SEAGLASS FOUNDRY
Technical Brief

ForgeData

The data-intelligence layer of the Forge suite

Cost-aware routing, a portable single-file catalog, and a 49-tool MCP surface that let an agent ask for data by intent and get back the best-fit correct dataset — ranked by coverage and resolution, with access cost as the deciding tiebreaker — across thirteen heterogeneous source types.

Release	Initial release (0.1.0)
Component of	The Forge suite (ForgeGIS, ForgeMind, ForgeGIS Studio)
Runtime	Single fat JAR, Java 17+, no native install step, no daemon
Interface	Model Context Protocol over stdio JSON-RPC, plus a CLI
Audience	Technical evaluators and GIS / platform engineers
Document date	June 2026

Distribution note. ForgeData is bundled with the Forge suite and is not sold separately. Its license is not yet finalized; nothing in this document should be read as a grant of rights. Contact the maintainer before redistributing ForgeData or any catalog bundle.

Contents

1. Executive summary

A modern geospatial workload almost never lives in one place. A single project routinely pulls elevation from an SRTM mirror, imagery from a Sentinel-2 STAC catalog on S3, hydrography from an internal PostGIS, building footprints from an ArcGIS REST service, bathymetry from NOAA, point clouds from local LAZ files, and pre-computed slope rasters from last week's pipeline run. A human analyst in QGIS handles this by knowing which file to open. An automated system cannot.

ForgeData is the component of the Forge suite that answers, programmatically, on every request: which datasets cover this area, which are at the right resolution for what is about to be computed, which is cheapest to actually read, and whether the suite already computed the result. **ForgeGIS computes. ForgeMind reasons. ForgeGIS Studio renders. ForgeData decides which data to use and how to get it** — solved once, in one place, so the other components stay focused on what each does best.

1.1 What you would otherwise build yourself

If your team adopted a raw stack — GDAL, PostGIS, a STAC index, some cloud COGs — and wired an agent or a pipeline on top, you would end up hand-rolling four things. ForgeData is those four things, built once and validated. The rest of this brief is the mechanism behind each; the section pointers show where the proof lives.

> 1. **Configuration that cannot drift from the data it describes.** The catalog and the recipes that refresh it live in one GeoPackage file — no companion `sources.yml` to fall out of sync, no secrets baked in. (§3)
> 2. **A binding from operations to their data requirements.** A registry declares, for each of 47 spatial operations, the category, resolution ceiling, and band count it needs — so routing is a lookup, not a pile of per-operation conditionals you maintain by hand. (§5)
> 3. **Fidelity-first routing across heterogeneous sources.** Candidates are ranked by coverage, then resolution, then extent-fit, with access cost as the deciding tiebreaker, then deduplicated to a minimal covering set — the logic teams usually reinvent badly as hardcoded paths and ad-hoc fallbacks. (§4, §6)
> 4. **Memory of what was already computed.** Derived products register back into the catalog and are preferred over recomputation, so a long-running workflow gets faster the longer it runs instead of recomputing the same slope raster forever. (§9)

One honest limit worth stating up front: routing quality **degrades gracefully** with sparse metadata rather than failing. Coverage and resolution are the primary sort keys, so thin or unknown metadata yields a coarser ranking, not an empty result — unknown-resolution rows are deliberately kept (they are only excluded when a caller sets an explicit hard fidelity floor, which by definition cannot be met by unknown fidelity), and imagery with an unknown band count falls back to an unfiltered category query so single-band sources still route. Richer metadata sharpens the ranking; it is not a precondition for getting an answer.

What ships in the initial release

Cost-aware routing. A five-tier access model and a 47-operation registry rank candidate datasets by coverage, then resolution, then extent-fit, with access cost as the final tiebreaker.

A portable single-file catalog. All state — datasets and the recipes that refresh them — lives in one standard GeoPackage. Safe to copy, ship, or commit.

Thirteen ingester types. Local directories, STAC, WCS, WFS, PostGIS, ArcGIS REST, Overpass, OpenTopography, NOAA, Copernicus CDS, TIGER/Line roads and addresses, and Mapzen Terrarium — one uniform query interface.

A 49-tool MCP surface. Stdio JSON-RPC tools for querying, routing, publishing, derived-product memory, async sync, and health checks — the same surface ForgeMind and Studio consume.

Derived-product memory. Pre-computed results register back into the catalog and are preferred over recomputation.

2. Where ForgeData sits in the suite

The routing work could in principle live inside ForgeGIS, ForgeMind, or Studio. It deliberately does not. Each of those components has a sharp job, and folding data-intelligence concerns into any of them would blur that boundary.

2.1 The separation of concerns

Component	Its job	Why routing does not belong here
ForgeGIS	Geospatial compute engine — raster I/O, terrain, spectral indices, point clouds, GPU acceleration.	A compute library should stay focused on pixels and geometries. The moment it grows a catalog and sync schedulers it stops being embeddable.
ForgeMind	Agent orchestrator — reasoning across tools.	If routing lived here, every other consumer would have to go through ForgeMind or reinvent the catalog locally.
ForgeGIS Studio	Browser-facing UI — catalog browser, source management, raster preview.	Its data screens are a thin frontend over the same catalog. Putting storage inside Studio couples browser concerns to data concerns.
ForgeData	Data intelligence — catalog, cost routing, sync, derived-product memory, window extraction.	This is the home for the cross-cutting problem, so it can evolve independently of every consumer.

2.2 How the pieces connect

ForgeMind and Studio talk to ForgeData over MCP — the same stdio JSON-RPC surface any third-party MCP client could use. ForgeGIS is consumed by ForgeData **as a Java library** for the actual data reads, so windowed-raster extraction and elevation queries happen in-process with no network

hop. Underlying sources stay exactly where they are; ForgeData never requires the customer to move data into a new store.

