Docker to Rails Local Setup - Complete Journey

Problem Summary

- Docker Desktop kept crashing with `com.docker.build: exit status 1` and WSL termination errors
- Tried multiple Docker fixes without success
- Needed to set up a Ruby on Rails project locally

Solution Path

1. Abandoned Docker/Podman Approach

Why it failed:

- Docker Desktop WSL integration issues on Windows
- Podman alternative had similar complications
- Project was designed for containerized development but containers kept failing

Lesson: Sometimes the "recommended" approach (Docker) isn't always the most practical for local development.

2. Switched to Native Rails Development

Key insight: The project README mentioned you could "develop without Docker" - this was the winning approach.

3. Environment Setup Challenges

Challenge A: Missing Environment Variables

- Project required 70+ environment variables from AWS Secrets Manager
- AWS credentials were "outdated" according to team
- **Solution:** Manually created `.env` file with required variables

Challenge B: Database Connection Issues

- PostgreSQL authentication failures
- Password mismatches between Rails config and database
- **Solution:** Reset PostgreSQL authentication and created proper user

Step-by-Step Solution

1. Install Prerequisites

bash

```bash
# Ruby and Rails already installed
bundle install
```

2. Create Environment File

bash

```bash
# Create .env with all required variables
cat > .env << 'EOF'
DATABASE_HOST=localhost
DATABASE_USER=postgres
DATABASE_PASSWORD=ptogenius_db_pwd
DATABASE_NAME=ptogenius
DATABASE_PORT=5432
RAILS_MAX_THREADS=5
ALERT_SYSTEM_PUT_SERVER_IN_MAINTENANCE_MODE=false
# ... (70+ other variables)
EOF
```

3. Fix Log Permissions

bash

```bash
sudo chown -R $USER:$USER .
mkdir -p log
touch log/development.log
chmod 0664 log/development.log
```

4. PostgreSQL Setup

bash

```bash
# Install PostgreSQL
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo service postgresql start

# Configure authentication
sudo sed -i 's/peer/trust/' /etc/postgresql/*/main/pg_hba.conf
sudo service postgresql restart

# Create database and user
sudo -u postgres psql
ALTER USER postgres PASSWORD 'ptogenius_db_pwd';
CREATE DATABASE ptogenius OWNER postgres;
\q

# Restore authentication
sudo sed -i 's/trust/md5/' /etc/postgresql/*/main/pg_hba.conf
sudo service postgresql restart
```

5. Rails Database Setup

bash

```bash
cp config/database.ci.yml config/database.yml
rails db:create db:migrate
```

Key Lessons Learned

1. Container Complexity vs Local Development

- **When to use containers:** Production, CI/CD, team consistency
- **When to avoid:** Local development troubleshooting, when containers add more problems than they solve

2. Documentation Reality Check

- README mentioned Docker as "recommended" but also had native development instructions
- Sometimes the "backup" approach is actually more reliable

3. Environment Variable Management

- Large applications have complex environment requirements
- AWS Secrets Manager is great for production, but adds complexity for local dev
- Manual `.env` files are sometimes the pragmatic choice

4. Database Authentication Evolution

- PostgreSQL default authentication has changed over versions
- `peer` vs `md5` authentication modes matter
- Sometimes you need to temporarily use `trust` mode to reset passwords

5. Permission Management in WSL

- File permissions can get complex in WSL environments
- `chown -R $USER:$USER .` often fixes mysterious permission issues
- Log files need specific permissions for Rails

Tools and Technologies Used

Successful Stack

- **OS:** WSL2 Ubuntu
- **Language:** Ruby (with rbenv/rvm)
- **Framework:** Ruby on Rails
- **Database:** PostgreSQL (native installation)
- **Environment:** Manual `.env` file

Abandoned Approaches

- Docker Desktop (WSL integration issues)
- Podman Desktop (similar container complexities)
- AWS Secrets Manager (authentication problems)

Final Working Commands

```bash
# Check everything is working
rails db:setup  # Should work without errors
rails server    # Start the application
```

Troubleshooting Tips for Future

Container Issues

1. Try native development first before debugging containers
2. Check if project has non-container development instructions
3. WSL + Docker often has permission/integration issues

Environment Variables

1. Start with minimal `.env` and add variables as errors occur
2. Check for formatting issues (no spaces around `=`, proper line endings)
3. Some variables can be empty strings, others need real values

Database Connection

1. Verify PostgreSQL is running: `sudo service postgresql status`
2. Test connection directly: `psql -h localhost -U postgres -d postgres`
3. Check authentication mode in `pg_hba.conf`

Permission Problems

1. Fix ownership: `sudo chown -R $USER:$USER .`
2. Create missing directories: `mkdir -p log tmp`
3. Set proper permissions: `chmod 0664 log/development.log`

Time Investment

- **Container debugging:** ~2 hours (unsuccessful)
- **Native setup:** ~1 hour (successful)
- **Total lesson:** Sometimes the "harder" path is actually easier

Conclusion

This experience demonstrates that modern development environments can be complex, and sometimes the recommended approach isn't the most practical. Being willing to pivot from containers to native development saved significant time and frustration.

The key was recognizing when to abandon a problematic approach and try alternatives mentioned in the documentation.