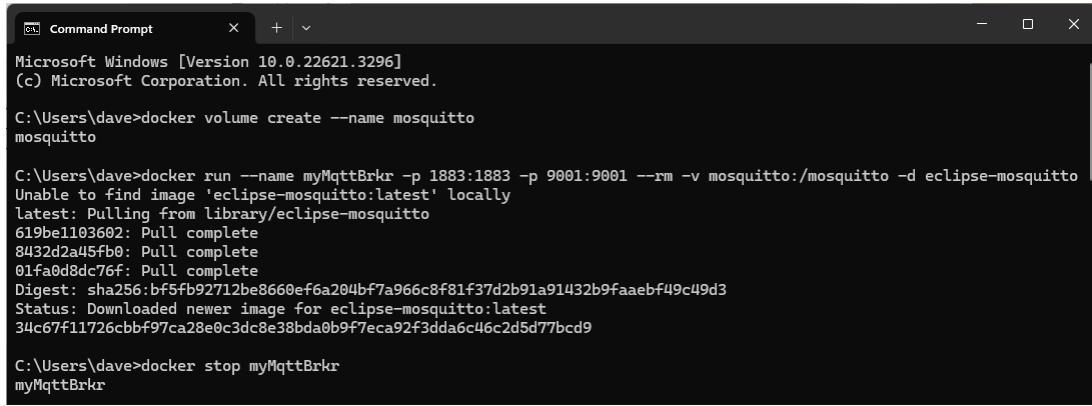
- Use [WSL 2 backend](#): This is the recommended approach for most users. It leverages WSL 2 to create a Linux environment for Docker to run containers.
- Use Hyper-V backend: This is an alternative that uses Hyper-V virtualization technology. However, [WSL 2](#) is generally considered more performant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

This setup is based on using [WSL 2](#) 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:bff5fb92712be8660ef6a204bf7a966c8f81f37d2b91a91432b9faebf49c49d3
Status: Downloaded newer image for eclipse-mosquitto:latest
34c67f11726cbbf97ca28e0c3dc8e38bda0b9f7eca92f3dda6c46c2d5d77bcd9

C:\Users\Dave>docker stop myMqttBrkr
myMqttBrkr
  
```

Figure C.1: Windows Docker mosquito Setup

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. Figure C.2 shows the volumes created in the Windows environment.

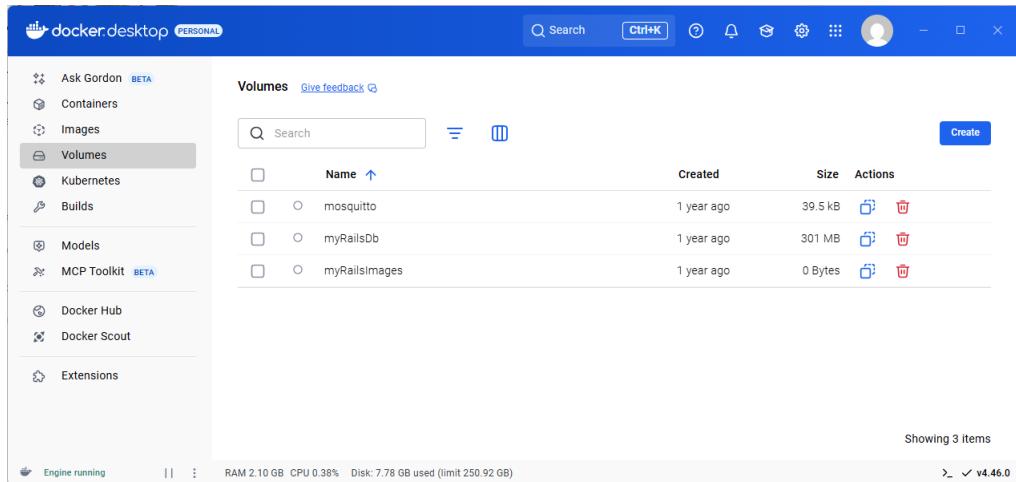


Figure C.2: Docker Desktop Volumes

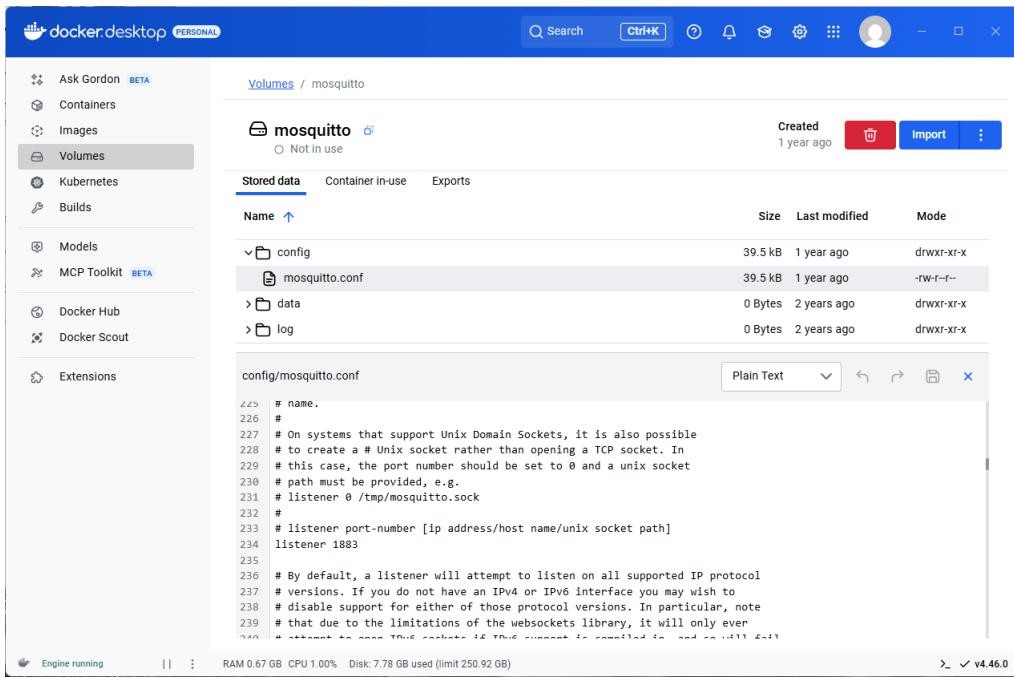


Figure C.3: Mosquitto Configuration File

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

```

PS C:\Users\Dave\Desktop\RAILS> docker compose up -d
[+] Running 87/87
  ✓ nginx Pulled
  ✓ myMppm Pulled
  ✓ myMrIm Pulled
  ✓ myMrFm Pulled
  ✓ myIsrs Pulled
  ✓ myMqttBrkr Pulled
  ✓ myRlds Pulled
  ✓ myMongo Pulled
  ✓ myIpts Pulled
  ✓ myRids Pulled
  ✓ myMrIm Pulled
  ✓ myRsrm Pulled
  ✓ myIsts Pulled
  ✓ myIsbs Pulled
  ✓ myPpds Pulled
  ✓ myIsms Pulled
  ✓ myIpds Pulled
  ✓ myIpls Pulled
  ✓ myMongo Pulled
  ✓ Container myMongo Started 33.3s
  ✓ Container myMqttBrkr Started 24.1s
  ✓ Container myMrFm Started 24.1s
  ✓ Container myPpds Started 43.5s
  ✓ Container myRlds Started 33.5s
  ✓ Container myIsbs Started 58.3s
  ✓ Container myIpds Started 29.4s
  ✓ Container myMongo Started 96.2s
  ✓ Container myIpls Started 39.6s
  ✓ Container myRids Started 29.5s
  ✓ Container myMrIm Started 24.4s
  ✓ Container myRsrm Started 24.3s
  ✓ Container myIsts Started 32.0s
  ✓ Container myIsrs Started 33.1s
  ✓ Container myPpds Started 31.2s
  ✓ Container myIsms Started 34.9s
  ✓ Container myIpds Started 39.5s
  ✓ Container myMongo Started 7.1s
  ✓ Container myMqttBrkr Started 7.2s
  ✓ Container myMrFm Started 7.2s
  ✓ Container myPpds Started 7.2s
  ✓ Container myRlds Started 6.5s
  ✓ Container myIsbs Started 7.6s
