



Faculty IT & Research Support Team

The Faculty of Science and Environments dedicated IT Research Support Team.

FIRST work as an independent specialist engineers supporting and providing IT solutions needed for the delivery of teaching and research in FSE.
<https://livenorthumbria.ac.sharepoint.com/sites/FIRST>

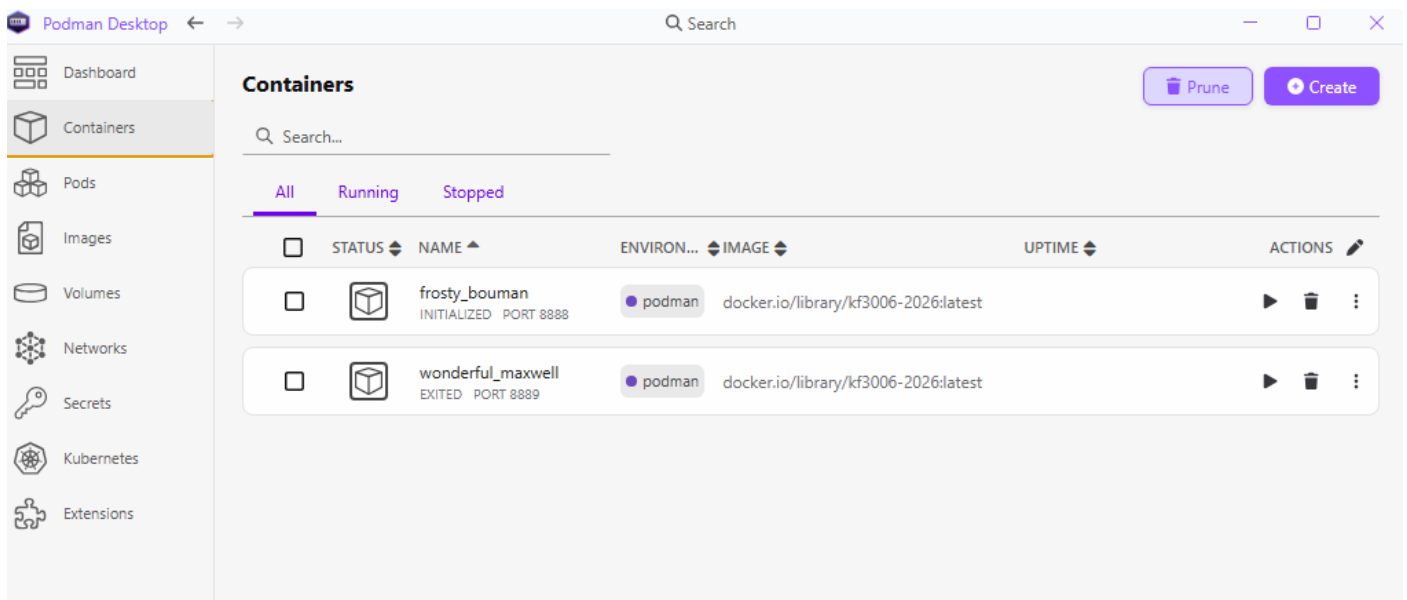
Document Version 1.0 | Kaveen Perera

This document provides sufficient information on running the container for the KF6003 jupyter container. Podman Desktop has many other features not covered in this document. Please refer to the latest Podman Desktop documentation online for any additional details.

Running container Instances with Podman

A container instance is a running (or stopped) copy created from an image. Each instance gets its own writable layer, process, network settings, and port/volume mappings, while the original image remains unchanged.

- Many containers can be created from the same image, each configured differently
- Changes made inside a container do not affect the source image
- Deleting a container does not delete the image
- An image cannot be deleted while containers still depend on it



The above screenshot shows two container instances running from the same kf6003-2026 image. Podman can assign a random name to each Container instance at start up. You can optionally set these names.

