Docker Setup Guide for AI Coding Agents

Alex Rieber (@AlexRieber) · Ulm University

2026-02-24

Table of contents

Who is this for? Students who have never used Docker before and want to set up a safe, reproducible environment for running AI coding agents (Claude Code, OpenAI Codex, Google Gemini CLI, and Aider) alongside R, Python, and Stata.
What you need: A Windows, Mac, or Linux computer with at least 16 GB RAM and 80 GB free disk space.
Workshop materials: All files (Dockerfile, docker-compose.yml, CLAUDE.md, this guide) are available at github.com/AlexRieber/Workshops/Docker.

---

1 Part 1: What is Docker and Why Do We Need It?

1.1 The Problem

Imagine you want to let an AI agent write and run code on your computer. That agent can:

- Install software packages
- Download files from the internet
- Create, modify, and delete files
- Run arbitrary shell commands

Letting an AI do all that directly on your laptop is risky. It might accidentally delete important files, install conflicting software, or fill up your hard drive. And if something goes wrong, cleaning up the mess can be painful.

1.2 The Solution: A Sandbox

Docker lets you create an isolated "mini-computer" (called a **container**) that runs inside your real computer. Think of it as a virtual room where the AI agent can work freely. If anything goes wrong, you simply delete the container and start fresh — your actual system remains untouched.

1.3 Three Concepts You Need to Know

Concept	Analogy	What It Is
Dockerfile	A recipe	A text file with instructions: "Install R, install Python, install Claude Code, set up these folders…"
Image	A frozen meal	The result of following the recipe — a snapshot of a complete environment, ready to use
Container	The meal, heated up and on your plate	A running instance of the image where you actually work

1.4 Why Docker for This Workshop?

Benefit	What It Means for You
Safety (sandbox)	The AI agent cannot touch your files, your system, or your other software. Everything stays inside the container.
Reproducibility	Every student gets the exact same environment. No more "it works on my machine" problems.
Easy cleanup	Done with the project? Delete the container and image. Your system is clean again.
Portability	Share the Dockerfile with a colleague and they can rebuild the exact same setup in minutes.

---

2 Part 2: Installing Docker

2.1 Option A: Docker Desktop (Recommended for Beginners)

Docker Desktop is a free application that lets you run containers on Windows and Mac (and Linux, though most Linux users prefer Docker Engine directly). It provides a graphical interface and handles all the technical complexity behind the scenes.

Good news for students and educators

Docker Desktop is **free for educational use**. Docker's subscription terms explicitly exempt educational institutions, so you can use Docker Desktop at university without any licensing concerns.

Windows

2.2 Windows (with Docker Desktop)

Prerequisites: Windows 10 version 2004 or higher, or Windows 11. You need administrator access.

Step 1: Enable WSL 2 (Windows Subsystem for Linux)

WSL 2 lets Docker run a lightweight Linux kernel inside Windows. This is required.

1. Open **PowerShell as Administrator** (right-click the Start button, select "Terminal (Admin)" or "Windows PowerShell (Admin)")
2. Run:

```
wsl --install
```

3. **Restart your computer** when prompted
4. After restart, a window will open asking you to create a Linux username and password. Choose something simple you will remember (e.g., your first name as username). **This is NOT your Windows password** — it is a new, separate password for the Linux subsystem.

Expected time: 5–10 minutes (including restart)

Step 2: Install Docker Desktop

1. Go to: https://www.docker.com/products/docker-desktop/
2. Click **"Download for Windows"**
3. Run the downloaded installer (`Docker Desktop Installer.exe`)
4. Keep all default settings and click through the installer
5. When installation finishes, **restart your computer** if prompted
6. Docker Desktop should start automatically after login. You will see a whale icon in your system tray (bottom-right).

Expected time: 5–10 minutes (including download)

Step 3: Configure Docker Desktop

1. Open Docker Desktop (click the whale icon in the system tray)
2. If you see a "Service Agreement" screen, click **Accept**
3. You can skip or close the tutorial/sign-in prompts — you do NOT need a Docker account
4. Go to **Settings** (gear icon, top-right):
   - **General** tab: Make sure **"Use the WSL 2 based engine"** is checked (it usually is by default)
   - **Resources → WSL Integration** tab: Toggle ON the switch next to your Ubuntu distribution
   - Click **"Apply & restart"**

Step 4: Verify the Installation

Open **Ubuntu** from your Start menu (NOT PowerShell, NOT CMD — use the Ubuntu app that WSL installed):

```
docker --version
```

You should see something like `Docker version 27.x.x`. Now test that Docker can run containers:

```
docker run hello-world
```

You should see a message starting with `Hello from Docker!`. If so, Docker is working.

From now on, **always work in the Ubuntu/WSL terminal** for all Docker commands. Do not use PowerShell or CMD.

Mac

2.3 Mac

Prerequisites: macOS 14 (Sonoma) or later. You need administrator access. Check whether your Mac has an **Apple Silicon** chip (M1, M2, M3, M4) or an **Intel** chip: click the Apple menu, select "About This Mac", and look at the "Chip" line.

Step 1: Install Docker Desktop

1. Go to: https://www.docker.com/products/docker-desktop/
2. Click **"Download for Mac"**
   - If you have Apple Silicon (M1/M2/M3/M4): select **"Apple Silicon"** (this is the default now)
   - If you have Intel: select **"Intel chip"**
3. Open the downloaded `.dmg` file
4. Drag the Docker icon into your Applications folder
5. Open Docker from your Applications folder (or Spotlight: Cmd+Space, type "Docker")
6. macOS will ask for your password to grant permissions — enter it
7. If you see a "Service Agreement" screen, click **Accept**
8. You can skip/close the tutorial and sign-in prompts — you do NOT need a Docker account

Expected time: 5–10 minutes (including download)

Step 2: Verify the Installation

Open **Terminal** (Spotlight: Cmd+Space, type "Terminal"):

```
docker --version
```

You should see something like `Docker version 27.x.x`. Now test:

```
docker run hello-world
```

You should see `Hello from Docker!`. If so, Docker is working.