  ✓ Container myIpds Started 10.2s
  ✓ Container myMongo Started 10.6s
  ✓ Container myMrIm Started 11.3s
  ✓ Container myIsms Started 9.9s
  ✓ Container myMrIm Started 13.2s
  ✓ Container myRsrm Started 13.1s
  ✓ Container myIpls Started 14.6s

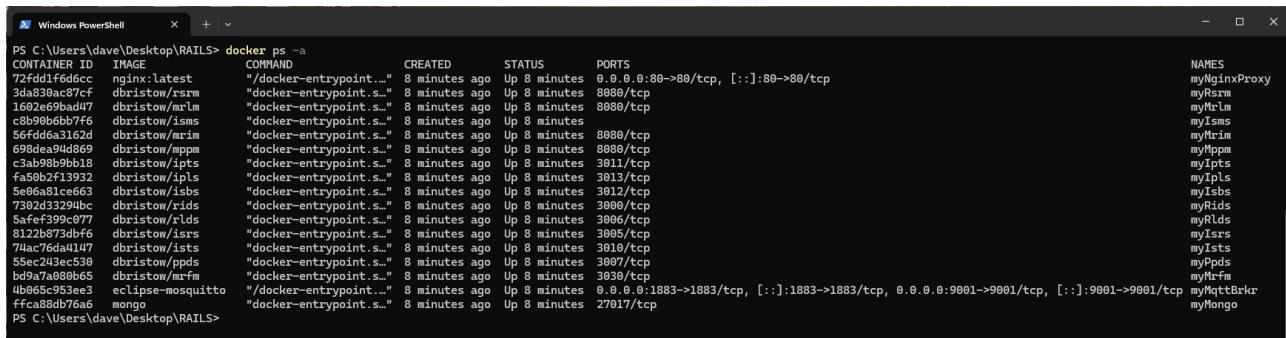
[+] Running 17/17
  ✓ Container myMongo Started 7.1s
  ✓ Container myMqttBrkr Started 7.2s
  ✓ Container myMrFm Started 7.2s
  ✓ Container myPpds Started 7.2s
  ✓ Container myRlds Started 6.5s
  ✓ Container myIsbs Started 10.2s
  ✓ Container myIpds Started 6.9s
  ✓ Container myMongo Started 6.9s
  ✓ Container myMqttBrkr Started 6.9s
  ✓ Container myMrIm Started 7.7s
  ✓ Container myPpds Started 7.7s
  ✓ Container myRlds Started 10.6s
  ✓ Container myIsbs Started 11.3s
  ✓ Container myIpds Started 10.6s
  ✓ Container myMongo Started 9.9s
  ✓ Container myMrIm Started 13.2s
  ✓ Container myRsrm Started 13.1s
  ✓ Container myIpls Started 14.6s

PS C:\Users\Dave\Desktop\RAILS>

```

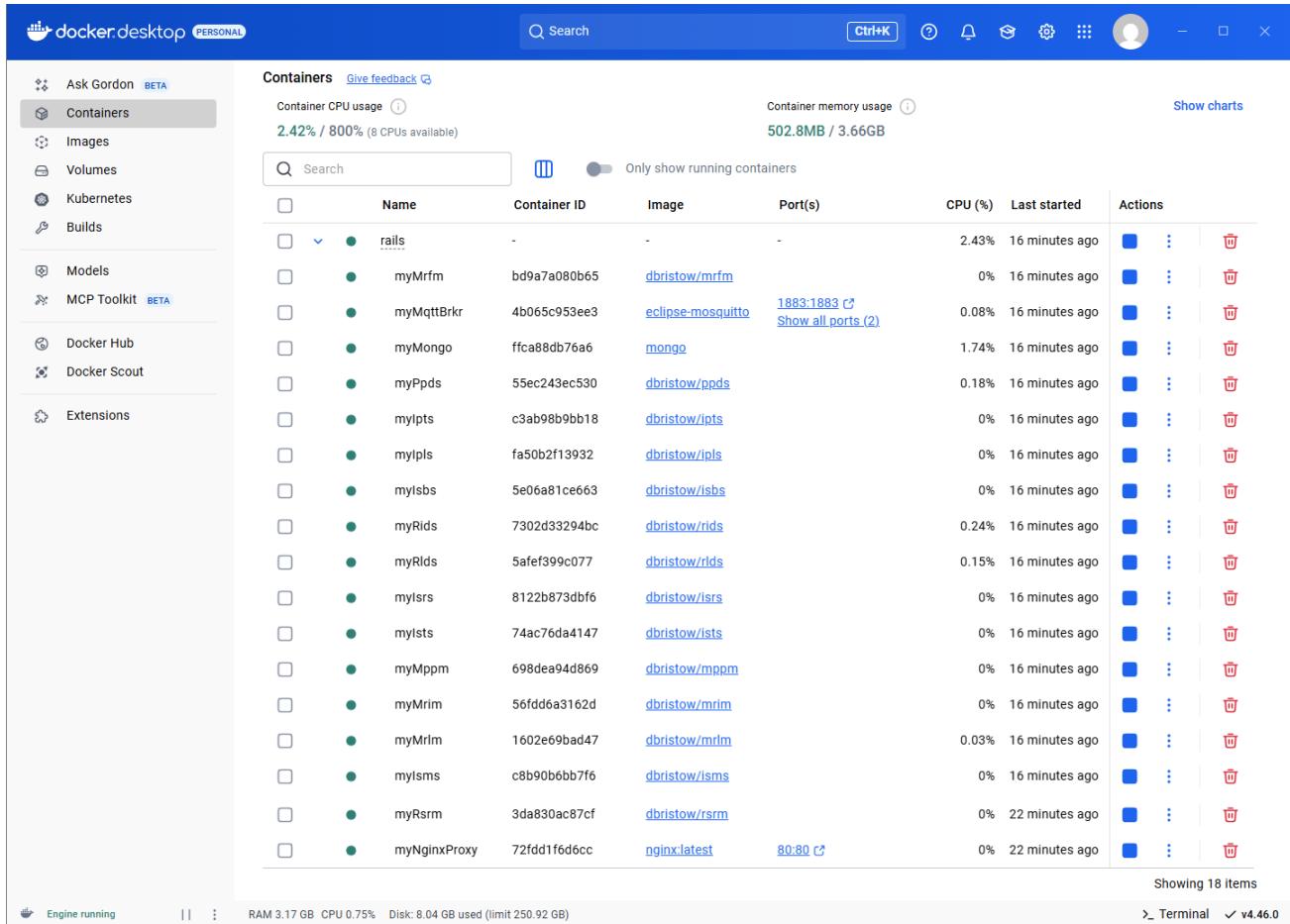
Figure C.4: Windows Docker Compose Up

The following screenshots confirm the Docker containers are running.



```
PS C:\Users\Dave\Desktop\RAILS> docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
72fd1f6d6cc nginx:latest "/docker-entrypoint...." 8 minutes ago Up 8 minutes 0.0.0.0:80->80/tcp, [::]:80->80/tcp myNginxProxy
3da830ac87cf dbristow/rsm "/docker-entrypoint...." 8 minutes ago Up 8 minutes 8080/tcp myRsrm
1602e69bad47 dbristow/mrlm "/docker-entrypoint...." 8 minutes ago Up 8 minutes 8080/tcp myMrLm
c8b90b6bb7f6 dbristow/isms "/docker-entrypoint...." 8 minutes ago Up 8 minutes 8080/tcp myIsms
56fd6a3162d dbristow/mrim "/docker-entrypoint...." 8 minutes ago Up 8 minutes 8080/tcp myMrIm
698dea94d869 dbristow/mppm "/docker-entrypoint...." 8 minutes ago Up 8 minutes 8080/tcp myMppm
c3ab98b9bb18 dbristow/ipts "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3011/tcp myIpts
fa50b2f13932 dbristow/ipls "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3013/tcp myIpls
5e06a81ce663 dbristow/isbs "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3012/tcp myIsbs
7302d33294bc dbristow/rlds "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3000/tcp myRlds
5afef399c077 dbristow/rls "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3006/tcp myRls
8122b873dbf6 dbristow/isrs "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3005/tcp myIsrs
