ODE Development Environment

Standard Operating Procedures

Version 1

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Date | Author (s) | Comment |
| 1 | 01/07/2021 | Brandon Payne | First version of the ODE Development Environment SOP |

**Table of Contents**

[1 Introduction 1](#_Toc60924607)

[1.1 Purpose of the Document 1](#_Toc60924608)

[1.2 Definition, Acronyms, and Abbreviations 1](#_Toc60924609)

[1.3 Document Overview 1](#_Toc60924610)

[2 Description of software requirements for development environment 2](#_Toc60924611)

[2.1 Overview of development environment 2](#_Toc60924612)

[2.2 GitHub Repository 2](#_Toc60924613)

[2.3 Visual Studio Code 2](#_Toc60924614)

[2.4 Docker 2](#_Toc60924615)

[2.5 Remote Development Extension Pack 3](#_Toc60924616)

[3 ODE Remote Container Overview 3](#_Toc60924617)

[3.1 devcontainer.json 3](#_Toc60924618)

[3.1.1 Settings 4](#_Toc60924619)

[3.1.2 Extensions 4](#_Toc60924620)

[3.1.3 ForwardPorts 5](#_Toc60924621)

[3.2 docker-entrypoint.sh 6](#_Toc60924622)

[3.3 Dockerfile 6](#_Toc60924623)

[3.4 jpo-ode.code-workspace 6](#_Toc60924624)

[3.5 kafka 6](#_Toc60924625)

[3.6 post-create.sh 7](#_Toc60924626)

[4 ODE Remote Container Development 7](#_Toc60924627)

[4.1 Debugging 7](#_Toc60924628)

[4.2 Additional Tools 9](#_Toc60924629)

[4.2.1 kafkacat 9](#_Toc60924630)

[4.2.2 telnet 9](#_Toc60924631)

**List of Figures**

[Figure 3‑1 Initial devcontainer.json file (Source: Brandon) 4](#_Toc60924632)

[Figure 3‑2 Kafka Service Status Example (Source: Brandon) 6](#_Toc60924633)

[Figure 4‑1 Dev Container Configuration Prompt(Source: Brandon) 7](#_Toc60924634)

[Figure 4‑2 Debug Tab(Source: Brandon) 8](#_Toc60924635)

[Figure 4‑3 Terminal(Source: Brandon) 8](#_Toc60924636)

[Figure 4‑4 ODE Demo Console(Source: Brandon) 9](#_Toc60924637)

[Figure 4‑5 Dev Container Zookeeper Telnet (Source: Brandon) 10](#_Toc60924638)

[Figure 4‑6 Dev Container Zookeeper Stats(Source: Brandon) 10](#_Toc60924639)

**List of Tables**

[Table 1‑1 Glossary of Terms. 1](#_Toc60924640)

[Table 1‑2. Acronym List. 1](#_Toc60924641)

# Introduction

## Purpose of the Document

To document the standard ODE development environment and standard operating procedures.

## Definition, Acronyms, and Abbreviations

Table 1‑1 Glossary of Terms.

| **Term** | **Definition** |
| --- | --- |
| ODE | US Department of Transportation (USDOT) Intelligent Transportation Systems (ITS) Joint Program Office (JPO) Operational Data Environment (ODE). The ITS ODE is a real-time virtual data router that ingests and processes operational data from various connected devices - including vehicles, infrastructure, and traffic management centers - and distributes it to other devices and subscribing transportation management applications. |
|  |  |

Table 1‑2. Acronym List.

| **Acronym/Abbreviation** | **Definition** |
| --- | --- |
| CV | Connected Vehicle |

## Document Overview

This document is organized into sections to describe the setup of the ODE development environment.

The remainder of this document if organized as follow:

* Section 2 provides an overview of software requirements for ODE development environment.
* Section 3 provides detailed explanation and overview of the ODE Remote Container
* Section 4 provides explanation of developing within the remote container

# Description of software requirements for development environment

## Overview of development environment

The ODE development environment is designed to be operating system agnostic, and the tools chosen can run in many environments. The development environment will consist of the following software requirements:

* GitHub repository
* Visual Studio Code
* Docker
* Remote Development Extension Pack

## GitHub Repository

A required element of this set up is to have access to the GitHub repository. For this document, we will use the JPO ODE repository located at: <https://github.com/usdot-jpo-ode/jpo-ode>

## Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. Numerous extensions exist that allow for a development experience customized to the application, such as the remote development extension outlined below. More detailed explanation on this software is located here: <https://code.visualstudio.com/>. For the purposes of the ODE development environment, Visual Studio Code will act as our primary source code editor and may be downloaded from <https://code.visualstudio.com/Download>.