2.4 Adjusting Docker Resources (Docker Desktop)

By default, Docker Desktop may allocate limited resources to containers. For running AI agents with R and Python, you should increase these limits:

1. Open **Docker Desktop → Settings → Resources**
2. Set:
   - **CPUs:** At least 4 (or half of your total cores)
   - **Memory:** At least 8 GB (ideally 12–16 GB)
   - **Disk image size:** At least 60 GB
3. Click **"Apply & restart"**

> **Note**
>
> These are *upper limits* — Docker will not use these resources unless a container needs them. Your other applications will still run normally.

2.5 Option B: Docker Engine in WSL 2 Without Docker Desktop (Windows Only)

If you prefer a **lightweight, CLI-only** setup — or if you want to avoid Docker Desktop entirely — you can install Docker Engine directly inside a WSL 2 Linux distribution. This is the most common Docker Desktop-free approach on Windows. You get the full `docker` and `docker compose` commands, but without the graphical interface.

When to choose this option

- You are comfortable working in a terminal and do not need a GUI
- You want a smaller footprint (no background Desktop app running)
- You are on a shared or restricted machine where installing Docker Desktop is not possible

Trade-offs: You lose the Docker Desktop GUI and some convenience features (like automatic WSL integration across multiple distros, one-click resource settings, and the visual container dashboard). Functionally, however, everything works the same.

Step 1: Enable and Install WSL 2

If you have not already installed WSL 2, open **PowerShell as Administrator** and run:

```
wsl --install
```

Restart your computer when prompted. After restart, a window will ask you to create a Linux username and password.

Expected time: 5–10 minutes (including restart)

Step 2: Open Your WSL 2 Ubuntu Terminal

Open **Ubuntu** from the Start menu. All the following commands are run inside this Linux terminal.

Step 3: Install Docker Engine

Follow the official Docker Engine installation steps for Ubuntu. Run these commands one by one:

```
# Update package index and install prerequisites
sudo apt-get update
sudo apt-get install -y ca-certificates curl

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add Docker's repository
echo \
  "deb [arch=$(dpkg --print-architecture)
    ↪  signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine and Docker Compose plugin
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
```

Expected time: 5–10 minutes

Step 4: Start the Docker Daemon

WSL 2 does not use `systemd` by default, so Docker does not start automatically. You need to start it manually:

```
sudo service docker start
```

You will need to run this command every time you open a new WSL terminal session (or after a reboot). To make it start automatically, add the following line to your `~/.bashrc` file:

```
echo 'sudo service docker start' >> ~/.bashrc
```

> **Avoid the sudo password prompt**
>
> To avoid the `sudo` password prompt every time Docker starts, you can add your user to the `docker` group:
>
> ```
> sudo usermod -aG docker $USER
> ```
>
> Then close and re-open your Ubuntu terminal for the change to take effect. After that, you can run `docker` commands without `sudo`.

Step 5: Verify the Installation

```
docker --version
docker run hello-world
```

You should see `Docker version 27.x.x` and then `Hello from Docker!`. If so, Docker Engine is working inside your WSL 2 environment.

> **Note**
>
> **From here, everything else in this guide works the same** — all `docker buildx build`, `docker compose`, and `docker run` commands work identically whether you use Docker Desktop or Docker Engine in WSL 2.

2.6 Option C: Docker Engine on Linux (Ubuntu)

If you are running Ubuntu (or another Debian-based distribution) as your main operating system — not inside WSL, but as a native Linux desktop or server — you can install Docker Engine directly. No Docker Desktop or WSL needed.

> **When to choose this option**
>
> - You are running Ubuntu 22.04, 24.04, or a similar Debian-based distribution as your primary OS
> - You are comfortable with the terminal
> - You want the most lightweight and direct Docker setup possible
>
> **Trade-offs:** No graphical dashboard. Resource limits are managed via `docker-compose.yml` or command-line flags rather than a GUI.

Step 1: Remove old Docker packages (if any)

If you have previously installed Docker from Ubuntu's default repositories, remove them first to avoid conflicts:

```
sudo apt-get remove -y docker docker-engine docker.io containerd runc
  ↪  2>/dev/null
```

This is safe to run even if none of these packages are installed.

Step 2: Set up Docker's official repository

```
# Update package index and install prerequisites
sudo apt-get update
sudo apt-get install -y ca-certificates curl

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
# Add Docker's repository
echo \
  "deb [arch=$(dpkg --print-architecture)
    ↪  signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Step 3: Install Docker Engine

```
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
```

> **Expected time:** 3–5 minutes

Step 4: Allow your user to run Docker without sudo

```
sudo usermod -aG docker $USER
```

Log out and log back in (or reboot) for this to take effect. After that, you can run `docker` commands without `sudo`.

Step 5: Verify the Installation

```
docker --version
docker run hello-world
```

You should see `Docker version 27.x.x` and then `Hello from Docker!`.

> **Note**
>
> On a native Linux system, Docker Engine starts automatically via `systemd`. You do **not** need to start it manually (unlike in WSL 2). If it is not running, use:
>
> ```
> sudo systemctl start docker
> sudo systemctl enable docker
> ```

> **Note**
>
> **From here, everything else in this guide works the same.** All `docker buildx build`, `docker compose`, and `docker run` commands work identically regardless of your operating system.

---

3 Part 3: Understanding the Dockerfile

Before building our container, let us look at what goes inside the Dockerfile. You do not have to write this yourself — we provide a ready-made Dockerfile. But understanding it helps you know what is happening.

A Dockerfile is just a text file called `Dockerfile` (no extension) that lists instructions. Each instruction adds a "layer" to the image. Here is a simplified overview of what our Dockerfile does:

```
1. Start from a base Linux image (Ubuntu 24.04)
2. Install system tools (curl, wget, git, build tools, LaTeX)
3. Install R and R packages
4. Install Python and Python packages
```

5. Install Node.js (needed for Codex and Gemini CLI)
6. Install Claude Code
7. Install OpenAI Codex
8. Install Google Gemini CLI
9. Set up working directories

3.1 Installing R Inside the Container

Our Dockerfile adds the official CRAN repository for Ubuntu to get the latest stable R version, then installs R and common packages:

```
# Add the CRAN repository for the latest R version
RUN wget -qO-
  ↪  https://cloud.r-project.org/bin/linux/ubuntu/marutter_pubkey.asc |
  ↪  \
        tee -a /etc/apt/trusted.gpg.d/cran_ubuntu_key.asc && \
    add-apt-repository \
       "deb https://cloud.r-project.org/bin/linux/ubuntu noble-cran40/"

# Install R and system libraries needed by R packages
RUN apt-get update && apt-get install -y r-base r-base-dev

# Install R packages
RUN Rscript -e "install.packages(c('tidyverse', 'fixest',
  ↪  'modelsummary', \
    'haven', 'data.table', 'rmarkdown'), \
    repos='https://cloud.r-project.org')"
```

> **Tip**
>
> Installing many R packages takes time. Our Dockerfile pre-installs ~80 econometrics packages so you do not have to wait during the workshop.

3.2 Installing Python Inside the Container

```
# Example: Installing Python with a virtual environment
RUN apt-get install -y python3 python3-pip python3-venv
RUN python3 -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Install Python packages
RUN pip install numpy pandas matplotlib jupyter
```

3.3 Installing Stata Inside the Container (Special Case)

Unlike R and Python, **Stata is commercial software**. You cannot freely download and install it. There are three options:

Option A: Use your university's Stata license (recommended if available)

If your university provides a Stata license, you can install it inside the container. You will need:

1. The Stata installer file (e.g., `Stata18Linux64.tar.gz`) — ask your IT department
2. Your license information (serial number, code, authorization)

```
# Copy the Stata installer into the container
COPY Stata18Linux64.tar.gz /tmp/
RUN cd /tmp && tar -xzf Stata18Linux64.tar.gz && \
    cd /tmp/Stata18 && ./install
```

```
# You will need to initialize the license separately after the
  ↪  container starts:
# Inside the container, run: stata -q -e "exit"
# and enter your license details when prompted
```

> **!** Important
>
> Stata for Linux is different from Stata for Windows/Mac. Even if you have Stata on your laptop, you need a **Linux version** for the Docker container. Some university licenses include all platforms.

Option B: Mount Stata from your host system

If you have Stata installed on your Mac/Windows machine, you can "share" it into the container using a Docker volume. This is more complex and depends on your specific Stata installation path.

Option C: Skip Stata and use R equivalents

For many econometric analyses, R packages like `fixest`, `plm`, `ivreg`, and `rdrobust` can replicate Stata functionality. The workshop's CLAUDE.md includes a complete Stata-to-R mapping table for this purpose. This is the easiest option if you do not have a Linux Stata license.

3.4 Installing Node.js (Required for Codex and Gemini CLI)

OpenAI Codex and Google Gemini CLI require Node.js. Claude Code uses a native binary installer and does not depend on Node.js.

```
# Install Node.js 24 LTS
RUN curl -fsSL
  ↪  https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | \
      gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg && \
    echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] \
      https://deb.nodesource.com/node_24.x nodistro main" | \
      tee /etc/apt/sources.list.d/nodesource.list > /dev/null && \
    apt-get update && apt-get install -y nodejs
```

---

4 Part 4: Building the Docker Image

Now we will actually build the image. This is where Docker reads the Dockerfile and creates the frozen environment.

Step 1: Download the workshop files

Open your terminal (Ubuntu on Windows, Terminal on Mac/Linux) and download or clone the workshop repository:

```
# Navigate to your home directory
cd ~

# Clone the workshop repository
git clone https://github.com/AlexRieber/Workshops.git

# Enter the Docker workshop directory
cd Workshops/Docker
```

> **Tip**
>
> If you received the files as a ZIP, unzip them and navigate to the folder using `cd`.

Step 2: Build the image

```
docker buildx build -t coding-agent .
```

Breaking this command down:

Part	Meaning
`docker buildx build`	Tell Docker to build an image
`-t coding-agent`	Name the image `coding-agent` (you can choose any name)
`.`	Look for the Dockerfile in the current directory

> **Build time: 20–30 minutes**
>
> The first build takes 20–30 minutes. Docker has to download the base image, install all system packages, compile R packages, etc.
> **Good news:** Docker *caches* every step. If you change only the CLAUDE.md file (for example) and rebuild, it takes seconds because Docker reuses the cached layers for everything else.

What to expect during the build:

- You will see lots of text scrolling by — that is normal
- Lines starting with `#6 [3/12] RUN apt-get...` show progress through the build steps
- Warnings about deprecated package versions are usually harmless
- If the build fails, read the last few lines of error output — they usually tell you what went wrong

Step 3: Verify the image was created

```
docker images
```

You should see `coding-agent` in the list with a size of several gigabytes.

---

5 Part 5: Running the Container

5.1 Starting the Container (Simple Version)

For a quick start, run:

```
docker run -it \
  --name my-agent \
  -v "$(pwd)/workspace":/home/agent/project \
  -v "$(pwd)/output":/home/agent/output \
  coding-agent \
  bash
```

This command:

Part	What It Does
`docker run -it`	Start a new container in interactive mode (you can type commands)
`--name my-agent`	Give the container a name so you can refer to it later
`-v "$(pwd)/workspace":/home/agent/project`	Share the `workspace/` folder between your computer and the container
`-v "$(pwd)/output":/home/agent/output`	Share the `output/` folder for results

Part	What It Does
`coding-agent`	Use the image we built
`bash`	Open a bash shell inside the container

You are now *inside* the container. Your prompt changes to something like `root@a1b2c3d4:/home/agent#`.

> **Tip**
>
> The `workspace/` and `output/` folders are shared between the container and your computer. Anything the AI agent saves there, you can see on your host system. Everything else inside the container is isolated.

5.2 Starting the Container (Recommended: with Resource Limits)

For a more controlled setup, use `docker compose`. Create a file called `docker-compose.yml` in your project directory:

```yaml
services:
  agent:
    image: coding-agent
    container_name: my-agent
    stdin_open: true
    tty: true
    mem_limit: 16g
    memswap_limit: 16g
    cpus: 4
    pids_limit: 512
    volumes:
      - ./workspace:/home/agent/project
      - ./output:/home/agent/output
      - ./CLAUDE.md:/home/agent/CLAUDE.md:ro
      - agent-auth:/root/.claude
    working_dir: /home/agent

volumes:
  agent-auth:
```

Then run:

```bash
# Create the shared folders if they do not exist
mkdir -p workspace output

# Start the container in the background
docker compose up -d

# Enter the running container
docker exec -it my-agent bash
```

5.3 What Are Resource Limits and Why?

Setting	Value	Purpose
`mem_limit: 16g`	Max 16 GB RAM	Prevents the agent from using all your computer's memory
`memswap_limit: 16g`	No swap allowed	Prevents slow-down from excessive swapping

Setting	Value	Purpose
`cpus: 4`	Max 4 CPU cores	Leaves CPU capacity for your other applications
`pids_limit: 512`	Max 512 processes	Prevents runaway process creation (fork bombs)

5.4 Exiting and Re-entering the Container

```
# To leave the container (it keeps running):
exit

# To re-enter a running container:
docker exec -it my-agent bash

# To stop the container:
docker compose down

# To start it again later:
docker compose up -d
```

---

6 Part 6: Setting Up Claude Code

Claude Code is Anthropic's terminal-based AI coding agent. It reads files, writes code, runs commands, and iterates on errors — all from the command line. There are two ways to authenticate.

6.1 Option A: Anthropic API Key (Pay-per-use)

This option charges you per token (per amount of text processed). You control your spending through the API dashboard.

Step 1: Get an API key

1. Go to: **https://platform.claude.com/**
2. Create an account (or sign in)
3. Go to **"API Keys"** in the left sidebar
4. Click **"Create Key"**, give it a name (e.g., "workshop"), and copy the key
5. **Important:** The key starts with `sk-ant-...`. Save it somewhere safe — you will not be able to see it again.
6. Add credit to your account under **"Billing"** (even $5–10 is enough for a workshop session)

Step 2: Install Claude Code inside the container

Our Dockerfile already includes Claude Code. If you need to install it manually:

```
# Inside the container:
curl -fsSL https://claude.ai/install.sh | bash
```

> **Expected time:** 1–2 minutes

Step 3: Authenticate with the API key

```
# Inside the container, set the API key:
export ANTHROPIC_API_KEY="sk-ant-your-key-here"

# Start Claude Code:
claude
```

Claude Code will detect the API key and start immediately.

> **Tip**
>
> To avoid re-entering the key every time, add it to the container's shell profile:
>
> ```
> echo 'export ANTHROPIC_API_KEY="sk-ant-your-key-here"' >> ~/.bashrc
> ```

6.2 Option B: Claude Max Subscription (Fixed Monthly Cost)

If you have a Claude Pro ($20/month) or Max ($100 or $200/month) subscription, you can use Claude Code without API charges. Usage counts against your subscription's rate limits.

Step 1: Get a subscription

1. Go to: **https://claude.ai/**
2. Sign up or sign in
3. Upgrade to Pro or Max under account settings

Step 2: Authenticate inside the container

```
# Inside the container:
claude
```

Claude Code will display a message like:

```
To authenticate, visit:
https://claude.ai/login?code=ABCD-1234-EFGH
```

1. **Copy** the URL from the terminal
2. **Open** the URL in a web browser on your host machine (your laptop)
3. Sign in with your Claude account
4. Click **"Authorize"**
5. Back in the terminal, you should see: `Authenticated successfully!`

> **Note**
>
> **The login is saved.** If you used `docker-compose.yml` with the `agent-auth` volume (as shown above), your login persists even if you stop, remove, and recreate the container. You only need to log in once.

6.3 Verifying Claude Code Works

After authentication (either method), test it:

```
claude --version
```

Then start an interactive session:

```
claude
```

You should see the Claude Code interface. Type a simple test message like `What is 2+2?` and press Enter. If you get a response, everything is working.

6.4 Running Claude Code with Specific Models

```
# Use the default model (Sonnet):
claude

# Use the most capable model (Opus --- requires Max $200 or API):
claude --model opus
```

```
# See real-time thinking (recommended for learning):
claude --verbose
```

---

7 Part 7: Setting Up OpenAI Codex

OpenAI Codex is OpenAI's terminal-based AI coding agent, similar in concept to Claude Code. It is open source (Apache 2.0 license) and available on GitHub at https://github.com/openai/codex.

7.1 Authentication Options for Codex

Like Claude Code, Codex supports two authentication methods:

Method	What You Need	Cost Model
OpenAI API Key	API key from platform.openai.com	Pay-per-token
ChatGPT Subscription	ChatGPT Plus, Pro, or Enterprise plan	Included in subscription

> **!** Important
>
> The ChatGPT subscription login uses a browser-based OAuth flow that opens a local web server on port 1455. This is difficult to use from inside a Docker container. For Docker setups, **we recommend using an API key**.

7.2 Step 1: Get an OpenAI API Key

1. Go to: **https://platform.openai.com/**
2. Create an account (or sign in)
3. Go to **"API Keys"** in the sidebar
4. Click **"Create new secret key"**, give it a name, and copy the key
5. The key starts with `sk-...`. Save it somewhere safe.
6. Add credit under **"Billing"** (again, $5–10 is plenty for experimenting)

7.3 Step 2: Install Codex Inside the Container

Our Dockerfile already includes Codex. If you need to install it manually:

```
# Inside the container:
npm install -g @openai/codex
```

> **Expected time:** 1–2 minutes

7.4 Step 3: Authenticate and Run

```
# Set the API key:
export OPENAI_API_KEY="sk-your-key-here"

# Start Codex interactively:
codex
```

For **non-interactive** use (e.g., in scripts), Codex provides the `exec` subcommand:

```
codex exec "analyze the data in data.csv and create a summary"
```

7.5 Step 4: Useful Codex Flags

Flag	What It Does
`--full-auto`	Agent applies changes without asking for approval
`--sandbox danger-full-access`	Disable sandboxing (appropriate when already in Docker)
`--model <name>`	Choose which OpenAI model to use (e.g., `o3-mini`)

> **Note**
>
> **Note on sandboxing:** Codex has its own built-in sandbox (using Linux Landlock on Linux). Since we are already running inside a Docker container (which is itself a sandbox), you can safely use `--sandbox danger-full-access` to avoid sandboxing conflicts. The Docker container provides the isolation.

---

8 Part 8: Setting Up Google Gemini CLI

Google Gemini CLI is Google's open-source terminal-based AI coding agent (Apache 2.0 license), available on GitHub at https://github.com/google-gemini/gemini-cli. It provides access to Gemini models with a 1M token context window.

8.1 Authentication Options for Gemini CLI

Method	What You Need	Cost Model
Google AI Studio API Key	API key from aistudio.google.com	Free tier (limits vary by model)
Google Account Login	Personal Google account	Free (same limits as above)
Vertex AI	Google Cloud project with billing	Pay-per-token

8.2 Option A: Google AI Studio API Key (Recommended)

Step 1: Get an API key

1. Go to: **https://aistudio.google.com/**
2. Sign in with your Google account
3. Click **"Get API key"** → **"Create API key"**
4. Copy the key and save it somewhere safe

> **Tip**
>
> The free tier is generous for the default model (Gemini Flash). Limits vary by model — more capable models like Gemini Pro have lower quotas. Either way, the free tier is more than enough for a workshop session without spending any money.

Step 2: Install Gemini CLI inside the container

Our Dockerfile already includes Gemini CLI. If you need to install it manually:

```
# Inside the container:
npm install -g @google/gemini-cli
```

> **Expected time:** 1–2 minutes

Step 3: Authenticate with the API key

```
# Inside the container, set the API key:
export GEMINI_API_KEY="your-key-here"

# Start Gemini CLI:
gemini
```

8.3 Option B: Google Account Login (No API Key Needed)

If you prefer not to create an API key, Gemini CLI can authenticate directly with your Google account:

```
# Inside the container:
gemini
```

On first run, Gemini CLI will display a URL. Open it in your browser, sign in with your Google account, and authorize the application. The same free tier limits apply.

> **!** Important
>
> The browser login flow requires network access from the container. If you are on a restricted network, use the API key method instead.

8.4 Verifying Gemini CLI Works

```
gemini --version
```

Then start an interactive session:

```
gemini
```

Type a test message. If you get a response, everything is working.

---

9 Part 9: The Complete Dockerfile

The complete Dockerfile is available in the workshop repository:

`Dockerfile` on GitHub

If you cloned the repository (see Part 4), you already have this file. Otherwise, download it directly and place it in your project directory as `Dockerfile` (no file extension).

Here is what each section of the Dockerfile does:

Layer	What It Does
Base image (`FROM ubuntu:24.04`)	Starts from a clean Ubuntu 24.04 LTS Linux system
System tools	Installs `curl`, `wget`, `git`, `nano`, `jq`, `tree`, build tools, and LaTeX (for rendering R Markdown / Quarto documents)

Layer	What It Does
R + R packages	Adds the CRAN repository for the latest stable R, then installs R plus ~25 core econometrics and data science packages (tidyverse, fixest, modelsummary, haven, etc.). This is the slowest step (~10–15 min).
Python + packages	Installs Python 3 in a virtual environment with `numpy`, `pandas`, `matplotlib`, and `jupyter`
AI provider SDKs	Installs the `openai`, `anthropic`, and `google-genai` Python SDKs for building custom agents
Aider	Open-source AI coding agent with built-in OpenRouter support (use any model)
Agent utilities	`rich` (terminal formatting), `prompt_toolkit` (interactive input), `pydantic` (structured data)
Node.js 24 LTS	Required runtime for OpenAI Codex and Google Gemini CLI
Claude Code	Anthropic's terminal-based coding agent (native binary, does not require Node.js)
OpenAI Codex	OpenAI's terminal-based coding agent
Google Gemini CLI	Google's terminal-based coding agent
Working directories	Creates the folder structure (`project/`, `output/`, `data/`, `code/`)
CMD	Sets `bash` as the default command when the container starts

Customizing the Dockerfile for your needs

The provided Dockerfile is a starting point. You will likely want to adapt it to your own research workflow — for example, adding specific R or Python packages, additional system libraries, or different tools.

Ask your preferred LLM to help you. Copy the Dockerfile into a chat with ChatGPT, Claude, Gemini, or whichever model you prefer and describe what you need:

- *"I need to add the `sf` and `terra` R packages for geospatial analysis. What system libraries do I need and how should I modify this Dockerfile?"*
- *"I want to add Julia to this container. What lines should I add?"*
- *"I work with LaTeX documents using Biblatex. What do I need to add?"*

LLMs are excellent at writing and debugging Dockerfiles. They can suggest the right `apt-get` packages, handle dependency chains, and help you keep the image size manageable. This is a great way to learn Docker syntax while getting a setup tailored to your needs.

Build time expectations

Step	Approximate Time
Download base image	1–2 min
System tools + LaTeX	3–5 min
R + R packages	10–15 min
Python + packages	2–3 min
Node.js + Claude Code + Codex + Gemini CLI	3–5 min
Total (first build)	**~20–30 min**
Rebuild after changing only CLAUDE.md	**< 10 seconds**

10 Part 10: Putting It All Together (Full Walkthrough)

Here is the complete process from zero to a running agent. Follow these steps in order.

10.1 1. Install Docker (~10 min)

- **Windows:** Follow Part 2, Option A (install WSL 2, then Docker Desktop) or Option B (Docker Engine in WSL 2)
- **Mac:** Follow Part 2, Option A (download and install Docker Desktop)
- **Linux:** Follow Part 2, Option C (install Docker Engine directly)
- Verify with `docker run hello-world`

10.2 2. Create Your Project Folder (~2 min)

Open your terminal (Ubuntu on Windows, Terminal on Mac/Linux):