74ac76da4147 dbristow/ists "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3010/tcp myIsts
55ec243ec530 dbristow/pdds "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3007/tcp myPdds
bd9a7a080b65 dbristow/mrfm "/docker-entrypoint...." 8 minutes ago Up 8 minutes 3030/tcp myMrfm
4b065c953ee3 eclipse-mosquitto "/docker-entrypoint...." 8 minutes ago Up 8 minutes 0.0.0.0:1883->1883/tcp, [::]:1883->1883/tcp, 0.0.0.0:9001->9001/tcp, [::]:9001->9001/tcp myMqttBrkr
ffca88db76a6 mongo "/docker-entrypoint...." 8 minutes ago 27017/tcp myMongo
PS C:\Users\Dave\Desktop\RAILS>
```

Figure C.5: Windows Docker PS



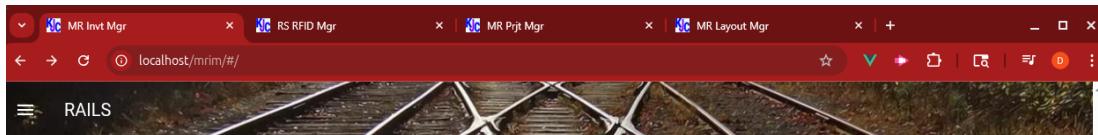
The screenshot shows the Docker Desktop interface on Windows. The left sidebar has sections for Ask Gordon, Containers (selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (Beta), Docker Hub, Docker Scout, and Extensions. The main area is titled "Containers" with a "Give feedback" link. It displays container usage statistics: Container CPU usage (2.42% / 800% (8 CPUs available)) and Container memory usage (502.8MB / 3.66GB). A search bar and a filter button ("Only show running containers") are at the top of the container list. The table lists 18 containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	rails	-	-	-	2.43%	16 minutes ago	[blue square] [dots] [red trash]
	myMrfm	bd9a7a080b65	dbristow/mrfm	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myMqttBrkr	4b065c953ee3	eclipse-mosquitto	1883:1883 [Show all ports (2)]	0.08%	16 minutes ago	[blue square] [dots] [red trash]
	myMongo	ffca88db76a6	mongo	-	1.74%	16 minutes ago	[blue square] [dots] [red trash]
	myPdds	55ec243ec530	dbristow/pdds	-	0.18%	16 minutes ago	[blue square] [dots] [red trash]
	myIpts	c3ab98b9bb18	dbristow/ipts	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myIpls	fa50b2f13932	dbristow/ipls	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myIsbs	5e06a81ce663	dbristow/isbs	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myRlds	7302d33294bc	dbristow/rlds	-	0.24%	16 minutes ago	[blue square] [dots] [red trash]
	myRls	5afef399c077	dbristow/rls	-	0.15%	16 minutes ago	[blue square] [dots] [red trash]