Workflow

1. Download and extract the provided compressed container build files
2. Build the image in Podman using the provided Containerfile (included inside build files archive)

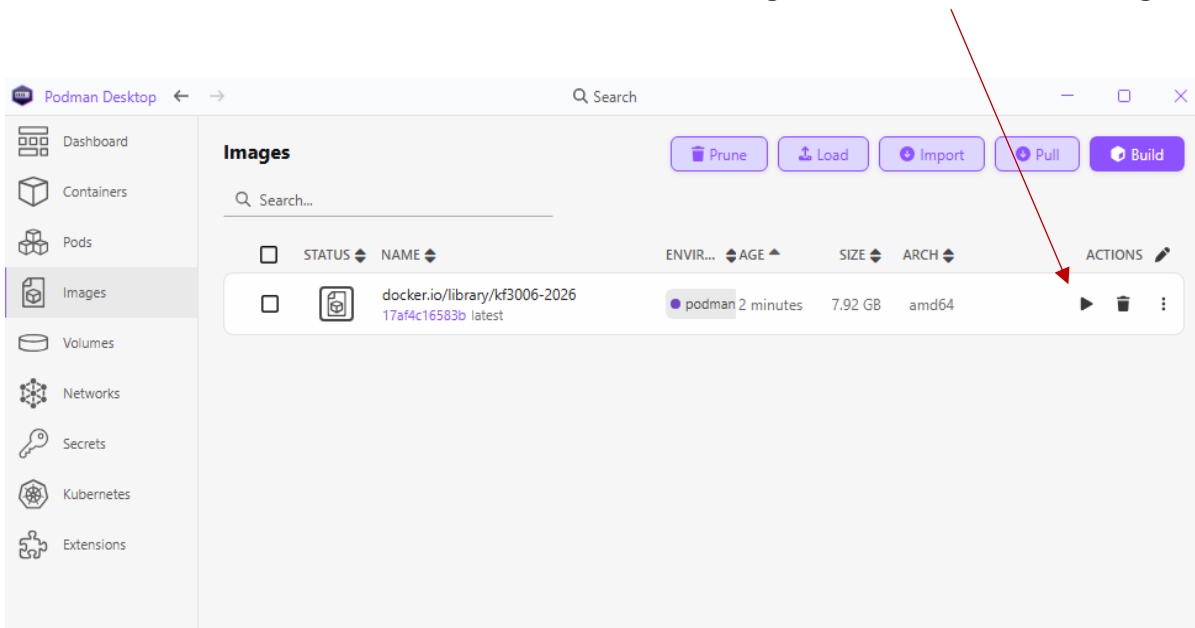
Multiple container instances can be run from a container image.

Previously created container instances are listed in the Containers tab on.

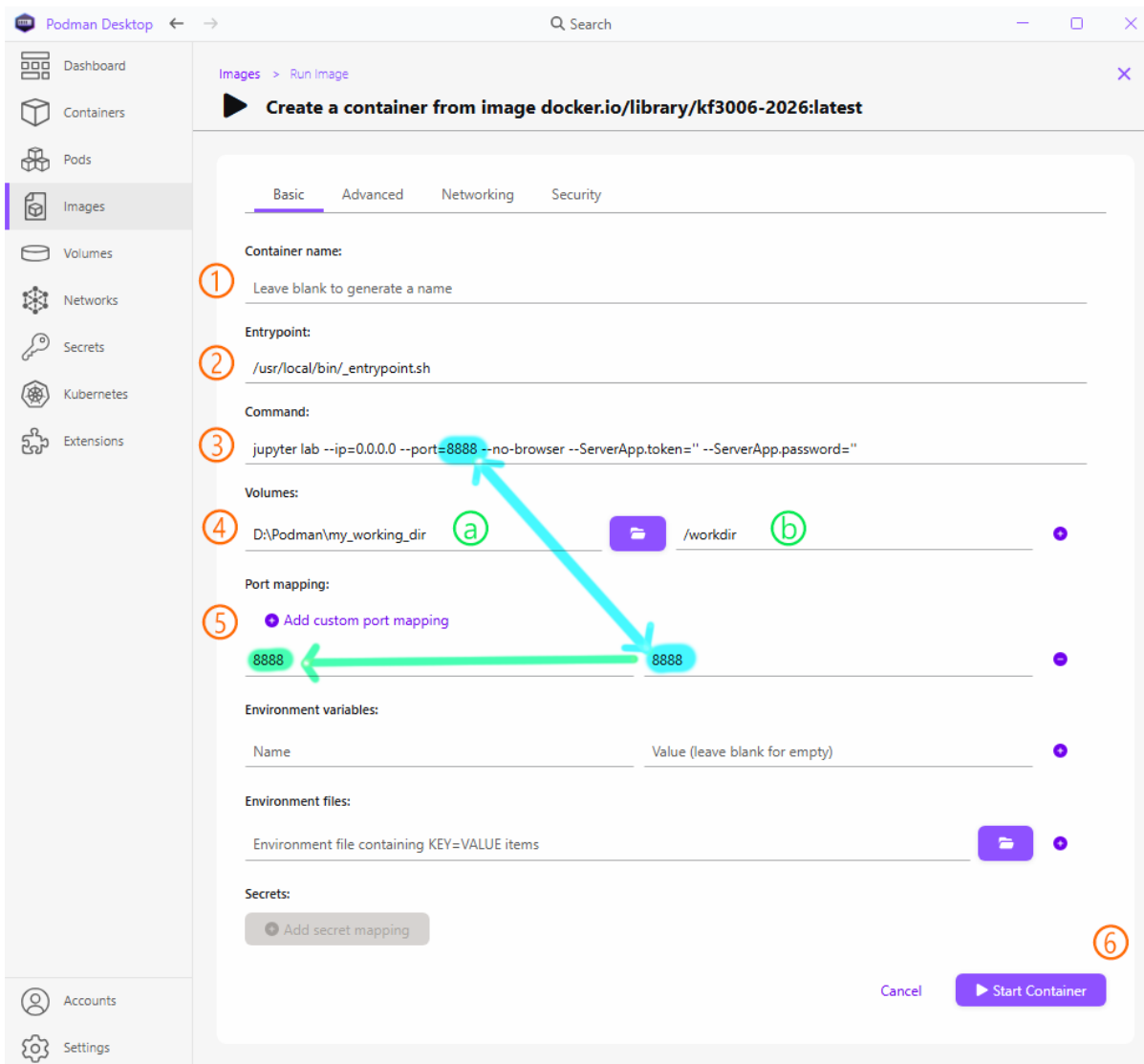
Running a Container for the First Time in Podman Desktop

