Lab Experiment: Packaging Jupiter Robot Software with Docker and Docker Swarm

Ahmed Métwalli

March 23, 2025

Contents

1 Introduction

This lab document is designed for a hands-on experiment with the Jupiter robot's software environment (running ROS Melodic) and demonstrates how to package it into Docker containers. The focus is on ensuring that these containers can interact with the robot's hardware (such as audio devices) as if running directly on the local system. We will also scale the deployment using Docker Swarm to interconnect two Jupiter robots for distributed task management in a swarm intelligence context.

Note to Students: Throughout this lab, you are encouraged to document every trial, error, and observation. Understanding the "why" behind each configuration step is key to mastering these concepts.

2 Objectives

- **Package the Environment:** Containerize the ROS Melodic environment and all required dependencies (e.g., speech recognition libraries) into Docker.
- **Enable Hardware Interaction:** Ensure that the container can access host hardware (audio interfaces) just as if running from the local terminal.
- **Achieve Scalability:** Deploy the containerized application using Docker Swarm across two Jupiter robots, ensuring distributed management and fault tolerance.

3 Prerequisites

3.1 Hardware

- Two Jupiter robots equipped with audio devices and sensors.

3.2 Software

- ROS Melodic installed on each robot.
- Docker Engine and Docker Swarm.
- Ubuntu (or a compatible Linux distribution).
- Speech recognition libraries (`pocketsphinx`) and audio drivers.

4 Workflow Overview

The lab is divided into these key phases:

Step 1: Audit Dependencies: Identify and document all software packages and libraries currently in use.

Step 2: Containerization: Develop a Dockerfile that installs ROS Melodic and additional dependencies.

Step 3: Hardware Integration: Configure the container to map host hardware devices (e.g., audio).

Step 4: Local Testing: Run the container on one robot and validate its access to hardware.

Step 5: Swarm Deployment: Set up Docker Swarm and deploy the containerized application across both robots.

5 Detailed Workflow and Documentation

5.1 Step 1: Audit Software Dependencies

Objective: List and verify all required ROS and system packages.

- **Check ROS packages:**

```
rospack list
```

Listing 1: List ROS Packages

- **Review the package.xml:** Ensure all dependencies are clearly listed.
- **System Packages:** Use `dpkg -l` to verify installations of audio libraries and other system dependencies.

Documentation Tip: Record any discrepancies or missing packages during this audit.

5.2 Step 2: Containerize the Application

Objective: Create a Dockerfile that replicates your ROS environment along with additional libraries for speech recognition.

```
FROM osrf/ros:melodic-desktop-full

# Install additional dependencies for speech recognition and audio
    handling
RUN apt-get update && apt-get install -y \
    pocketsphinx \
    alsa-utils \
    pulseaudio

# Copy your ROS workspace into the container
COPY ./catkin_ws /catkin_ws
WORKDIR /catkin_ws
RUN /bin/bash -c "source /opt/ros/melodic/setup.bash && catkin_make"

# Configure ROS environment variables for communication
ENV ROS_MASTER_URI=http://<manager-ip>:11311
ENV ROS_IP=<robot-ip>

# Start the speech recognition node
CMD ["roslaunch", "your_speech_recognition_package", "launch_file.
    launch"]
```

Listing 2: Sample Dockerfile

Note: Replace `<manager-ip>` and `<robot-ip>` with the correct addresses.

5.3 Step 3: Enable Hardware Access

Objective: Allow the container to interact with the robot's audio hardware.

```
docker run --device /dev/snd -v /run/user/1000/pulse:/run/user/1000/
    pulse your_docker_image
```

Listing 3: Running Container with Audio Hardware Access

Observation: If you encounter issues with audio capture, verify that:

- The host's `/dev/snd` is correctly mapped.
- The PulseAudio socket path matches your system configuration.

5.4 Step 4: Test the Container Locally

Objective: Build and run the container on one Jupiter robot to confirm that it accesses hardware as expected.

1. **Build the Image:**

```
docker build -t jupiter-speech .
```

Listing 4: Build Docker Image

2. **Run and Test:**

```
docker run --device /dev/snd -v /run/user/1000/pulse:/run/user
   /1000/pulse jupiter-speech
```

Listing 5: Test Container Execution

Documentation Tip: Note any errors or unexpected behaviors. Use these insights to adjust device mappings or environment configurations.

5.5 Step 5: Deploy with Docker Swarm for Scalability

Objective: Scale the setup by connecting two Jupiter robots in a Docker Swarm cluster.

5.5.1 Initialize Docker Swarm on the Manager Robot

```
docker swarm init
```

Listing 6: Initialize Swarm

This command outputs a join token and manager IP address. Record these details.

5.5.2 Join the Worker Robot

On the second robot, join the swarm:

```
docker swarm join --token <JOIN_TOKEN> <MANAGER_IP>:2377
```

Listing 7: Join Worker Node

5.5.3 Verify the Swarm Setup

From the manager node, list the swarm nodes:

```
docker node ls
```

Listing 8: List Swarm Nodes

5.5.4 Deploy the Application via Docker Compose

Create a `docker-compose.yml` file:

```
version: "3.8"
services:
  speech_recognition:
    image: jupiter-speech
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
    networks:
      - robot-net
    devices:
      - /dev/snd:/dev/snd
    volumes:
      - /run/user/1000/pulse:/run/user/1000/pulse

networks:
  robot-net:
    driver: overlay
```

Listing 9: docker-compose.yml

Deploy the stack:

```
docker stack deploy -c docker-compose.yml jupiter_stack
```

Listing 10: Deploy the Application Stack

Troubleshooting Note: Monitor the swarm services using `docker service ls` and `docker service ps`. Document any issues with network connectivity or hardware access and experiment with configurations until resolved.

6 Expected Outcomes and Documentation

Upon successful completion:

- The containerized ROS node should interact with the robot's audio hardware seamlessly.
- Both Jupiter robots will be connected in a Docker Swarm, ensuring distributed management.
- The system is scalable and resilient, with clear logs for any faults encountered.

Documentation: Make sure to record each trial, the configuration changes made, and any observed behavior. This will serve as an invaluable resource for troubleshooting and future improvements.