	myIsrs	8122b873dbf6	dbristow/isrs	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myIsts	74ac76da4147	dbristow/ists	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myMppm	698dea94d869	dbristow/mppm	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myMrIm	56fd6a3162d	dbristow/mrim	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myMrLm	1602e69bad47	dbristow/mrlm	-	0.03%	16 minutes ago	[blue square] [dots] [red trash]
	myIsms	c8b90b6bb7f6	dbristow/isms	-	0%	16 minutes ago	[blue square] [dots] [red trash]
	myRsrm	3da830ac87cf	dbristow/rsm	80:80 [green checkmark]	0%	22 minutes ago	[blue square] [dots] [red trash]
	myNginxProxy	72fd1f6d6cc	nginx:latest	80:80 [green checkmark]	0%	22 minutes ago	[blue square] [dots] [red trash]

At the bottom, status indicators show "Engine running", system resources (RAM 3.17 GB, CPU 0.75%, Disk 8.04 GB used / limit 250.92 GB), and a terminal icon with version v4.46.0.

Figure C.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.



About Model Railroad Inventory Manager

Version 5.1.3

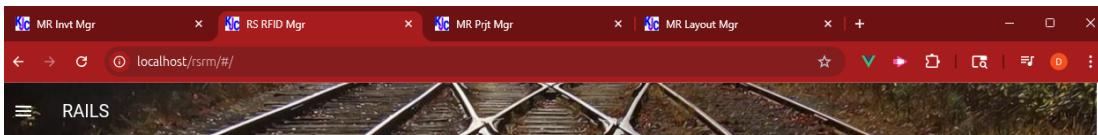
The Railway Administration and Information Logical System (RAILS) MRIM Application is one of several single page applications (SPAs) in the RAILS family of applications. It uses MongoDB, Vue 3, Vuetify and Node to construct the SPA. MRIM provides a user with a web SPA to show inventories of modeled railroad elements, such as rolling stock, AAR codes, structures, images, and companies. Rolling stock has a sub collections of locomotives along with DCC decoders. MRIM provides reports on the various elements as well as the ability to export and import CSV files.

The user guide is available at: tbd

The database has the following number of documents:

- Rollingstock: 0
- Decoders: 0
- AAR Codes: 0
- Structures: 0
- Companies: 0
- Images: 0

Figure C.7: RAILS MRIM Running



About RAILS RS RFID Manager

Version 6.0.5

The Railway Administration and Information Logical System (RAILS) RFID Application is one of several applications in the RAILS family of applications. It is a MongoDB, Express, Vue, and Node (MEVN) application that provides a user with a web application to show, when an RFID tag is read, the road name and number of the rolling stock associated with that tag. If there is no association with a piece of rolling stock input fields are provided to allow the user to enter the road name and number.

The user guide is available [here](#).

The database has the following number of documents:

- Rollingstock: 0 of which 0 have RFID tags.