The first time an image is used, a new container instance must be created and configured (name, port mapping, volumes, command). This is done through the 'Run Image' dialog rather than the 'Start' button, since no existing instance exists yet.

To create and run a new container instance from an image, click the ▶ button in image Actions.



This will open the Run image dialog. The settings shown must be used with the KF6003 Jupyter container. A detailed description is provided beneath the screenshot.



The **Basic** tab presents the following fields:

① Container name

Optional. Leave blank to let Podman auto-generate a name (e.g. funky_turing). Set explicitly when scripting or referencing the container later (e.g. docker/podman exec <name>).

② Entrypoint

The fixed executable the container runs on startup, inherited from the image (here /usr/local/bin/_entrypoint.sh, the micromamba/conda activation wrapper). Normally left untouched unless overriding image defaults.

Copy paste entry point if not auto filled:

```
/usr/local/bin/_entrypoint.sh
```

③ Command

Arguments passed to the entrypoint (what actually runs *inside* the activated environment)

Copy paste this command:

```
jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --
ServerApp.token=' ' --ServerApp.password=' '
```

This is where the container's internal listening port is defined (--port=8888).

④ Volumes

Bind-mounts a host directory into the container filesystem.

- (a) Host path - the folder on the physical/Windows machine (e.g. D:\Podman\my_working_dir). You can create your own working folder in a convenient location.
- (b) Container path - where that folder appears *inside* the container. The KF6003 module container has been pre-configured to use /workdir as the working directory. You can use this setting as it is.

Left side = outside (host), right side = inside (container). This is a one-way mapping: what's on the host becomes visible at the container path, and changes sync both directions.

The container working director has been set to /workdir inside the container at build time. Optionally binding a local folder will allow you to save any files you create during runtime. Any other changes to the container during runtime is discarded. (If you are curious examine the Containerfile inside the image build folder to understand the build process.)

⑤ Port mapping

Exposes a container-internal port to the host network.

- Left value = host port (what you connect to from outside, e.g. http://localhost:8888).
- Right value = container port (what the process inside is actually bound to).
These do not have to match numerically, only the *right* side must equal the port the process binds to internally.

