Announcing Docker+Wasm Technical Preview 2

This article was fetched from an rss feed(22/03/2023)

We recently announced the first Technical Preview of Docker+Wasm, a special build that makes it possible to run Wasm containers with Docker using the WasmEdge runtime. Starting from version 4.15, everyone can try out the features by activating the containerd image store experimental feature.

We didn't want to stop there, however. Since October, we've been working with our partners to make running Wasm workloads with Docker easier and to support more runtimes.

Now we are excited to announce a new Technical Preview of Docker+Wasm with the following three new runtimes:

- `spin` from Fermyon
- `slight` from Deislabs
- `wasmtime` from Bytecode Alliance

All of these runtimes, including WasmEdge, use the runwasi library.

What is runwasi?

Runwasi is a multi-company effort to make a library in Rust that makes it easier to write containerd shims for Wasm workloads. Last December, the runwasi project was donated and moved to the Cloud Native Computing Foundation's containerd organization in GitHub.

With a lot of work from people at Microsoft, Second State, Docker, and others, we now have enough features in runwasi to run Wasm containers with Docker or in a Kubernetes cluster. We still have a lot of work to do, but there are enough features for people to start testing.

If you would like to chat with us or other runwasi maintainers, join us on the CNCF's #runwasi channel.

Get the update

Ready to dive in and try it for yourself? Great! Before you do, understand that this is a technical preview build of Docker Desktop, so things might not work as expected. Be sure to back up your containers and images before proceeding.

Download and install the appropriate version for your system, then activate the containerd image store (**Settings > Features in development > Use containerd** for pulling and storing images), and you'll be ready to go.

![Features in development screen with "Use containerd" option selected.](https://www.docker.com/wp-content/uploads/2023/03/docker-desktop-containerd-enabled-1110x669.png "Features in development screen with "Use containerd" option selected. - docker desktop containerd enabled")

Figure 1: Docker Desktop beta features in development.

- Mac (Intel)
- Mac (Arm)
- Linux (deb, Intel)
- Linux (deb, Arm)
- Linux (rpm, Intel)
- Linux (Arch)
- Windows

Let's take Wasm for a spin

The WasmEdge runtime is still present in Docker Desktop, so you can run:

```
$ docker run --rm --runtime=io.containerd.wasmedge.v1 --platform=wasi/wasm secondstate/rust-example-hello:latest Hello WasmEdge!
```

You can even run the same image with the `wasmtime` runtime:

```
$ docker run --rm --runtime=io.containerd.wasmtime.v1 --platform=wasi/wasm secondstate/rust-example-hello:latest Hello WasmEdge!
```

In the next example, we will deploy a Wasm workload to Docker Desktop's Kubernetes cluster using the `slight` runtime. To begin, make sure to activate Kubernetes in Docker Desktop's settings, then create an `example.yaml` file:

cat > example.yaml <<EOT apiVersion: apps/v1 kind: Deployment metadata: name: wasm-slight spec: replicas: 1 selector: matchLabels: app: wasm-slight template: metadata: labels: app: wasm-slight spec: runtimeClassName: wasmtime-slight-v1 containers: - name: hello-slight image: dockersamples/slight-rust-hello:latest command: ["/"] resources: requests: cpu: 10m memory: 10Mi limits: cpu: 500m memory: 128Mi

apiVersion: v1 kind: Service metadata: name: wasm-slight spec: type: LoadBalancer ports: - protocol: TCP port: 80 targetPort: 80 selector: app: wasm-slight EOT

Note the `runtimeClassName`, kubernetes will use this to select the right runtime for your application.

You can now run:

```
$ kubectl apply -f example.yaml
```

Once Kubernetes has downloaded the image and started the container, you should be able to curl it:

```
$ curl localhost/hello hello
```

You now have a Wasm container running locally in Kubernetes. How exciting!

Note: You can take this same yaml file and run it in AKS.

Now let's see how we can use this to run Bartholomew. Bartholomew is a micro-CMS made by Fermyon that works with the spin runtime. You'll need to clone this repository; it's a slightly modified Bartholomew template.

The repository already contains a Dockerfile that you can use to build the Wasm container:

```
FROM scratch COPY . . ENTRYPOINT [ "/modules/bartholomew.wasm" ]
```