- Micro Controllers: 0 of which 0 are RFID readers

Figure C.8: RAILS RSRM Running

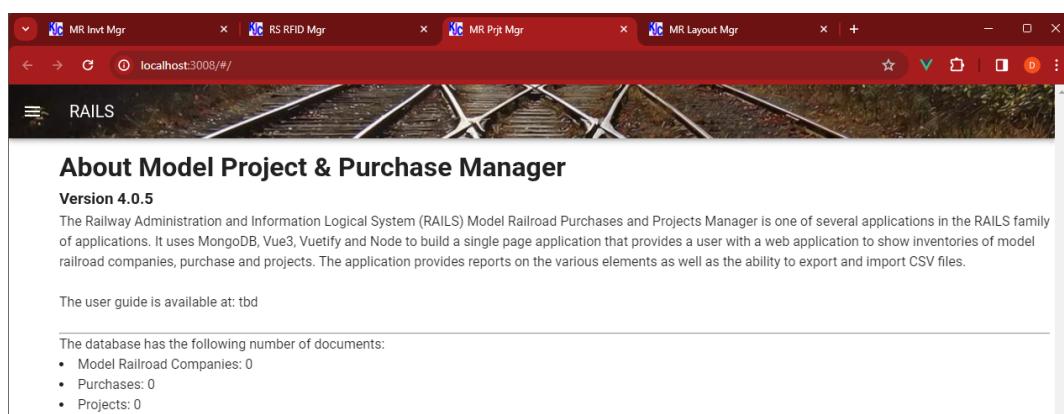


Figure C.9: RAILS MPPM Running

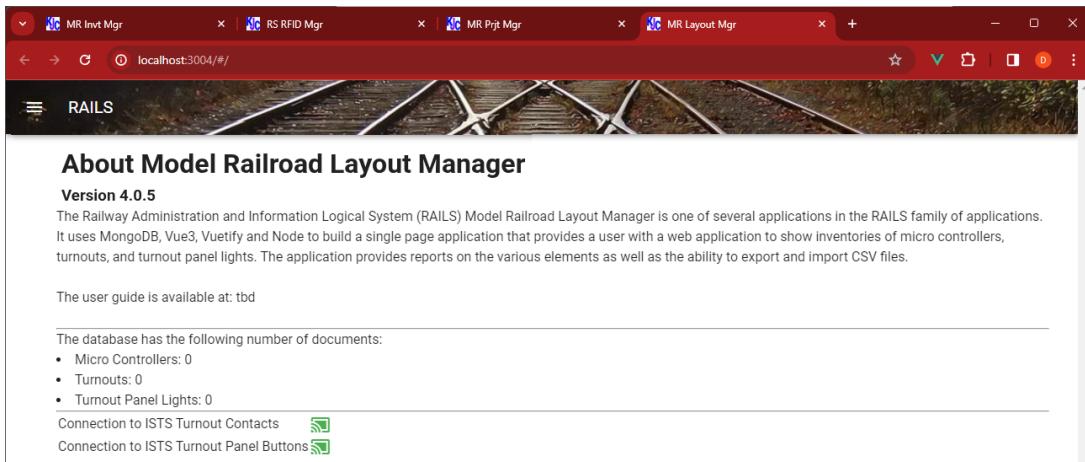


Figure C.10: RAILS MRLM Running

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

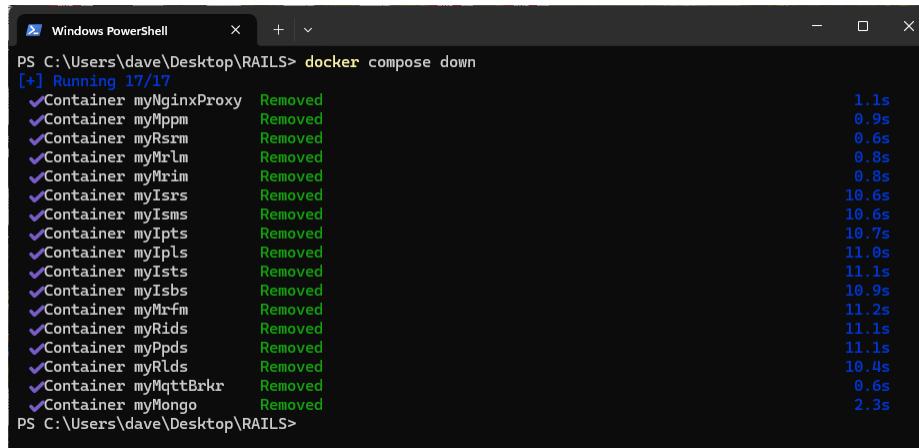


Figure C.11: Windows Docker Compose Down

C.2 Docker in a Linux Environment

The following screenshot shows the initial 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 C.12: Docker Setup mosquitto

The following screenshots show the Docker command used to pull images and run the RAILS Docker containers in a Linux environment. The process starts by checking for images in the local repository, not finding any Docker will pull them from Docker Hub, the progress is displayed. Once all of the images are pulled, as seen in the top half of figure C.13 Docker will then start each container. The bottom half of figure C.13 shows the successful creation of all of the running containers.

```
root@ottawa:/home/dbristow# docker compose up -d
[+] Running 87/35
✓ myRsrm Pulled                                17.5s
✓ myRlds Pulled                                 22.0s
✓ myMqttBrkr Pulled                            23.1s
✓ myPpds Pulled                                23.3s
✓ myIsbs Pulled                                 33.8s
✓ myIpts Pulled                                 43.1s
✓ myMongo Pulled                               62.5s
✓ myMrIm Pulled                                17.6s
✓ myMrfm Pulled                                44.9s
✓ myMpmp Pulled                                17.5s
✓ myIsts Pulled                                 29.6s
✓ myRids Pulled                                 21.7s
✓ myIsms Pulled                                43.0s
✓ nginx Pulled                                  43.3s
✓ myMrIm Pulled                                17.3s
✓ myIpIs Pulled                                43.1s
✓ myIsrs Pulled                                 23.9s

[+] Running 17/17
✓ Container myMrfm      Started                6.2s
✓ Container myMqttBrkr Started                6.3s
✓ Container myMongo     Started                6.3s
✓ Container myRlds      Started                3.0s
✓ Container myRids      Started                3.1s
✓ Container myPpds      Started                3.2s
✓ Container myIsrs      Started                2.7s
✓ Container myIpts      Started                2.9s
✓ Container myIpIs      Started                2.7s
✓ Container myIsbs      Started                2.7s
✓ Container myIsts      Started                2.8s
✓ Container myMrIm      Started                4.6s
✓ Container myMpmp      Started                4.7s
✓ Container myIsms      Started                4.4s
✓ Container myMrIm      Started                4.5s
✓ Container myRsrm      Started                4.8s
✓ Container myNginxProxy Started                5.5s
root@ottawa:/home/dbristow# 
```

Figure C.13: Linux Docker-Compose Up

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.

