Namazu Conductor

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

Namazu Conductor is an `Element` that provides a unified container/session orchestration API for the Namazu Elements SDK. It abstracts a set of provider-specific game-server orchestration platforms — EdgeGap, AWS ECS, Kubernetes, and Unity Multiplay — behind a single `OrchestrationService` interface, so your authoritative code can launch, monitor, and stop game server workloads without depending on which platform actually hosts them.

Overview

Every orchestration platform Conductor supports has its own idea of a "profile" (a build config, a task definition, a pod template, a fleet), its own placement model (region, IP, lat/long), and its own scoping concept (namespace, cluster). Conductor normalizes these into a small, common set of types — `JobProfile`, `JobRequest`, `JobExecution`, `JobPlacement`, and `JobScope` — and exposes them through one `OrchestrationService` interface per provider. Your code targets that interface; which provider actually launches the workload is a deployment-time configuration decision, not a code change.

Each provider ships as its own Element, bound behind a Guice `PrivateModule` that exposes only `OrchestrationService` to the rest of the platform. You deploy the provider Element(s) relevant to your infrastructure — you don't need Kubernetes, ECS, EdgeGap, and Multiplay support all loaded at once.

Key Features

- One `OrchestrationService` interface across EdgeGap, AWS ECS, Kubernetes, and Unity Multiplay
- Launch, poll, list, and stop jobs (game server instances) through a common request/response model
- Portable placement hints (region, IP, latitude/longitude) and scoping (namespace, cluster) that each provider honors where it has an equivalent concept, and silently ignores where it doesn't

- Live stdin/stdout/stderr streaming for a running job via `streamStdio` — native on Kubernetes, and available on EdgeGap and ECS through a small sidecar (see Streaming Job Stdio in Namazu Conductor)
- A superuser-only cross-provider admin REST API and dashboard panel for listing, launching, and stopping jobs (see Namazu Conductor Admin API)

Module Structure

Module	Purpose
`api`	Core interfaces and data types: `OrchestrationService`, `JobRequest`, `JobExecution`, `JobProfile`, `JobPlacement`, `JobScope`, `JobStatus`, `JobStdio`
`edgegap`	EdgeGap REST API v1 implementation
`ecs`	AWS ECS implementation (AWS SDK v2; Fargate and EC2 launch types)
`kubernetes`	Kubernetes implementation (Fabric8 client; maps `PodTemplate` resources to profiles, launching `Pod` or `batch/v1 Job` workloads)
`multiplay`	Unity Multiplay implementation (fleet allocations via the Unity Services API)
`admin`	Superuser REST API and dashboard panel for listing/launching/stopping jobs across every deployed provider Element
`debug`	Local runner — boots a local MongoDB replica set, then the Elements runtime with whichever provider Elements you've configured

You typically deploy `admin` alongside whichever provider module(s) match your infrastructure. None of the modules depend on each other at runtime beyond sharing the `api` module's types.

Core Concepts

JobProfile

A `JobProfile` is a provider-specific job template, identified by a string `id`. What backs a profile differs per provider: an EdgeGap app + version, an ECS task definition family, a Kubernetes `PodTemplate`, or a Multiplay fleet + build configuration. Call `OrchestrationService.getAvailableProfiles()` to discover the profiles a given provider currently exposes, or `findAvailableProfile(id)` to look up one by id.

JobRequest and JobExecution

Calling `OrchestrationService.execute(request: JobRequest)` launches a job. A `JobRequest` carries:

- `profile` — the `JobProfile` to launch
- `command` / `args` — optional command and argument overrides (provider support varies — see Configuring Namazu Conductor Providers)
- `environment` — optional environment variable overrides
- `placement` — a list of `JobPlacement` hints
- `scope` — a list of `JobScope` overrides

`execute()` returns a `JobExecution`: an `id`, a `status` (see below), a list of `endpoints` the job is reachable on (each a host/port/protocol triple), and an opaque provider-specific `details` payload. Use `getFutureForStatus()` / `getStageForStatus()` to wait for a job to reach a particular `JobStatus`, `listExecutions()` to see everything currently tracked, and `stop()` to terminate a job.

> Note
>
> Hold on to the exact `JobExecution` returned by `execute()` (or one derived from it via `getFutureForStatus`/`getStageForStatus`) if you plan to call `streamStdio()` later. On EdgeGap and ECS, the credential needed to open a stdio session travels on that object's `details` field and isn't recoverable from `listExecutions()`.

JobStatus

Every job reports one of four statuses: `PENDING` (accepted, not yet running), `RUNNING`, `COMPLETED` (exited or was stopped normally), or `FAILED`.

JobPlacement

`JobPlacement` is a hint about where a job should run. A `JobRequest` can carry any number of them; each provider honors the ones it understands and silently ignores the rest, so the same request can be sent to multiple providers without branching your code per platform.

- `RegionPlacement(id)` — a provider-native region/zone identifier
- `IpPlacement(ip)` — a target IP address
- `LatitudeLongitudePlacement(latitude, longitude)` — a geographic coordinate

See Configuring Namazu Conductor Providers for which placement types each provider actually acts on.

JobScope

`JobScope` overrides where, within a provider, a job is launched — for providers that have such a concept.

- `NamespaceScope(namespace)` — Kubernetes; overrides the default namespace a workload is created in
- `ClusterScope(cluster)` — ECS; overrides the default cluster a task is launched into

Providers without an equivalent concept — EdgeGap, whose scoping is fully determined by the profile 's app/version, and Multiplay — silently ignore any `JobScope` entries.

Related Pages

- Configuring Namazu Conductor Providers — attributes, placement/scope support, and provider-specific behavior for EdgeGap, ECS, Kubernetes, and Multiplay
- Streaming Job Stdio in Namazu Conductor — how `streamStdio()` works natively on Kubernetes and via the `namazu-stdio-bridge` sidecar on EdgeGap/ECS
- Namazu Conductor Admin API — the cross-provider superuser REST API and dashboard panel