```
mkdir -p ~/coding-agent-workshop
cd ~/coding-agent-workshop
mkdir -p workspace output
```

10.3 3. Get the Dockerfile (~2 min)

If you cloned the repository in step 1 of Part 4, the `Dockerfile` is already included. Otherwise, download it from the workshop repository on GitHub and save it as `Dockerfile` (no extension) in `~/coding-agent-workshop/`. On the command line, you can open the folder:

Windows

Open the folder in Explorer:

```
explorer.exe $(wslpath -w
  ↪  ~/coding-agent-workshop)
```

Mac

Open the folder in Finder:

```
open ~/coding-agent-workshop
```

Then place the `Dockerfile` (no extension!) in this folder.

10.4 4. Create the docker-compose.yml (~1 min)

Create `docker-compose.yml` in the same folder with the content from Part 5.

10.5 5. Build the Image (~20–30 min)

```
cd ~/coding-agent-workshop
docker buildx build -t coding-agent .
```

Go get a coffee. This will take a while the first time.

10.6 6. Start the Container (~10 sec)

```
docker compose up -d
docker exec -it my-agent bash
```

You are now inside the container.

10.7 7. Verify Everything Is Installed (~1 min)

Inside the container, run:

```
# Check R
R --version | head -1

# Check Python
python3 --version

# Check Node.js
node --version

# Check Claude Code
claude --version

# Check Codex
codex --version

# Check Gemini CLI
gemini --version

# Check Aider
aider --version
```

All commands should print a version number.

10.8 8. Authenticate Your AI Agent (~2 min)

For Claude Code:

```
# Option A: API key
export ANTHROPIC_API_KEY="sk-ant-your-key-here"
claude

# Option B: Max subscription
claude
# Follow the browser login flow (see Part 6)
```

For OpenAI Codex:

```
export OPENAI_API_KEY="sk-your-key-here"
codex
```

For Google Gemini CLI:

```
export GEMINI_API_KEY="your-key-here"
gemini
```

10.9 9. Start Working

You are ready! Give the agent a task:

```
# Claude Code (interactive):
claude

# Or with verbose output to see its reasoning:
claude --verbose
```

---

11 Part 11: Everyday Commands (Cheat Sheet)

11.1 Starting and Stopping