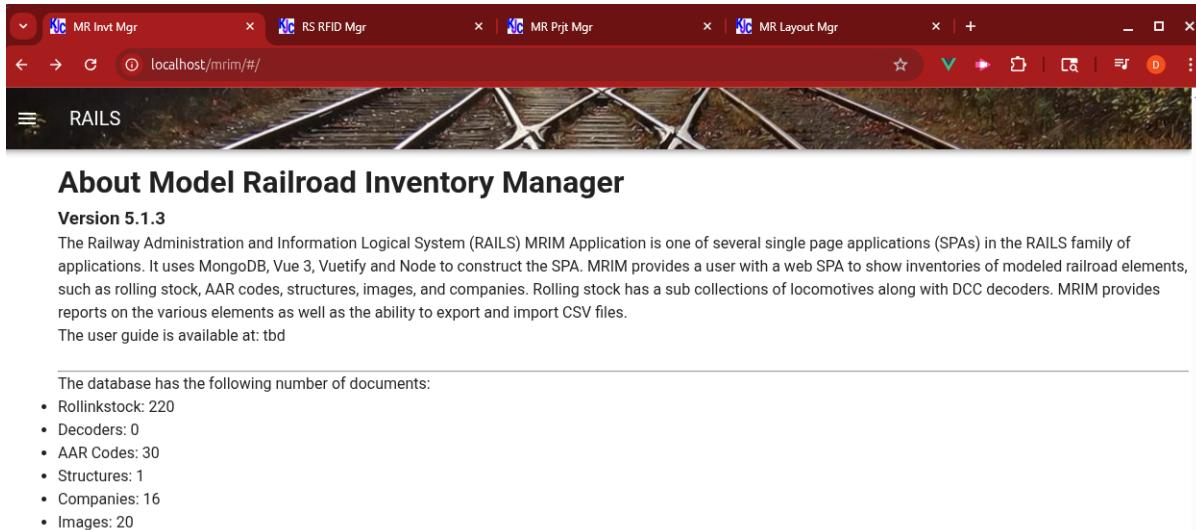


Figure C.14: RAILS MRIM Running

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

```
root@ottawa:/home/dbristow# docker compose down
[+] Running 17/17
✓ Container myNginxProxy  Removed          0.4s
✓ Container myMppm        Removed          0.6s
✓ Container myRsrn        Removed          0.7s
✓ Container myMrin        Removed          0.7s
✓ Container myMrml        Removed          0.6s
✓ Container myIsts        Removed          11.4s
✓ Container myIsbs        Removed          10.9s
✓ Container myIpls        Removed          11.2s
✓ Container myIpns        Removed          11.2s
✓ Container myPpds        Removed          11.4s
✓ Container myIsms        Removed          11.3s
✓ Container myIsrs        Removed          11.3s
✓ Container myMrfm        Removed          11.0s
✓ Container myRids        Removed          11.4s
✓ Container myMqttBrkr   Removed          0.6s
✓ Container myRlds        Removed          10.4s
✓ Container myMongo       Removed          0.4s
root@ottawa:/home/dbristow#
```

Figure C.15: Linux Docker-Compose Down

C.3 Docker in a MacOS Environment

Docker itself doesn't run directly on MacOS 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 mosquitto configuration file. Using the Docker Desktop to create the volumes and to edit the mosquitto configuration file, similarly to that done in Linux the result is shown in figure C.16

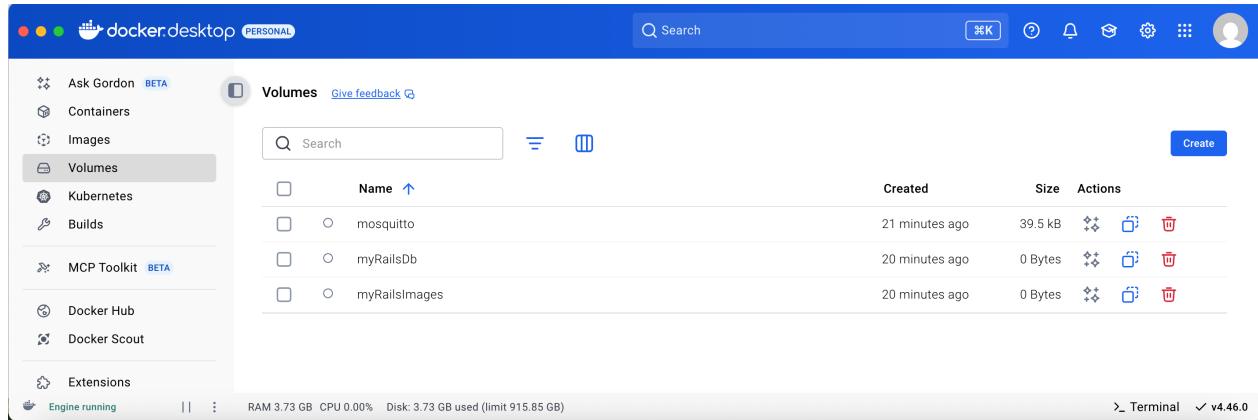


Figure C.16: Mac Docker Desktop Volumes

Open a Mac terminal to initiate Docker Compose to setup and run the RAILS container. Figure C.17 is a snapshot showing the pulling of the images from Docker Hub. Figure C.18 captures the process of pulling all of the images nearing completion. Figure C.19 shows the terminal once the “docker compose up -d” command has completed and all of teh containers are running.