## Docker

Docker is an industry standard software for running containers. Containers are a standardized unit of software that allows developers to isolate their app from its environment, solving the “it works on my machine” headache. A Docker container not only runs a given application but allows the developer to specify the exact environment in which to run the code. More details can be found on the website here: <https://www.docker.com/>. For the ODE development environment, Docker Desktop must be installed to run the extension pack. Docker Desktop can be downloaded from <https://www.docker.com/get-started>.

## Remote Development Extension Pack

The Visual Studio Code Remote - Containers extension lets you use a Docker container as a full-featured development environment. It allows you to open any folder inside (or mounted into) a container and take advantage of Visual Studio Code's full feature set. In layman’s terms, this extension allows us to customize a development environment along with various required tools and distribute this environment along with the code base to allow developers to avoid common pitfalls with setting up a new environment. More detailed explanation on this software is located here: <https://code.visualstudio.com/docs/remote/containers>.

# ODE Remote Container Overview

Within the GitHub repository, several files compose the remote container development environment. These files are aggregated under the **.devcontainer** directory. Each is outlined below, with details as to their contents. Note that this section is an overview and explanation of the files themselves and what they do. To use the development environment, see section 4 below.

## devcontainer.json

The container configuration of the ODE development environment is housed in this **devcontainer.json** file. The file is similar to a **launch.json** file but is used instead to launch the development container. The initial file is given below:

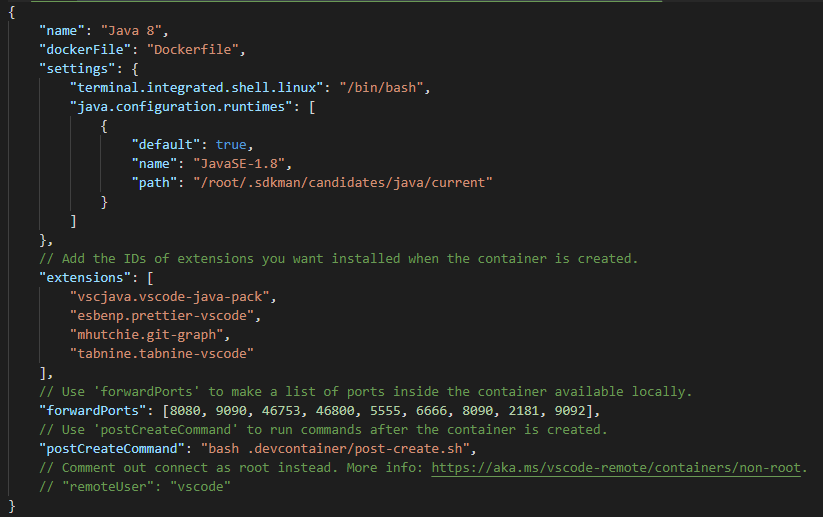


Figure 3‑1 Initial devcontainer.json file (Source: Brandon)

Many of these fields are self-explanatory, however there are a few customizations for the ODE container worth explaining.

### Settings

The settings field is used to pass desired settings into the container for use. As detailed in Dockerfile below, we are running two simultaneous version of Java using sdkman to allow our extensions to run properly. Because of this, we call out the sdkman “current” runtime explicitly in the “java.configuration.runtimes” property.

### Extensions

This section allows us to specify VS Code extensions to install on container startup. These extensions allow for a customized development environment and support of our chosen languages and styles.

#### vscjava.vscode-java-pack

This is a Java extension pack composed of several other extensions to help with Java development. It installs the following extensions:

1. [Language Support for Java™ by Red Hat](https://marketplace.visualstudio.com/items?itemName=redhat.java)
   * Code Navigation
   * Auto Completion
   * Refactoring
   * Code Snippets
2. [Debugger for Java](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-debug)
   * Debugging
3. [Java Test Runner](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-test)
   * Run & Debug JUnit/TestNG Test Cases
4. [Maven for Java](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-maven)
   * Project Scaffolding
   * Custom Goals
5. [Project Manager for Java](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-dependency)
   * Manage Java projects, referenced libraries, resource files, packages, classes, and class members
6. [Visual Studio IntelliCode](https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscodeintellicode)
   * AI-assisted development
   * Completion list ranked by AI

#### esbenp.prettier-vscode

This is a [Prettier Formatter](https://prettier.io/) extension to enforce consistent styling of code.

#### mhutchie.git-graph

Git Graph allows users to view a graph of the repository and perform Git actions from this graph.

#### tabnine.tabnine-vscode

Tabnine is an artificial intelligence assistant that runs within VS Code. This extension generates code completion suggestions and allows for much faster code generation as well as fewer mistakes.