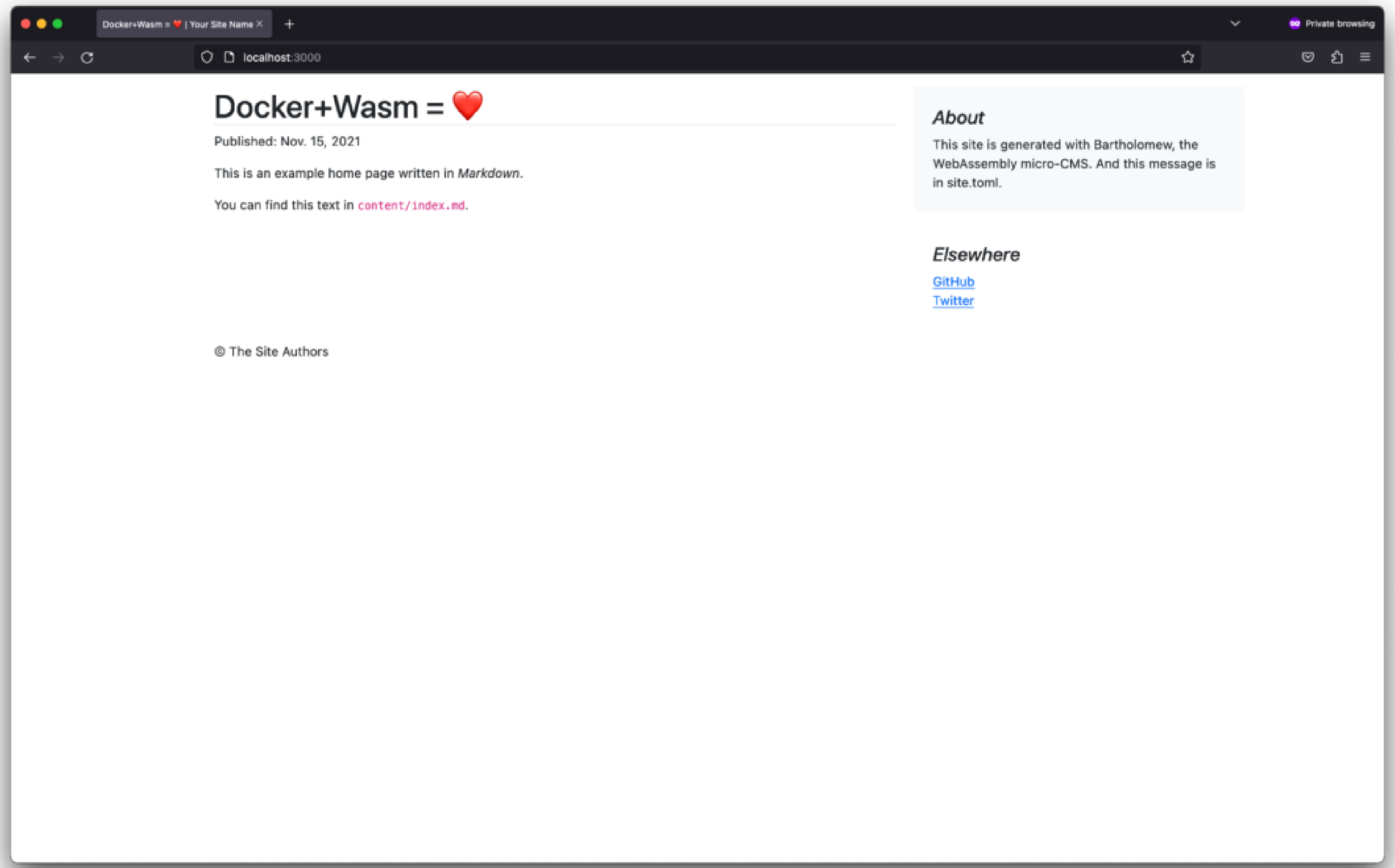
The Dockerfile copies all the contents of the repository to the image and defines the build bartholomew Wasm module as the entry point of the image.

```
$ cd docker-wasm-bartholomew $ docker build -t my-cms . [+] Building 0.0s (5/5) FINISHED => [internal] load build definition from Dockerfile 0.0s => => transferring dockerfile: 147B 0.0s => [internal] load .dockerignore 0.0s => => transferring context: 2B 0.0s => [internal] load build context 0.0s => => transferring context: 2.84kB 0.0s => CACHED [1/1] COPY . . 0.0s => exporting to image 0.0s => => exporting layers 0.0s => => exporting manifest sha256:cf85929e5a30bea9d436d447e6f2f2e 0.0s => => exporting config sha256:0ce059f2fe907a91a671f37641f4c5d73 0.0s => => naming to docker.io/library/my-cms:latest 0.0s => => unpacking to docker.io/library/my-cms:latest 0.0s
```

You are now ready to run your first WebAssembly micro-CMS:

```
$ docker run --runtime=io.containerd.spin.v1 -p 3000:80 my-cms
```

If you go to http://localhost:3000, you should be able to see the Bartholomew landing page (Figure 2).



Docker+Wasm = ❤️

Published: Nov. 15, 2021

This is an example home page written in *Markdown*.

You can find this text in `content/index.md`.

About

This site is generated with Bartholomew, the WebAssembly micro-CMS. And this message is in site.toml.

Elsewhere

GitHub
Twitter

© The Site Authors

Figure 2: Bartholomew landing page.

We'd love your feedback

All of this work is fresh from the oven and relies on the containerd image store in Docker, which is an experimental feature we've been working on for almost a year now. The good news is that we already see how this hard work can benefit everyone by adding more features to Docker. We're still working on it, so let us know what you need.

If you want to help us shape the future of WebAssembly with Docker, try it out, let us know what you think, and leave feedback on our public roadmap.