Technical background

How the port numbers are linked

The port isn't automatically synced between the **Command** field and **Port mapping**, it's a manual convention you have to keep consistent yourself:

- In **Command**, --port=8888 tells Jupyter to bind to port 8888 *inside* the container.
- In **Port mapping**, the **right-hand box** (container side) must match that same value (8888), because that's the port Podman needs to forward *from*.
- The **left-hand box** (host side) is the port you'll actually browse to on your machine, it can be any free port, but is set to 8888 here for simplicity/consistency. If this port number is linked

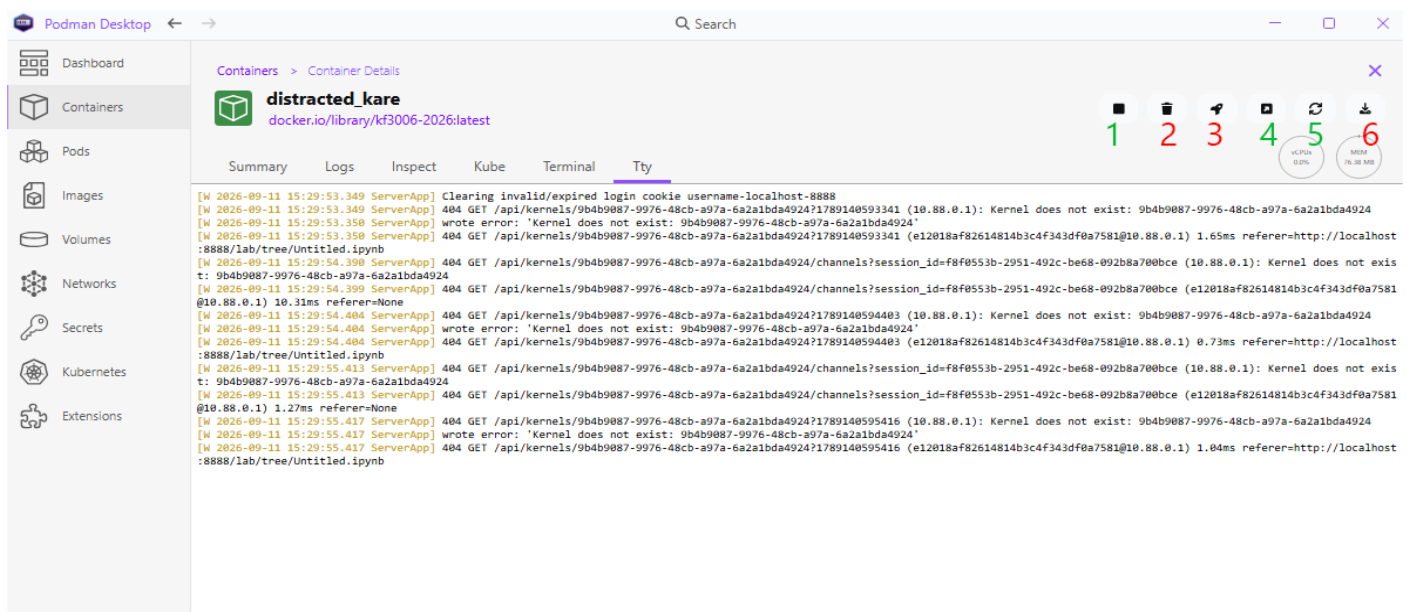
to a different service Podman will produce a warning, you can change it to any free port. For consistency we use increments of the default port (8889, 8890 etc.)

The teal arrow (Command > container-side port) represents "this is the port the app binds to," and the green arrow (container-side < - > host-side) represents "this is the forwarding rule." If you changed --port=8889 in Command but left mapping as 8888:8888, the container's internal service and the port mapping would no longer line up, and the app would be unreachable.

A clean rule of thumb is Container-side port (mapping) must always equal the port in Command. Host-side port is your free choice.

⑥ Start Container

Applies all settings above and launches the container. Podman will automatically switch to the Containers > Container Details > Tty and display the live Jupyter server output.



Once a Container instance is created the following options are available.

1: Stop Container

2: Delete Container – Use only if you are building a new updated image.