```
davidbristow@MacBook-Pro RAILS % docker compose up -d
[+] Running 0/34
  myRsm Pulling
  myRsm [====] 38.96MB / 104.1MB Pulling
    "8c57b7c4e6b6 Downloading [=====>] 1.713MB/1.713MB
    "4f4fb700ef54 Downloading [=====>] 32B/32B
    "d187e437f729 Downloading [=====>] 10.49MB/28.23MB
    "64f3213fd4bd Downloading [=====>] 1.15MB/1.15MB
    "3c8daccdc842 Downloading [=====>] 3.319kB/3.319kB
    "49bc316433b7 Downloading [=====>] 7.34MB/12.26MB
    "118562997779 Downloading [=====>] 9.961MB/51.71MB
    "2929c9881585 Downloading [=====>] 9.437MB/10.15MB
    "956d818f7a54 Downloading [=====>] 13.8kB/13.8kB
    "d7dec3d6ad98 Downloading [=====>] 449B/449B
    "6835f124c0e9 Downloading [=====>] 119B/119B
  myPds Pulling
  myIsts Pulling
  myRids [ ] Pulling
    "63fdc030f085 Pulling fs layer
    "427c146d9c75 Downloading [=====>] 417B/417B
    "2f3f7e218edc Pulling fs layer
  myIsbs Pulling
  myMrlm Pulling
  myMongo Pulling
  myIppls Pulling
  myIisms Pulling
  myMrfm Pulling
  myMpmm Pulling
  myKrim Pulling
  nginx Pulling
  mySrs Pulling
  myRlds [ ] Pulling
    "6c9c20fbdbef Pulling fs layer
    "c59d44327534 Pulling fs layer
    "bb846ac4b2aa Pulling fs layer
  myMqttrkr Pulling
```

Figure C.17: MacOS Docker Compose Up Pulls

```
davidbristow@MacBook-Pro RAILS % docker compose up -d
[+] Running 18/87
  myRsrm [::] 3.144MB / 3.144MB Pulling
  myIpts [::] 104.1MB / 104.1MB Pulling
  myPdps [::] 11.18MB / 11.18MB Pulling
  myIsts [::] 12.05MB / 12.05MB Pulling
  myRids [::] 9.639MB / 9.639MB Pulling
  myIsbs [::] 13.01MB / 13.01MB Pulling
  myMrIm [::] 5.946MB / 5.946MB Pulling
  myMongo [::] Pulling
  myIpls [::] 22.41MB / 22.41MB Pulling
  myLisms [::] 21.65MB / 21.65MB Pulling
  myMrFm [::] 85.07MB / 85.07MB Pulling
  myMpPm [::] 3.135MB / 3.135MB Pulling
  myMrIm [::] 3.141MB / 3.141MB Pulling
  nginx [::] Pulling
  myIsrs [::] 12.05MB / 12.05MB Pulling
  myRldS [::] 9.586MB / 9.586MB Pulling
  myMqtBrkr [::] Pulling
```

Figure C.18: MacOS Docker Compose Up Completing Pulls

```
davidbristow@MacBook-Pro RAILS % docker compose up -d
[+] Running 83/87
  ✓ myRsm Pulled
  ✓ myIpts Pulled
  ✓ myPds Pulled
  ✓ myIsts Pulled
  ✓ myRids Pulled
  ✓ myIsbs Pulled
  ✓ myMrlm Pulled
  ✓ myMongo Pulled
  ✓ myIpls Pulled
  ✓ myIsms Pulled
  ✓ myRfrm Pulled
  ✓ myMppm Pulled
  ✓ myMrilm Pulled
  ✓ nginx Pulled
  ✓ myIsrs Pulled
  ✓ myRlds Pulled
  ✓ myMqtBrkr Pulled
[+] Running 17/17
  ✓ Container myMrfm Started 1.3s
  ✓ Container myMongo Started 1.3s
  ✓ Container myMqtBrkr Started 1.4s
  ✓ Container myRids Started 1.7s
  ✓ Container myPds Started 1.9s
  ✓ Container myRlds Started 1.8s
  ✓ Container myIpts Started 2.0s
  ✓ Container myIsrs Started 1.9s
  ✓ Container myIsbs Started 2.0s
  ✓ Container myIpls Started 2.0s
  ✓ Container myIsts Started 1.9s
  ✓ Container myIsms Started 2.4s
  ✓ Container myMppm Started 2.5s
  ✓ Container myMrilm Started 2.2s
  ✓ Container myMrlm Started 2.8s
  ✓ Container myRsm Pulled 3.1s
  ✓ Container myNginxProxy Started 3.4s
davidbristow@MacBook-Pro RAILS %
```

Figure C.19: MacOS Docker Compose Up Started

Figure C.20 shows in the Docker Desktop GUI that all the images are now local to the Mac.

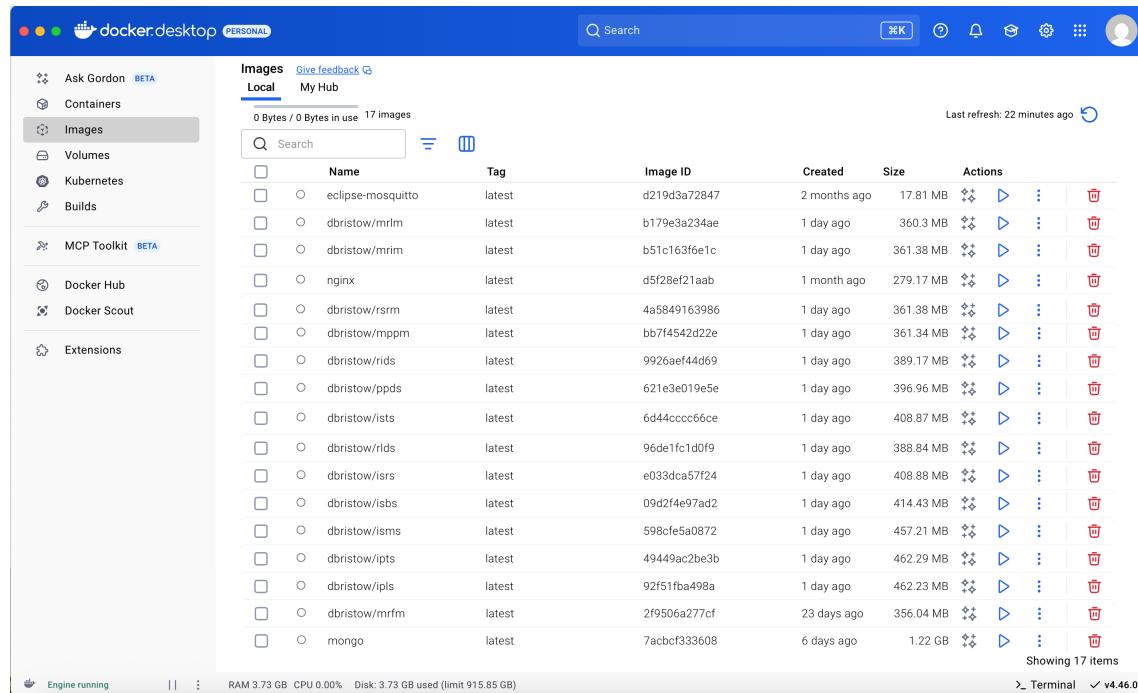


Figure C.20: MacOS Docker Desktop Images

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.

