Wasm workloads

- Turn on Wasm workloads
- Usage examples
  - Running a Wasm application with docker run
  - Running a Wasm application with Docker Compose
  - Running a multi-service application with Wasm
  - Building and pushing a Wasm module
- Troubleshooting
  - Unknown runtime specified
  - Failed to start shim: failed to resolve runtime path
- Known issues

{{< summary-bar feature_name="Wasm workloads" >}}

> Wasm workloads are deprecated and will be removed in a future Docker Desktop release. This feature is no longer actively maintained.

WebAssembly (Wasm) is a fast, light alternative to Linux and Windows containers. With Docker Desktop, you can now run Wasm workloads side by side with traditional containers.

This page provides information about the ability to run Wasm applications alongside your Linux containers in Docker.

> Learn more about Wasm use cases and tradeoffs in the Docker Wasm technical preview blog post.

Turn on Wasm workloads

Wasm workloads require the containerd image store feature to be turned on. If you're not already using the containerd image store, then pre-existing images and containers will be inaccessible.

1. Navigate to **Settings** in Docker Desktop.
2. In the **General** tab, check **Use containerd for pulling and storing images**.
3. Go to **Beta features** and check the **Enable Wasm** option.
4. Select **Apply** to save the settings.

Docker Desktop downloads and installs the following runtimes:

- `io.containerd.slight.v1`
- `io.containerd.spin.v2`
- `io.containerd.wasmedge.v1`
- `io.containerd.wasmtime.v1`
- `io.containerd.lunatic.v1`
- `io.containerd.wws.v1`
- `io.containerd.wasmer.v1`

Usage examples

Running a Wasm application with `docker run`

The following `docker run` command starts a Wasm container on your system:

```
$ docker run \
  --runtime=io.containerd.wasmedge.v1 \
  --platform=wasi/wasm \
  secondstate/rust-example-hello
```

After running this command, you can visit http://localhost:8080/ to see the "Hello world" output from this example module.

If you are receiving an error message, see the troubleshooting section for help.

Note the `--runtime` and `--platform` flags used in this command:

- `--runtime=io.containerd.wasmedge.v1`: Informs Docker Engine that you want to use the Wasm containerd shim instead of the standard Linux container runtime

- `--platform=wasi/wasm`: Specifies the architecture of the image you want to use. By leveraging a Wasm architecture, you don't need to build separate images for the different machine architectures. The Wasm runtime takes care of the final step of converting the Wasm binary to machine instructions.

Running a Wasm application with Docker Compose

The same application can be run using the following Docker Compose file:

```
services:
  app:
    image: secondstate/rust-example-hello
    platform: wasi/wasm
    runtime: io.containerd.wasmedge.v1
```

Start the application using the normal Docker Compose commands:

```
$ docker compose up
```

Running a multi-service application with Wasm

Networking works the same as you'd expect with Linux containers, giving you the flexibility to combine Wasm applications with other containerized workloads, such as a database, in a single application stack.

In the following example, the Wasm application leverages a MariaDB database running in a container.

1. Clone the repository.

   ```
   $ git clone https://github.com/second-state/microservice-rust-mysql.git
   Cloning into 'microservice-rust-mysql'...
   remote: Enumerating objects: 75, done.
   remote: Counting objects: 100% (75/75), done.
   remote: Compressing objects: 100% (42/42), done.
   remote: Total 75 (delta 29), reused 48 (delta 14), pack-reused 0
   ```

```
Receiving objects: 100% (75/75), 19.09 KiB | 1.74 MiB/s, done.
Resolving deltas: 100% (29/29), done.
```

2. Navigate into the cloned project and start the project using Docker Compose.

```
$ cd microservice-rust-mysql
$ docker compose up
[+] Running 0/1
 server
Warning
                    0.4s
[+] Building 4.8s (13/15)
...
microservice-rust-mysql-db-1      | 2022-10-19 19:54:45 0 [Note] mariadbd: ready for
connections.
microservice-rust-mysql-db-1      | Version: '10.9.3-MariaDB-
1:10.9.3+maria~ubu2204'  socket: '/run/mysqld/mysqld.sock'  port: 3306  mariadb.org
binary distribution
```

   If you run `docker image ls` from another terminal window, you can see the Wasm image in your image store.

```
$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
server       latest    2c798ddecfa1   2 minutes ago   3MB
```

   Inspecting the image shows the image has a `wasi/wasm` platform, a combination of OS and architecture:

```
$ docker image inspect server | grep -A 3 "Architecture"
        "Architecture": "wasm",
        "Os": "wasi",
        "Size": 3001146,
        "VirtualSize": 3001146,
```

3. Open the URL `http://localhost:8090` in a browser and create a few sample orders. All of these are interacting with the Wasm server.
4. When you're all done, tear everything down by hitting `Ctrl+C` in the terminal you launched the application.

Building and pushing a Wasm module

1. Create a Dockerfile that builds your Wasm application.
   Exactly how to do this varies depending on the programming language you use.
2. In a separate stage in your `Dockerfile`, extract the module and set it as the `ENTRYPOINT`.

```
# syntax=docker/dockerfile:1
FROM scratch
COPY --from=build /build/hello_world.wasm /hello_world.wasm
ENTRYPOINT [ "/hello_world.wasm" ]
```

3. Build and push the image specifying the `wasi/wasm` architecture. Buildx makes this easy to do in a single command.

```
$ docker buildx build --platform wasi/wasm -t username/hello-world .
...
=> exporting to
image
0.0s
=> => exporting
layers
0.0s
=> => exporting manifest
sha256:2ca02b5be86607511da8dc688234a5a00ab4d58294ab9f6beaba48ab3ba8de56  0.0s
=> => exporting config
sha256:a45b465c3b6760a1a9fd2eda9112bc7e3169c9722bf9e77cf8c20b37295f954b    0.0s
=> => naming to docker.io/username/hello-
world:latest                                           0.0s
=> => unpacking to docker.io/username/hello-
world:latest                                          0.0s
$ docker push username/hello-world
```

Troubleshooting

This section contains instructions on how to resolve common issues.

Unknown runtime specified

If you try to run a Wasm container without the containerd image store, an error similar to the following displays:

```
docker: Error response from daemon: Unknown runtime specified io.containerd.wasmedge.v1.
```

Turn on the containerd feature in Docker Desktop settings and try again.

Failed to start shim: failed to resolve runtime path

If you use an older version of Docker Desktop that doesn't support running Wasm workloads, you will see an error message similar to the following:

```
docker: Error response from daemon: failed to start shim: failed to resolve runtime path:
runtime "io.containerd.wasmedge.v1" binary not installed "containerd-shim-wasmedge-v1": file
does not exist: unknown.
```

Update your Docker Desktop to the latest version and try again.

Known issues

- Docker Compose may not exit cleanly when interrupted. As a workaround, clean up `docker-compose` processes by sending them a SIGKILL (`killall -9 docker-compose`).
- Pushes to Docker Hub might give an error stating `server message: insufficient_scope: authorization failed`, even after signing in through Docker Desktop. As a workaround, run `docker login` in the CLI

□□□ #2
□ □□□ □□□ 2026-10-31 19:12:29 UTC
□ □□□ □□□ 2026-10-31 19:12:31 UTC