How it works

Here we will describe how Conduit runs standalone processors under the hood. This is useful to understand when writing a processor SDK.

WebAssembly

Conduit usesWebAssembly (Wasm) to run standalone processors. Wasm is a binary instruction format, designed as a portable target for compilation of high-level languages like C/C++, Rust, Java andmore . Wasm binaries are run using a Wasm runtime, in our case thewazero runtime.

The main reason for using Wasm is to provide a secure and portable execution environment for the processor. Wasm is designed to be run in a sandboxed environment, which means that the processor cannot access the host system directly. This is important to ensure that the processor cannot access or modify the host system, and that it cannot access or modify other processors running in the same environment.

WASI support and limitations

Conduit usesWASI (WebAssembly System Interface) to provide a set of standard APIs that the processor can use to interact with the host system. Since WASI is still a work in progress, we are currently using the interface defined inWASI preview 1 .

info Because the processor has access to a limited set of system calls, it is not possible to use all the features of the host system. For example, the processor cannot access the file system, network, or any other system resource directly.

Processor lifecycle

A standalone processor is a plugin, compiled into a Wasm binary which interacts with Conduit using two host functions:command_request andcommand_response . The plugin is expected to continuously call these functions, allocate memory as needed, parse commands, execute them, and respond back to the host.

Below is a sequence diagram that shows the high level interaction between Conduit and a standalone processor.

1. The processor starts to run only when Conduit starts a pipeline containing a
2. standalone processor. The lifetime of the processor is tied to the lifetime
3. of the pipeline.
4. The processor allocates initial memory for command retrieval. This buffer
5. should be reused for every command request. In case the buffer is too small,
6. the processor should reallocate a larger buffer (see point 4).
7. The processor callscommand_request
8. to retrieve a command from Conduit.
9. This call is blocking and will return when a command is available.
10. Conduit populates the allocated memory with a command and returns the size of
11. the command or an error code. Seeerror codes
12. for more a list
13. of possible error codes. Seecommand_request
14. for more
15. information about the command format.
16. In case Conduit returns an error code, the processor should exit with the
17. error code.
18. The processor parses and executes the command. Note that everything except
19. the actual command business logic can and should be implemented by a
20. processor SDK.
21. The processor callscommand_response
22. to send the result back to Conduit.
23. This call should happen even if the command execution failed (the result
24. should include the error). Seecommand_response
25. for more information about the response format.
26. Conduit acknowledges the result and the processor can continue retrieving
27. the next command.

Host functions

Conduit exposes a module namedconduit with host functions, that allow the processor to interact with Conduit.

Example host function import in Go:

//go:wasmimport conduit command_request func

_commandRequest (ptr unsafe . Pointer , size uint32)

uint32 All host functions accept two arguments:

1. A pointer to a memory address where Conduit can retrieve the request and/or
2. write the result.
3. The size of the memory buffer.

All host functions return a single integer which indicates the size of the response or an error code. Seeerror codes for a list of possible error codes. If the host function returns 0 it indicates success. If the returned integer is an error code, the processor should exit immediately.

The data in the memory buffer is always serialized usingProtocol Buffers . The schema for the data can be found in theBuf schema registry .

command_request

The processor should call this function to retrieve a command request from Conduit. The function is blocking and will return only when a command is available. The returned integer represents the size of the command, or anerror code . If the allocated memory buffer is smaller than the returned command size, the processor should reallocate a larger buffer and callcommand_request again.

Conduit will populate the memory buffer with aCommandRequest , which can contain one of the following commands:

- Specify.Request
- -
- The processor should return its specifications by callingcommand_response
- with aSpecify.Response
- .
- Configure.Request
- -
- Conduit passes the configuration to the processor which should be parsed and
- stored. The processor should respond with aConfigure.Response
- .
- Open.Request
- -
- The processor should initialize its state and respond with anOpen.Response
- .
- Process.Request
- -
- The processor should process the records in the request and return the result
- using aProcess.Response
- .
- Teardown.Request
- -
- The processor should clean up its state, close any open resources and respond
- with aTeardown.Response
- .
- The plugin can expect that the next command after aTeardown.Request
- will be
- an error code indicating no more commands.

command_response

The processor should call this function to send aCommandResponse back to Conduit. The function returns 0 if the response was successfully received, or anerror code .

The memory address sent tocommand_response should contain aCommandResponse :

- Specify.Response
- should be sent in response to aSpecify.Request
- .
- Configure.Response
- should be sent in response to aConfigure.Request

- .
- Open.Response
- should be sent in response to aOpen.Request
- .
- Process.Response
- should be sent in response to aProcess.Request
- .
- Teardown.Response
- should be sent in response to aTeardown.Request
- .
- Error
- should be sent in response to any command request that failed to be processed.

Error codes

The last 100 numbers at the end of anuin32 (between 4,294,967,195 and 4,294,967,294) are reserved for error codes. If one of the host functions returns a number in this range it represents an error code and the plugin should stop running.

Current error codes:

- 4,294,967,294
-
  - no more commands. Returned bycommand_request
- when Conduit
- has no more commands to return and the plugin should gracefully stop running
- (exit code 0).
- 4,294,967,293
-
  - unknown command request. Internal error in Conduit, where
- the command request failed to be marshaled into a proto message.
- 4,294,967,292
-
  - unknown command response. Internal error in Conduit, where
- the command response failed to be unmarshaled from the proto message into a
- struct.
- 4,294,967,291
-
  - memory out of range. Internal error in Conduit, where it
- tried to write into the allocated memory, but failed to do so, as the size was
- insufficient.

Logging

The processor can log messages tostderr andstdout and they will show up in the Conduit logs. Even though no specific format is enforced, it is recommended to emit logs in a JSON format, as Conduit will be able to parse and display them in a more readable way.

There are a few standard fields that should be included in every log message:

- level
-
  - The log level, one oftrace
- ,debug
- ,info
- ,warn
- ,error
- . If
- the log level is not specified, the message will be logged without a level.
- message
-
  - The log message.

Conduit will automatically add the following fields to all messages:

- time
-
  - Will contain the time when the log was emitted.
- processor_id

-
    - The unique identifier of the processor entity in Conduit
- associated with this plugin instance.

Example log message emitted by the plugin:

{ "level" :

"info" , "contextual-info" :

"some contextual info" , "message" :

"Processor is running" } tip Please be mindful of the log level you use. Processors are executed in the hot-path of the data processing pipeline and excessive logging can have a negative impact on the performance of the pipeline.

Also make sure you respect the log level set by Conduit in the environment variableCONDUIT_LOG_LEVEL to reduce unnecessary verbosity.

Environment variables

When Conduit starts the processor, it will set the following environment variables:

- CONDUIT_LOG_LEVEL
- indicates the log level that the processor should use.
- This is useful to control the verbosity of the logs emitted by the processor.
- The log level can be one oftrace
- ,debug
- ,info
- ,warn
- ,error
- .
- CONDUIT_PROCESSOR_ID
- is the unique identifier of the processor entity in
- Conduit associated with this plugin instance. Note that Conduit will
- automatically add this ID to log messages emmitted by the plugin, so you don't
- have to attach it manually. Edit this page Previous Build your own Next Conditional Execution