### ForwardPorts

This section allows us to open ports from the development container to our host machine. In this way we can debug and access services, web pages, etc. from the host machine itself. Necessary ports for debugging the various ODE applications have been opened here already. If further ports need added in the future, this section will be modified accordingly.

## docker-entrypoint.sh

This script file is used as the entry point for the Docker container. Once built, the container will execute code found within this file. For the ODE, this file is used to manage zookeeper and Kafka instances. On start of the container, zookeeper and Kafka are started via the kafka service mentioned below. On container shutdown, these services are shut down properly to avoid issues with killing their respective processes outright.

## Dockerfile

The Dockerfile here performs in the same way as a typical Dockerfile does. It is a text document that contains all the commands used to assemble an image. For the ODE Development environment, this image is built on the base image of **openjdk:11-jdk**. This is done because our extensions above (including Java language support) require a more modern Java environment to run. On top of this image, we install a few tools that may be used within the development environment including zip/unzip, curl, and snmp among others. [SDKMAN](https://sdkman.io/) is also installed and used to install the Java 8 SDK required by the ODE, as well as Maven for building the ODE.

## jpo-ode.code-workspace

The workspace file is used to help create a consistent development experience across machines. In the case of the ODE, settings have been added to control behavior of the Java tooling. The “java.configuration.updateBuildConfiguration” property is set to “automatic” to enable the environment to stay in sync with any build configuration changes without requiring user input. The “java.server.launchMode” property is set to “Standard” to allow a full load of the Java server rather than a shallow but quick load, enabling better code navigation.

## kafka

This file is used to create a service to manage zookeeper and Kafka. The service contains basic controls to start, stop, restart, and check the status of these applications. For instance, to verify the service is running within the container, simply execute `sudo service kafka status`. If successful, a similar message to following will be seen:



Figure 3‑2 Kafka Service Status Example (Source: Brandon)

## post-create.sh

This script is referenced from the devcontainer.json and is executed a single time after container creation. This enables the ODE Development environment to configure one-time settings such as setting up required Kafka topics required by the ODE.

# ODE Remote Container Development

Development within a remote container is relatively seamless and should perform much the same way as development on a local machine. Providing the above required software is installed and Docker is running, clone the repository to your local machine and open the parent **jpo-ode** folder using VS Code. The below prompt should pop up:

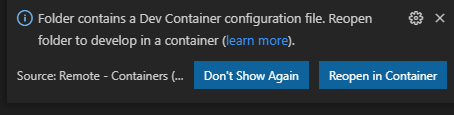


Figure 4‑1 Dev Container Configuration Prompt(Source: Brandon)

Select the “Reopen in Container” option and allow the Docker container to build. This first build may take some time as it is creating new container from scratch. After this initial build, subsequent load times will be much quicker. Once the container has built, development may continue as usual. Required tools for developing against the ODE are installed in the container and Maven build commands will run properly.

## Debugging

Debugging capabilities have been added to the development container. Various projects may be launched and debugged individually or as a group. To access this, select the Debug tab on the far left.

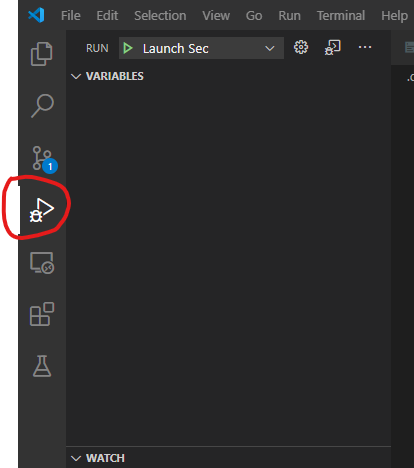


Figure 4‑2 Debug Tab(Source: Brandon)

From this view, developers may select a project to debug from the dropdown near the top. “Launch ODE” launches the main application including subscribing to various Kafka topics. Note that because this is somewhat computationally intensive it may take a few seconds to fully start. During this startup process, the terminal at the bottom of VS Code will display progress and logging as below. The terminal will likely pause at this screen for several seconds while the ODE boots up.

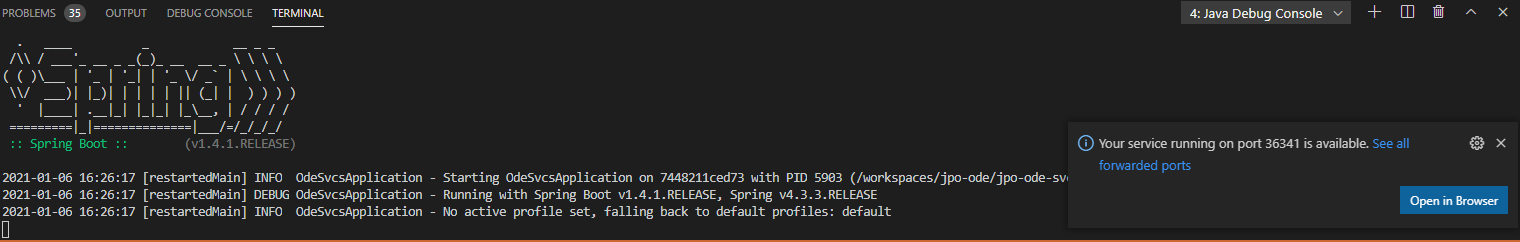


Figure 4‑3 Terminal(Source: Brandon)

