Streaming Job Stdio in Namazu Conductor

Container on Demand · https://namazustudios.com/docs/add-ons/container-on-demand/streaming-job-stdio-in-namazu-conductor/

`OrchestrationService.streamStdio(execution)` returns a live, bidirectional connection to a running job's stdin/stdout/stderr — a `JobStdio` exposing `stdin` as an `OutputStream` and `stdout` / `stderr` as `InputStream`s. What backs that connection differs by provider: it's native on Kubernetes, and sidecar-based on EdgeGap and ECS, whose platforms have no built-in container-exec API of their own.

Support by Provider

Provider	Mechanism	Notes
Kubernetes	Native — equivalent to `kubectl attach`	Requires the pod to be in the `Running` phase.
ECS	`namazu-stdio-bridge` sidecar	Requires the bridge baked into your task's container image.
EdgeGap	`namazu-stdio-bridge` sidecar	Requires the bridge baked into your app version's container image.
Multiplay	Not supported	The Multiplay allocation API has no equivalent; calling `streamStdio()` throws `UnsupportedOperationException`.

Note

On EdgeGap and ECS, `streamStdio()` only works on a `JobExecution` obtained directly from `execute()` (or a copy derived from it via `getFutureForStatus` / `getStageForStatus`). An execution reconstructed from `listExecutions()` has no credential attached and throws `StdioUnavailableException`. Hold on to the object `execute()` gives you if you plan to stream stdio.

Why a Sidecar for EdgeGap/ECS

Kubernetes exposes a container-exec/attach API that Conductor can call into directly. EdgeGap and ECS don't — there's no platform-level API to reach into a running container's stdio. **namazu-stdio-bridge** closes that gap: a small wrapper process (conceptually similar to `dockerize`) that you make the entrypoint of your own container image. It execs your real entrypoint, forwards your image's `CMD` (or the `command` / `args` a `JobRequest` supplies at runtime) as that process's argv, and exposes the child's stdin/stdout/stderr over three WebSocket endpoints that Conductor's EdgeGap and ECS providers know how to speak to.

The bridge is published as a standalone binary (PyInstaller + `staticx` — no Python runtime required in your final image, and portable across glibc- and musl-based base images) at `ghcr.io/namazustudios/namazu-stdio-bridge:latest`. Vendor it into your image with a multi-stage build:

```
COPY --from=ghcr.io/namazustudios/namazu-stdio-bridge:latest
     /usr/local/bin/namazu-stdio-bridge /usr/local/bin/namazu-stdio-bridge
ENTRYPOINT ["/usr/local/bin/namazu-stdio-bridge"]
```

By default the bridge looks for your real entrypoint at `/docker-entrypoint.sh`, `/usr/local/bin/docker-entrypoint.sh`, or `/app/docker-entrypoint.sh` (first match wins) — override the candidate list with `NAMAZU_CONDUCTOR_STDIO_ENTRYPOINT` if your entrypoint lives elsewhere. Declare the bridge's listen port (`10080` by default) in your app version's or task definition's port mappings, since it has to be reachable from Conductor.

Wire Protocol

The bridge exposes three WebSocket endpoints, carrying raw bytes with no additional framing:

Path	Direction	Semantics
`{base}/0`	client → server	stdin — frames forwarded to the child process's stdin
`{base}/1`	server → client	stdout — raw chunks as produced
`{base}/2`	server → client	stderr — raw chunks as produced

`{base}` is the bridge's configured URI prefix — set it via `NAMAZU_CONDUCTOR_STDIO_URI` on the bridge, and the matching `...stdio.bridge.base.path` attribute on the Conductor provider (see Configuring Namazu Conductor Providers).

When the child process exits, the server closes `/1` and `/2` with WebSocket close code `4000 + <exit code>` (e.g. `4000` for a clean exit, `4007` for exit code 7 — clamped to the `4000`–`4255` range). If the process was killed by a signal instead, the close reason string carries the signal name (e.g. `"SIGTERM"`).

Connecting to `/1` or `/2` immediately replays a bounded ring buffer of the most recent output — up to 16 KiB for stdout and 4 KiB for stderr by default — before switching to live streaming, so a client that attaches a moment after the process started producing output still sees recent history rather than only what's written from that point on. This is a replay buffer, not a log API: output older than its capacity is gone for good.

Authentication

Every WebSocket connection to `/0`, `/1`, or `/2` must present `Authorization: Bearer <token>` at the handshake. Connections that omit it, or present the wrong token, are rejected with HTTP `401` at the handshake itself — the connect call fails outright rather than appearing to succeed and then closing. The bridge refuses to start at all if no token is configured; its stdio endpoints would otherwise be reachable, unauthenticated, by anyone who can reach the port.

You never set this token by hand. Conductor generates a fresh one per job execution, injects it into the container as the `NAMAZU_CONDUCTOR_STDIO_TOKEN` environment variable, and carries it internally on the `JobExecution` returned by `execute()` so that a later `streamStdio()` call can present it automatically.

Bridge Configuration Reference

These environment variables configure the bridge itself, inside your image — they're independent of the Conductor-side attributes covering the same concerns (port, base path) on the EdgeGap/ECS providers.

Variable	Default	Description
`NAMAZU_CONDUCTOR_STDIO_TOKEN`	*required*	Bearer token every WebSocket connection must present. The bridge refuses to start without it.
`NAMAZU_CONDUCTOR_STDIO_URI`	`/`	Base path prefix for the three WebSocket endpoints.
`NAMAZU_CONDUCTOR_STDIO_ENTRYPOINT`	`/docker-entrypoint.sh:/usr/local/bin/docker-entrypoint.sh:/app/docker-entrypoint.sh`	`os.pathsep`-separated list of candidate entrypoint paths, tried in order; the first existing, executable path wins.
`NAMAZU_CONDUCTOR_STDIO_BUFFER_SIZE`	`4096`	Chunk size in bytes per WebSocket message (distinct from the replay buffers below).
`NAMAZU_CONDUCTOR_STDIO_STDOUT_BUFFER_SIZE`	`16k`	Ring buffer capacity for stdout replay. Accepts a plain byte count or a `k` / `m` suffix. `0` disables replay.
`NAMAZU_CONDUCTOR_STDIO_STDERR_BUFFER_SIZE`	`4096`	Same, for stderr.

Variable	Default	Description
`NAMAZU_CONDUCTOR_STDIO_PORT`	`10080`	WebSocket listen port.
`NAMAZU_CONDUCTOR_STDIO_LOG_LEVEL`	`INFO`	The bridge's own log level.

Whatever you set for `NAMAZU_CONDUCTOR_STDIO_PORT` and `NAMAZU_CONDUCTOR_STDIO_URI` inside the image must match the corresponding `...stdio.bridge.port` and `...stdio.bridge.base.path` attributes on the Conductor provider Element, or Conductor won't be able to reach the bridge.