```
# Start the container (from your project folder)
docker compose up -d

# Enter the container
docker exec -it my-agent bash

# Leave the container (it keeps running)
exit

# Stop the container
docker compose down
```

11.2 Inside the Container

```
# Start Claude Code
claude

# Start Claude Code with Opus model and verbose output
claude --model opus --verbose

# Start OpenAI Codex
codex

# Start Google Gemini CLI
gemini

# Start Aider with an OpenRouter model
OPENROUTER_API_KEY="sk-or-..." aider --model
  ↪  openrouter/moonshotai/kimi-k2

# Run an R script
Rscript my_analysis.R

# Start an interactive R session
R

# Start Python
python3
```

11.3 Managing Docker (on your host system)

```
# See running containers
docker ps

# See all images
docker images

# See resource usage of a running container
docker stats my-agent

# Rebuild the image after changing the Dockerfile
docker buildx build -t coding-agent .

# Remove the container (data in workspace/ and output/ is preserved)
docker rm -f my-agent
```

```
# Remove the image (frees disk space)
docker rmi coding-agent

# Nuclear option: remove everything Docker-related (containers, images,
  ↪  volumes)
# WARNING: This deletes ALL Docker data, including saved authentication!
docker system prune -a --volumes
```

---

12 Part 12: Troubleshooting

12.1 Installation Problems

Problem	Solution
Windows: "WSL 2 installation is incomplete"	Open PowerShell as Admin, run `wsl --update`, restart
Windows: "Hardware assisted virtualization is not enabled"	Enable VT-x/AMD-V in your BIOS settings (search for your laptop model + "enable virtualization")
Windows: Docker Desktop does not start	Run `wsl --status` in PowerShell. If WSL is not running, try `wsl --shutdown` then restart Docker Desktop
Mac: "Docker Desktop requires macOS 14 or later"	Update your macOS, or try an older version of Docker Desktop
`docker: command not found`	Docker Desktop is not running, or the PATH is not set. Restart Docker Desktop. On Windows, make sure you are using the Ubuntu terminal.

12.2 Build Problems

Problem	Solution
Build fails at R package installation	Some R packages need system libraries. Check the error message for missing `-dev` packages and add them to the `apt-get install` line in the Dockerfile
Build runs out of disk space	Increase Docker Desktop's disk image size (Settings → Resources) and run `docker system prune` to free space
Build is very slow	First builds are always slow. Subsequent rebuilds use cached layers and are much faster
"network error" during build	Check your internet connection. Some corporate/university networks block Docker Hub. Try a different network or VPN.

12.3 Runtime Problems

Problem	Solution
Container immediately exits	Use `docker run -it ... bash` (the `-it` flags keep it interactive)
`claude`, `codex`, `gemini`, or `aider` command not found	Rebuild the image: `docker buildx build -t coding-agent .`
Claude Code: "Authentication required"	Follow the login steps in Part 6. For API key: check that `ANTHROPIC_API_KEY` is set (`echo $ANTHROPIC_API_KEY`)
Codex: "API key not found"	Check that `OPENAI_API_KEY` is set (`echo $OPENAI_API_KEY`)
Gemini CLI: authentication error	Follow login steps in Part 8. For API key: check that `GEMINI_API_KEY` is set (`echo $GEMINI_API_KEY`)
"No space left on device" in container	The container's disk is full. Remove unnecessary files, or increase Docker's disk image size
Everything is very slow	Check Docker Desktop resource settings (Part 2). Give Docker more RAM and CPUs. On Windows: make sure files are in the WSL filesystem (`~/...`), not in `/mnt/c/...`
Lost authentication after container restart	Make sure you are using the `agent-auth` volume in `docker-compose.yml` (see Part 5). Without it, auth is lost when the container is removed.

---

13 Part 13: Security and Isolation

Running an AI coding agent means giving software the ability to read files, write files, run commands, and access the internet. This section explains how Docker keeps you safe, and how to add extra layers of protection.

13.1 What the Container CAN Do

- Read and write files inside the container
- Access files in the mounted `workspace/` and `output/` folders
- Make network requests (download data, call APIs)
- Install software inside the container

13.2 What the Container CANNOT Do

- Access files on your host system outside of the mounted folders
- Modify your operating system or installed software
- Access other running applications on your computer
- Persist changes (except in mounted volumes) after being deleted

13.3 Best Practices

1. **Never put sensitive files** (passwords, private keys, personal data) in the `workspace/` or `output/` folders
2. **Review output** before sharing it — AI agents can sometimes include unexpected content
3. **Use resource limits** (as shown in the `docker-compose.yml`) to prevent runaway resource usage
4. **Store API keys carefully** — do not commit them to Git repositories. Use environment variables.
5. **Regularly stop containers** you are not using — they consume system resources even when idle

---

14 Part 14: Advanced Safety — Isolating Docker on Its Own Partition

14.1 Why a Separate Partition?

By default, Docker stores all its data (images, containers, volumes) on your main system drive. An autonomous AI agent could, in theory:

- Download very large datasets, filling up your disk
- Create many intermediate files during analysis
- Pull additional Docker images

If your main drive runs out of space, your **entire system** can become unstable — not just the container. The solution is to give Docker its own, separate storage area with a hard size limit. If that area fills up, Docker stops working but your operating system and personal files stay safe.

Think of it like giving someone a dedicated filing cabinet rather than access to your entire office. When the cabinet is full, they cannot fill up anything else.

14.2 Option A: Move Docker Desktop's Storage (Recommended — Easiest)

This is the simplest approach and works well for most students.

Windows

14.2.1 Windows

Docker Desktop on Windows stores its data inside WSL 2. You can move this to a different drive:

Step 1: Stop Docker Desktop (right-click the whale icon in the system tray, select "Quit Docker Desktop")

Step 2: Open PowerShell as Administrator:

```
# Stop WSL
wsl --shutdown

# Export the Docker data distribution to your target drive
wsl --export docker-desktop-data
  ↪  D:\DockerData\docker-desktop-data.tar

# Unregister the original (this removes it from your C: drive)
wsl --unregister docker-desktop-data

# Import it at the new location with a size you control
wsl --import docker-desktop-data D:\DockerData
  ↪  D:\DockerData\docker-desktop-data.tar

# Clean up the export file
del D:\DockerData\docker-desktop-data.tar
```

Expected time: 5–15 minutes depending on how much Docker data you already have.

Replace `D:\DockerData` with any folder on a drive with enough space. If you have an external SSD or a second internal drive, use that.

Recent versions of Docker Desktop (4.30+) may no longer create a separate `docker-desktop-data` WSL distribution. If the `wsl --export docker-desktop-data` command fails, use Docker Desktop's built-in setting instead: **Settings → Resources → Disk image location** and move the storage from there.

Step 3: Restart Docker Desktop. All Docker operations now use the new location.

Mac

14.2.2 Mac

On Mac, Docker Desktop stores its data in a single disk image file. You can control its location and size:

1. Open **Docker Desktop → Settings → Resources → Disk image location**
2. Click **"Browse"** and select a folder on a different volume or external drive
3. Set the **"Disk image size"** to your desired limit (e.g., 60 GB)
4. Click **"Apply & restart"**

Docker will move its data to the new location. This may take a few minutes.

14.3 Option B: Create a Dedicated Disk Image (Advanced — Maximum Isolation)

This approach creates a fixed-size virtual disk that Docker uses. When it fills up, Docker hits a wall but your system stays safe. This is ideal for running AI agents that you want to constrain strictly.

Windows

14.3.1 Windows (WSL 2)

On Windows, Docker runs inside WSL 2, which already uses a virtual disk (`.vhdx` file). You can control its maximum size:

Step 1: Stop Docker Desktop and WSL:

```
wsl --shutdown
```

Step 2: Create a `.wslconfig` file in your Windows home directory (`C:\Users\YourName\.wslconfig`) with:

```
[wsl2]
# Limit WSL 2 resource usage
memory=8GB
processors=4
swap=0
```

This `.wslconfig` file limits WSL 2's memory and CPU usage but does **not** directly cap the virtual disk size. The virtual disk (`ext4.vhdx`) grows dynamically up to 256 GB by default. To shrink an existing one, you need to use `diskpart` in PowerShell — this is more complex and not necessary for a fresh setup. For hard disk limits on Windows, use Option A (move Docker storage to a drive with limited free space) instead.

Step 3: Restart WSL and Docker Desktop.

Mac

14.3.2 Mac

On Mac, you can create a dedicated APFS or HFS+ disk image:

Step 1: Open **Disk Utility** (Spotlight: Cmd+Space, type "Disk Utility")

Step 2: Go to **File → New Image → Blank Image** and configure:

Setting	Value
Name	DockerWorkspace
Size	60 GB (or your preferred limit)
Format	APFS
Partitions	Single partition - GUID

Image Format	sparse disk image (grows as needed, up to the size limit)

Step 3: Click **"Save"**. The disk image (`.sparseimage`) will be created.

Expected time: < 1 minute

Step 4: Double-click the `.sparseimage` file to mount it. It appears as a drive in Finder (e.g., `/Volumes/DockerWorkspace`).

Step 5: Point Docker Desktop to this location:

1. Open **Docker Desktop → Settings → Resources → Disk image location**
2. Set it to `/Volumes/DockerWorkspace`
3. Click **"Apply & restart"**

Step 6: **After every reboot**, double-click the `.sparseimage` file to mount it before starting Docker Desktop. Or automate it by adding it to your Login Items (System Settings → General → Login Items → add the `.sparseimage` file).

14.4 Option C: Create a Size-Limited Workspace Folder (Simplest Extra Protection)

If options A and B feel too complex, you can simply limit the workspace folder that the container uses. This does not protect Docker's internal storage, but it does protect against the agent filling your drive with data files.

Windows

```
# Create a 30 GB disk image file
dd if=/dev/zero
  ↪  of=~/agent_workspace.img \
   bs=1M count=30720
  ↪  status=progress

# Format it as a filesystem
mkfs.ext4 ~/agent_workspace.img

# Create the mount point and
  ↪  mount it
mkdir -p
  ↪  ~/coding-agent-workshop/workspa
sudo mount -o loop
  ↪  ~/agent_workspace.img \

  ↪  ~/coding-agent-workshop/workspa
sudo chmod 777
  ↪  ~/coding-agent-workshop/workspace
```

After every reboot, re-mount with:

```
sudo mount -o loop
  ↪  ~/agent_workspace.img \

  ↪  ~/coding-agent-workshop/workspace
```

Mac

```
# Create a 30 GB sparse disk
  ↪  image
hdiutil create -size 30g -fs
  ↪  APFS \
  -volname AgentWorkspace \
  -type SPARSE
    ↪  ~/agent_workspace.sparseimage

# Mount it
hdiutil attach
  ↪  ~/agent_workspace.sparseimage

# Now use
  ↪  /Volumes/AgentWorkspace as
  ↪  your
# workspace in
  ↪  docker-compose.yml
```

Then update your `docker-compose.yml` to use this mounted volume:

```
volumes:
  - /Volumes/AgentWorkspace:/home/agent/project    # Mac
  # or for Windows/WSL:
```

```
  # - ~/coding-agent-workshop/workspace:/home/agent/project
```

14.5 Understanding Volume Mounts: How Host Folders Connect to the Container

Volume mounts are the bridge between your computer and the container. They control what the agent can see and where results appear.

```
Your Computer (Host)              Docker Container
====================              ================

~/coding-agent-workshop/            /home/agent/
  workspace/  ←  volume  →  project/      (agent reads & writes here)
  output/     ←  volume  →  output/       (results appear here)
  CLAUDE.md   ←  volume  →  CLAUDE.md     (read-only instructions)

  Dockerfile  (not mounted, only used during build)
  my-files/   (NOT visible to container --- stays private)
```

Key rules:

- The agent can **only** access folders you explicitly mount with `-v` or `volumes:` in `docker-compose.yml`
- Anything NOT mounted is invisible to the container
- Mounts are bidirectional: changes inside the container appear on your host, and vice versa
- Use `:ro` (read-only) to mount files the agent should read but not change (e.g., `CLAUDE.md:ro`)

Choosing your shared folder: You decide where on your host the shared folder lives. Here is how to think about it:

What You Want	How to Set It Up
Simple sharing in your project folder	`-v ./workspace:/home/agent/project` (the default)
Shared folder on a different drive	`-v /Volumes/ExternalSSD/agent-data:/home/agent/proje` (Mac) or `-v /mnt/d/agent-data:/home/agent/project` (Windows/WSL)
Size-limited workspace	Mount a disk image (see Option C above), then use that as the volume path
Read-only data sharing	`-v ~/my-datasets:/home/agent/data:ro` — agent can read your datasets but not modify them
Multiple shared folders	Add multiple `-v` lines, each mapping a different host path to a different container path

14.6 Summary: Picking Your Isolation Level

Level	Setup Effort	What It Protects	Recommended For
Default Docker (container only)	None	Agent cannot escape the container	Quick experiments, supervised sessions
Resource limits (docker-compose.yml)	Low	Limits RAM, CPU, processes	All use cases (always recommended)

Level	Setup Effort	What It Protects	Recommended For
Size-limited workspace (Option C)	Medium	Prevents agent from filling your disk with data	Longer autonomous sessions
Separate Docker storage (Option A)	Medium	Keeps Docker data off your system drive	Regular Docker users
Dedicated disk image (Option B)	High	Hard cap on ALL Docker storage	Maximum safety, unattended agents

Our recommendation for the workshop

Use the default Docker setup with resource limits (docker-compose.yml from Part 5). This provides excellent isolation with minimal setup. Add Option C (size-limited workspace) if you plan to let agents run for extended periods.

---

15 Glossary

Term	Definition
Container	A running, isolated environment created from an image. Like a lightweight virtual machine.
Dockerfile	A text file with instructions for building an image.
Image	A frozen snapshot of an environment. Containers are created from images.
Volume	A way to share data between your host system and a container, or to persist data across container restarts.
WSL 2	Windows Subsystem for Linux — lets you run Linux tools on Windows. Required by Docker Desktop on Windows.
API Key	A secret string that authenticates you with an AI service. Like a password for the API.
Sandbox	An isolated environment where software can run without affecting the rest of the system.
Terminal	A text-based interface for running commands. “Ubuntu” app on Windows, “Terminal” app on Mac and Linux.
Shell	The program that interprets your commands in the terminal (usually `bash` or `zsh`).