Once the ODE fully starts, the terminal will display various logging messages related to subscribing to topics and mapping paths. To verify the ODE is running locally, open a browser to <http://localhost:8080/> and verify the Demo Console appears as below.

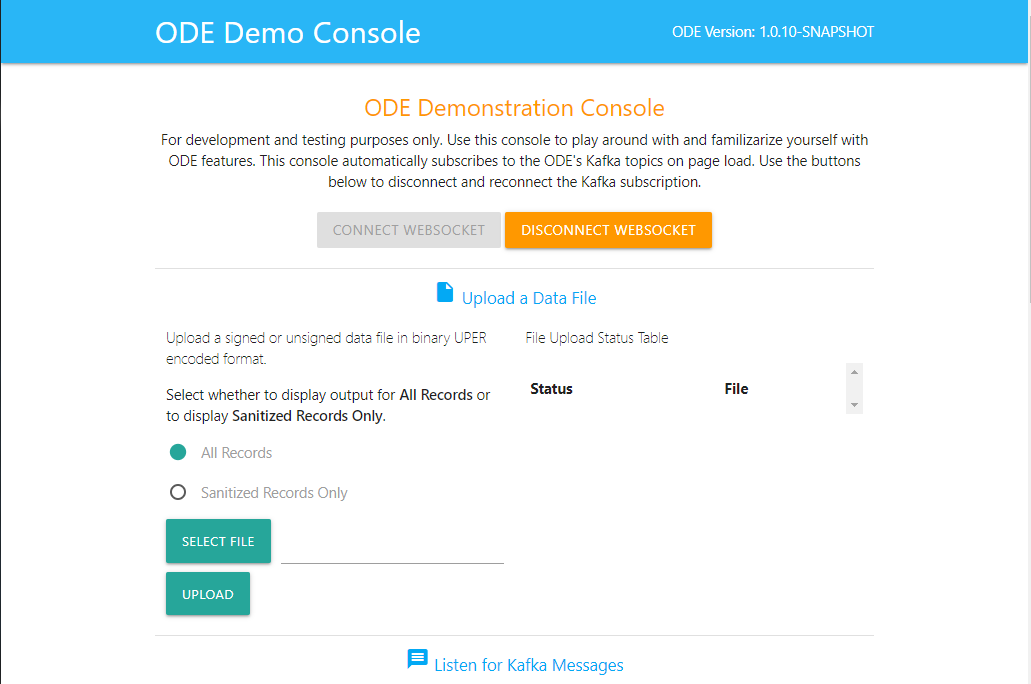


Figure 4‑4 ODE Demo Console(Source: Brandon)

## Additional Tools

Several additional tools have been included in the default development container to enable better debugging and troubleshooting capabilities. These are detailed below.

### kafkacat

kafkacat is a tool used to help debug and interact with Kafka. Information can be found on its [GitHub](https://github.com/edenhill/kafkacat), where it is described as “netcat for Kafka”. For use with the ODE, kafkacat is used to verify topic creation, as well as consume and produce messages for testing various ODE components.

### telnet

Telnet is a network protocol used to connect to remote systems. It has been included here to help troubleshoot when issues arise. For instance, to verify zookeeper is running we can run ‘telnet localhost 2181’ which produces the following on success:

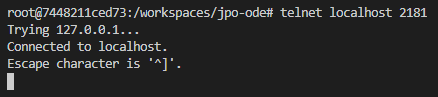


Figure 4‑5 Dev Container Zookeeper Telnet (Source: Brandon)

Further, we can enter ‘stats’ to get statistics about the zookeeper instance.

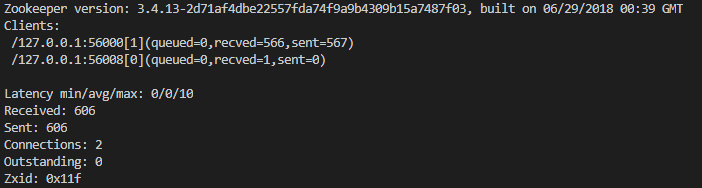


Figure 4‑6 Dev Container Zookeeper Stats(Source: Brandon)