WebAssembly (WASM) Workloads

Bacalhau supports running programs that are compiled to WebAssembly (WASM). With the Bacalhau client, you can upload WASM programs, retrieve data from public storage, read and write data, receive program arguments, and access environment variables.

Prerequisites and Limitations

1. **Supported WebAssembly System Interface (WASI)** Bacalhau can run compiled WASM programs that expect the WebAssembly System Interface (WASI) Snapshot 1. Through this interface, WebAssembly programs can access data, environment variables, and program arguments.
2. **Networking Restrictions** All ingress/egress networking is disabled – you won't be able to pull `data/code/weights` etc. from an external source. WASM jobs can say what data they need using URLs or

CIDs (Content IDentifier) and can then access the data by reading from the filesystem.

3. **Single-Threading:** There is no multi-threading as WASI does not expose any interface for it.

Onboarding Your Workload

Step 1: Replace network operations with filesystem reads and writes

If your program typically involves reading from and writing to network endpoints, follow these steps to adapt it for Bacalhau:

1. **Replace Network Operations:** Instead of making HTTP requests to external servers (e.g., example.com), modify your program to read data from the local filesystem.
2. **Input Data Handling:** Specify the input data location in Bacalhau using the `--input` flag when running the job. For instance, if your program used to fetch data from `example.com`, read from the `/inputs` folder locally, and provide the URL as input when executing the Bacalhau job. For example, `--input http://example.com`.
3. **Output Handling:** Adjust your program to output results to standard output (`stdout`) or standard error (`stderr`) pipes. Alternatively, you can write results to the filesystem, typically into an output mount. In the case of

WASM jobs, a default folder at `/outputs` is available, ensuring that data written there will persist after the job concludes.

By making these adjustments, you can effectively transition your program to operate within the Bacalhau environment, utilizing filesystem operations instead of traditional network interactions.

> You can specify additional or different output mounts using the `-o` flag.

Step 2: Configure your compiler to output WASI-compliant WebAssembly

You will need to compile your program to WebAssembly that expects WASI. Check the instructions for your compiler to see how to do this.

For example, Rust users can specify the `wasm32-wasi` target to `rustup` and `cargo` to get programs compiled for WASI WebAssembly. See the Rust example for more information on this.

Step 3: Upload the input data

Data is identified by its content identifier (CID) and can be accessed by anyone who knows the CID. You can use either of these methods to upload your data:

- Copy data from a URL to public storage
- Pin Data to public storage
- Copy Data from S3 Bucket to public storage.

> You can mount your data anywhere on your machine, and Bacalhau will be able to run against that data

Step 4: Run your program

You can run a WebAssembly program on Bacalhau using the `bacalhau wasm run` command.

```
bacalhau wasm run
```

Run Locally Compiled Program:

If your program is locally compiled, specify it as an argument. For instance, running the following command will upload and execute the `main.wasm` program:

```
bacalhau wasm run main.wasm
```

The program you specify will be uploaded to a Bacalhau storage node and will be publicly available.

Alternative Program Specification:

You can use a Content IDentifier (CID) for a specific WebAssembly program.

```
bacalhau wasm run Qmajb9T3jBdMSp7xh2JruNrqg3hniCnM6E
```

Input Data Specification:

Make sure to specify any input data using `--input` flag.

```
bacalhau wasm run --input http://example.com
```

This ensures the necessary data is available for the program's execution.

Program arguments

You can give the WASM program arguments by specifying them after the program path or CID. If the WASM program is already compiled and located in the current directory, you can run it by adding arguments after the file name:

```
bacalhau wasm run echo.wasm hello world
```

For a specific WebAssembly program, run:

```
bacalhau wasm run Qmajb9T3jBdMSp7xh2JruNrqg3hniCnM6E
```

Write your program to use program arguments to specify input and output paths. This makes your program more flexible in handling different configurations of input and output volumes.

For example, instead of hard-coding your program to read from `/inputs/data.txt`, accept a program argument that should contain the path and then specify the path as an argument to `bacalhau wasm run`:

```
bacalhau wasm run prog.wasm /inputs/data.txt
```

Your language of choice should contain a standard way of reading program arguments that will work with WASI.

Environment variables

You can also specify environment variables using the `-e` flag.

```
$ bacalhau wasm run prog.wasm -e HELLO=world
```

Examples

See the Rust example for a workload that leverages WebAssembly support.

Support

If you have questions or need support or guidance, please reach out to the Bacalhau team via Slack (**#general** channel).

Previous
Docker Workloads

Next
Running Nodes