3: ~~Deploy To Kubernetes~~ – **Not supported or required** for this module

4: Open Browser – Open the URL on the browser with correctly mapped ports

5: Restart Container

6: ~~Export Container~~ - **Do not use**, this will create an incompatible broken export

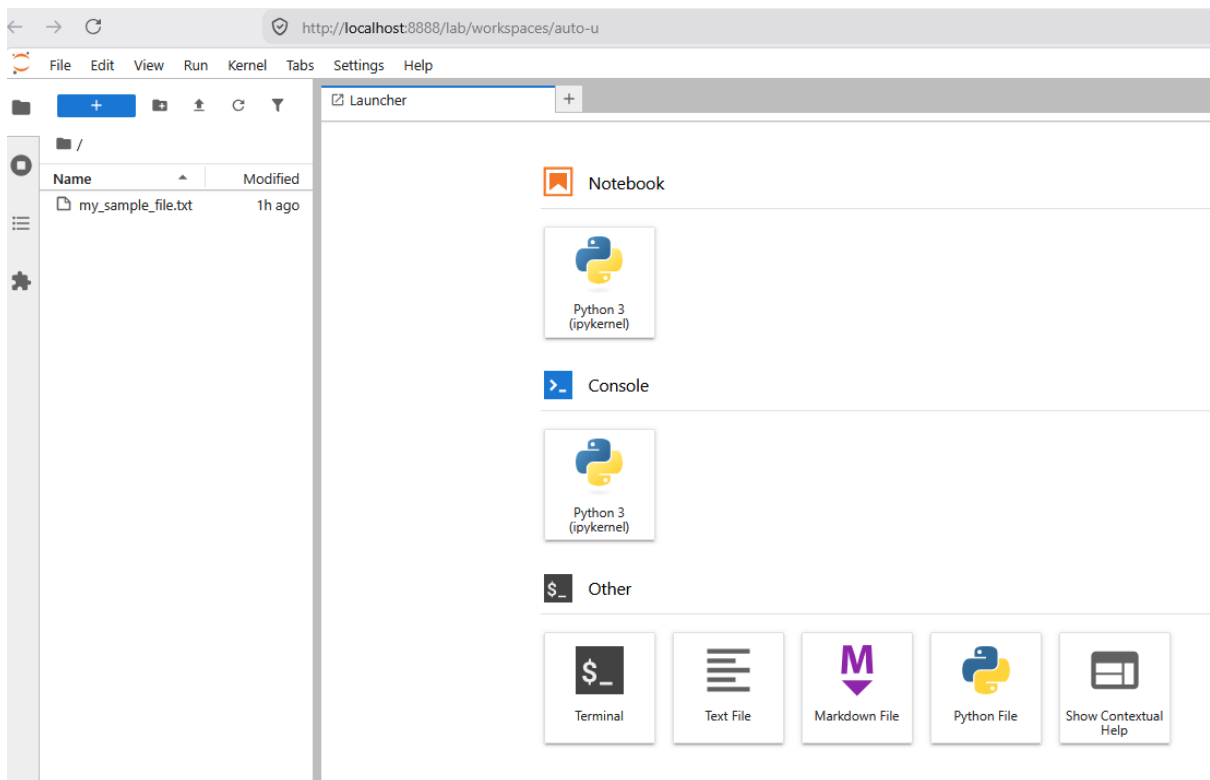


Figure 1: 'Open Browser' option will open the localhost url with correct URL mappings