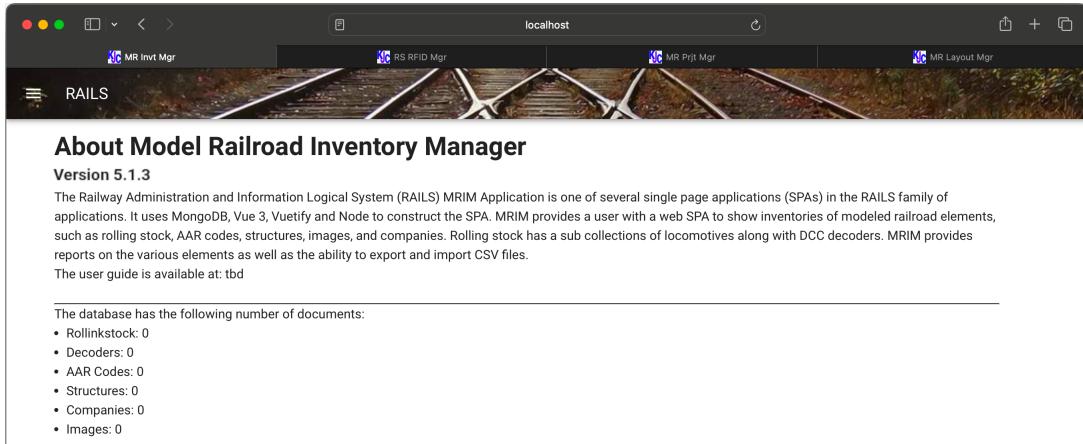


Figure C.21: RAILS MRIM Running

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

```
[davidbristow@MacBook-Pro RAILS % docker compose down
[+] Running 17/17
✓ Container myNginxProxy  Removed          0.0s
✓ Container myMrilm      Removed          0.0s
✓ Container myMppm       Removed          0.1s
✓ Container myRsrm       Removed          0.1s
✓ Container myMrim       Removed          0.1s
✓ Container myIpis       Removed          0.1s
✓ Container myIsts       Removed          0.1s
✓ Container myIsbs       Removed          0.1s
✓ Container myIpnts      Removed          0.1s
✓ Container myIsms       Removed          0.1s
✓ Container myIsrs       Removed          0.1s
✓ Container myMrfm       Removed          0.1s
✓ Container myRids       Removed          0.1s
✓ Container myPpds       Removed          0.1s
✓ Container myRlds       Removed          0.1s
✓ Container myMqttrkr     Removed          0.1s
✓ Container myMongo       Removed          0.0s
davidbristow@MacBook-Pro RAILS % ]
```

Figure C.22: MacOS DockerCompose Down

Appendix D

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 tagged images
- `docker images ls -a` - List all images
- `docker rmi <image-id>` - Remove an image.
- `docker volume ls` - List volumes.
- `docker volume prune` - This will remove all local volumes not used by at least one container. This command is designed to clean up any volumes that are not currently attached to a container, which is a good thing to do because after running RAILS as Docker Compose creates some randomly named volumes. Be sure to do this while all of the RAILS containers are running otherwise the volumes: mosquito, myRailsDb, and myRailsImages may be destroyed causing a complete lose of your data.
- `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 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.yaml`).
- `docker-compose -f docker-compose-dev.yaml up -d` - Start the services defined in an alternate Docker Compose file.
- `docker-compose down` - Stop the services defined in the Docker Compose file.

Acronyms

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 4

command line interface (CLI) is a text based way to interact with a computer program by typing commands and seeing text based responses 4, 5, 6, 21, 30

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

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, 22, 30, 32

Internet of Things (IoT) 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 2, 3

IoT Publisher Turnout Panel Light Services (IPLS) 3

IoT Publisher Turnout Services (IPTS) 3

infrared (IR) 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

IoT Subscriber Turnout Panel Button Services (ISBS) 3

IoT Subscriber Location Services (ISLS) 3

IoT Subscriber Micro-controller Services (ISMS) 3

IoT Subscriber RFID Services (ISRS) 3

IoT Subscriber Turnout Services (ISTS) 3

Model Projects and Purchase Manager (MPPM) 4, 10

Message Queuing Telemetry Transport (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 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

Railway Administration and Information Logical System (RAILS) 2, 3, 4, 6, 7, 8, 9, 10, 21, 23, 24, 26, 28, 29, 30, 32, 33, 34

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

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

Rollingstock RFID Manager (RSRM) 3, 4, 10

Single Page Application (SPA) 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, 24, 28, 32, 34

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, 29, 30

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 21, 22

yet another markup language (YAML) is a human-readable data serialization language used for configuration files, data exchange, and system automation, known for its use of indentation to define structure, similar to Python, and for being a strict superset of JSON 6, 8, 10