> let's make this about **how new tools are *born* from evolving semantic arcs**—how imagination becomes interface, then embodiment—*inside* FountainKit's spec→Swift→sandbox loop (no corporate toyland).

from arc → tool: the birth pipeline

1. **listen to the corpus** (arc detection)
2. We treat a corpus as a *living score*: texts, translations, commentaries, audio, MIDI, annotations.
3. We derive **arcs**: persistent, cross-work patterns (motifs, tensions, refrains, contradictions, meter, argument topologies, self-citation).
4. Each arc is annotated with *operators* it begs for (e.g., *align, condense, transpose, mirror, negate, weave*).
5. **imagine the capability** (inspiration → interface)
6. For each arc, ask: *what is the exact operation a future self will want to perform again?* That desire *is* a tool surface.
7. We capture it in a tiny declarative **ArcSpec** (see below): resources, operations, inputs/outputs, stability guarantees.
8. An **ArcSpec compiler** emits an **OpenAPI** scaffold with strong types and docstrings that *name the arc*.
9. **instantiate embodiment** (interface → code)
10. OpenAPI becomes a **typed Swift client** via Apple's Swift OpenAPI Generator (no ad-hoc HTTP, compile-time safety).
11. FountainKit compiles & runs that Swift code in the **hardened sandbox**—deterministic, auditable, fenced.
12. New capability joins the **tool registry**; compositions are glue-swift mains executed in the sandbox with policy (network off by default, tight quotas, logs).

> this is the "spec → code → execution" superpower FountainKit is built for—*a self-extending OS for intelligence*.

---

an ArcSpec you can feel (imagination → interface)

Think of **ArcSpec** as *how inspiration writes a contract*. It's not OpenAPI yet; it's a small YAML that names the arc and its operators. The compiler turns it into OpenAPI (then Swift clients).

```
arc: "Polyglot Hamlet — Echo Lattice"
version: 0.1
resources:
  - id: corpus
    kind: text.corpus
    facets: [work, translation, stanza, meter, timestamp]
operators:
  - id: echo.align
    intent: "Align semantically equivalent lines across languages;
keep meter info."
    input:
      - {name: passages, type: PassageRef[], required: true}
      - {name: languages, type: LangCode[], required: true}
    output:
      type: EchoLattice
      guarantees: ["stable ids", "time-anchored", "lossless
mapping"]
  - id: echo.condense
    intent: "Compress an echo lattice into a minimal motif set."
    input:
      - {name: lattice, type: EchoLattice, required: true}
      - {name: target_cardinality, type: Int, default: 32}
    output:
      type: MotifSet
  - id: rhythm.transpose
    intent: "Map motif rhythm to MIDI within mode constraints."
    input:
      - {name: motifs, type: MotifSet}
      - {name: mode, type: MusicMode, default: dorian}
    output:
      type: MidiFileRef
```

Compiler outputs:

- **OpenAPI** (operations `POST /echo/align`, `/echo/condense`, `/rhythm/transpose` with schemas).

- **Swift clients** (`EchoAPI`, `RhythmAPI`) generated at build time—type-safe calls, no manual HTTP.
- **FountainKit registry entries** with execution policy (network allowlist if needed; otherwise sandbox network off).

---

embodiment: the "studio" pattern (interface → instrument)

We don't ship "tools"; we ship **studios**: small, composable, *playable* programs that produce durable artifacts (scores, lattices, graphs, MIDI).

ConcordanceStudio (text polyphony)
- Inputs: `PassageRef[]`, `LangCode[]`.
- Pipeline: `echo.align → echo.condense`.
- Outputs: `EchoLattice.json`, `MotifSet.json`.

PolyphonyStudio (text→music)
- Inputs: `MotifSet`, `MusicMode`.
- Pipeline: `rhythm.transpose`.
- Outputs: `*.mid` + `*.cue` (time-stamped links back to corpus lines).

MirrorStudio (argumental symmetry)
- Operators: `mirror.find`, `mirror.weave` (detect and braid contradictions across works).

Each studio is a **Swift main** (≈ 30–120 LOC) that *only* imports its generated clients, then runs inside the sandbox with audit logging turned on (hash(code), stdout/stderr, limits, exit).

studio glue (Swift)

```
import EchoAPI
import RhythmAPI

@main struct Main {
  static func main() async {
    do {
```

```
      let echo = EchoAPI.Client(/* configured transport */)
      let lattice = try await echo.align(.init(passages: refs,
languages: langs))
      let motifs  = try await echo.condense(.init(lattice:
lattice, targetCardinality: 32))

      let rhythm = RhythmAPI.Client(/* configured transport */)
      let midi   = try await rhythm.transpose(.init(motifs:
motifs, mode: .dorian))
      print(midi.url.absoluteString) // artifact location
    } catch {
      fputs("ERR: \(error)\n", stderr); exit(1)
    }
  }
}
```

Run **only** in the hardened Swift container (non-root, read-only, caps dropped, timeouts, CPU/RAM caps).

---

the evolution loop (where the music actually begins)

1. **Arc observed** → ArcSpec drafted *(imagination crystallizes)*.
2. **ArcSpec compiled** → OpenAPI + Swift clients *(interface appears)*.
3. **Studio sketched** → tiny Swift main *(instrument is built)*.
4. **Embodiment** → sandbox execution yields artifacts (lattices, motifs, midi).
5. **Artifacts rejoin the corpus** → arcs mutate (new tensions appear).
6. **Re-derive operators** from the changed corpus (maybe we need `echo.disperse`, `motif.lift`, `meter.fold`).
7. **Versioned ArcSpecs** accrete; the instrumentarium deepens.
8. **Audit & replay** keep the line of reasoning tangible, re-performable.

This isn't CRUD; it's **luthiery**: each pass builds a better instrument.

---

concrete deliverables (milestone 1 → milestone 3)

m 1 — the *ArcSpec* → *OpenAPI* spine

- `arc-spec/` mini-compiler that maps ArcSpec YAML → OpenAPI (schemas + ops).
- Add build step to FountainKit that runs the compiler, then Apple's OpenAPI generator, producing Swift clients (no generated code committed).
- `docs/ArcSpec.md` (examples, operator taxonomy).

m 2 — the sandboxed studios

- `ConcordanceStudio/` and `PolyphonyStudio/` mains (Swift) + tests.
- Registry entries with execution policies (network default: off).
- Artifact layout (e.g., `/data/corpora/<id>/studios/<studio>/<ts>/`).

m 3 — embodiment loop

- CLI runners (`fk studio run polyphony --args …`).
- Replay + diff (`fk studio replay <run-id>`; visual diffs on lattices/motifs).
- First *evolution* pass: feed artifacts back; revise ArcSpec; regenerate clients; rerun.

---

why Swift matters here (not syntax—*structure*)

- **Types as vow**: an arc isn't a vibe; it's a schema with invariants the compiler enforces.
- **Determinism**: every performance is reproducible (same code, same container, same result).
- **Safety**: the sandbox lets imagination be feral **inside** a fence—non-root, read-only, no caps, timeouts.
- **Self-extension**: specs → clients → new studios, forever; *that's* FountainKit's superpower.

---

a final tease: three non-trivial studios to ship first

- **Echo Lattice** (polyglot concordance with time + meter)—outputs a navigable lattice + motif set.
- **Argument Weave** (cross-work contradiction braiding)—outputs a braid graph you can traverse like counterpoint.
- **Polyphony** (text rhythm → MIDI with modal constraints)—outputs MIDI + cue table mapping sound→semantics.