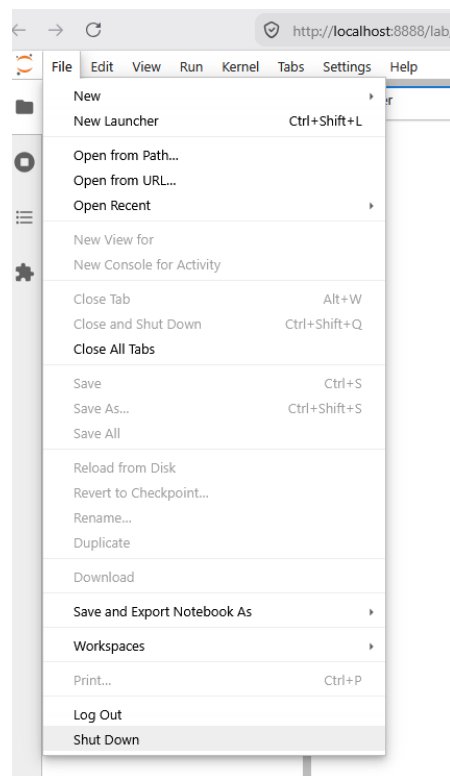
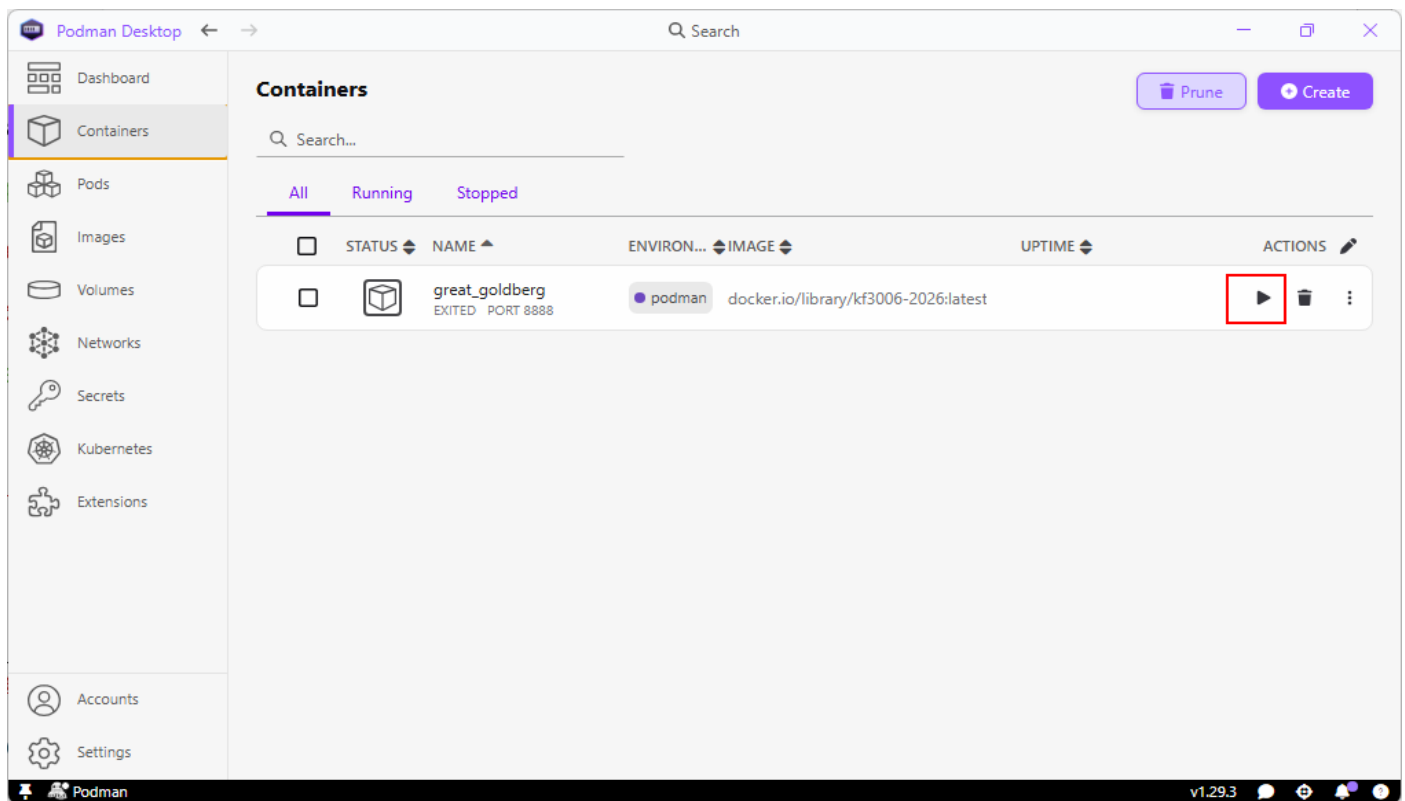


Figure 2: Shutting down Jupyter server will also stop the container instance.

Re-running an Existing Container in Podman Desktop



Once a container instance has been created and stopped, it does not need to be recreated from the image each time. Podman Desktop allows the same instance to be started again directly, preserving its name, port mappings, volume mounts, and any files written inside it.

- Click the ► button on the Container actions to start the existing container instance.
- **Do not use 'Run Image' again**, as this creates a new instance rather than resuming the existing one
- Wait for the status to change to 'Running'
- Click on the container name to view logs, or open the exposed port in a browser to access the running service

Any files created inside the container (outside of mounted volumes) persist between stop/start cycles, but are lost if the container is deleted.

Use 'Run Image' only when a new, separate instance is required

If you stop / restart a running container instance the browser will display the following error. Browser connection will auto restore after restarting the container instance