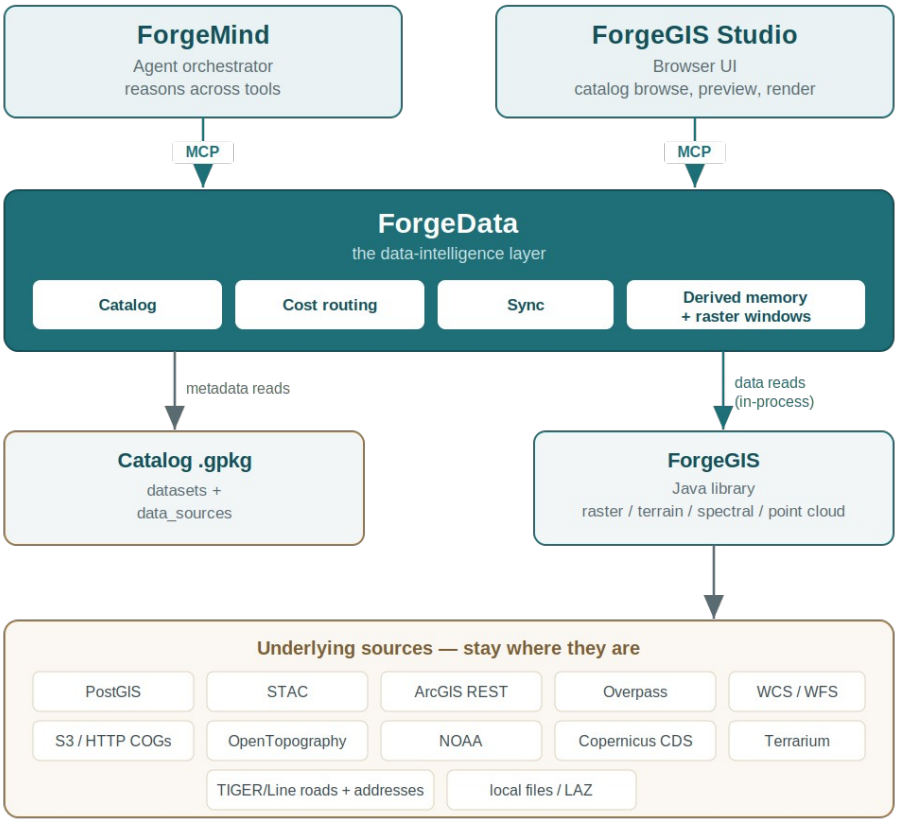

Figure 1. Consumers reach ForgeData over MCP; ForgeData reads the catalog for metadata and drives ForgeGIS in-process for data.

3. The catalog: configuration as data

The unit of state in ForgeData is a single `.gpkg` file — a GeoPackage, which is a SQLite database with standard GeoPackage metadata tables. ForgeData adds two tables of its own. There is no separate inventory file, no manifest, no lock file. Copy the `.gpkg` to another machine and ForgeData there sees exactly the same catalog.

3.1 Two tables

Table	One row per	Read by	Written by
`datasets`	Logical dataset you can query (a tile, a layer, a derived raster).	Every query and routing tool.	Ingesters, insert_dataset, register_derived_product, publish_file.
`data_sources`	Place ForgeData fetches from (a directory, an API, a database).	Sync and preview tools, the auto-sync scheduler.	add-source (CLI) / add_data_source

			(MCP).

The key design decision is that **the catalog IS the configuration**. The data-source recipes live inside the same file as the inventory, in a separate table — so there is no companion `sources.yml` to drift out of sync. A single sync run typically inserts many `datasets` rows for one `data_sources` row.

> **Why secrets are safe to ship**
>
> **API keys never live in the catalog.** The catalog stores only the environment-variable name that holds a key or password — never the secret itself. That is what makes a `.gpkg` safe to copy, ship to a field laptop or air-gapped site, or commit to an internal repository. The same property makes pre-curated catalog bundles a first-class distribution artifact.

3.2 The dataset row

A handful of fields on each dataset drive every downstream decision:

Field	Drives
`category`	Routing — find_optimal_for_operation filters by category (ELEVATION, IMAGERY, BATHYMETRY, VECTOR, LIDAR, and the derived VISUALIZATION / ANALYTICAL, among others).
`format`	Reader selection (GeoTIFF, COG, SRTM_HGT, NetCDF, JPEG2000, Shapefile, GeoJSON, LAZ, and more).
`bbox_west/south/ east/north`	Spatial intersection — four indexed columns back the bbox range query.
`file_path`	Access tier — a bare path is LOCAL; postgis://, arcgis://, http(s)://, s3://, wcs://, wfs:// map to REMOTE tiers. managed:// is on-disk managed storage and resolves to a LOCAL tier.
`resolution_m`	Resolution filtering for operations that need a minimum fidelity.
`band_count`	Spectral-operation gating (an op may require N or more bands).

3.3 Catalog integrity and concurrency

Each MCP tool call is its own JDBC transaction; the system does not promise multi-call atomicity. Catalogs open in SQLite **WAL mode** with a busy timeout, so multiple processes on one host can share a file — readers run concurrently with a single writer, and a second writer waits on the busy timeout rather than failing outright. What you do not get is two writers progressing at once, and WAL is local-disk only (no network filesystems).

A `migrateSchema` step runs on every open and retrofits columns onto catalogs created by older versions; it is forward-only, with no downgrade path. A partial UNIQUE index on the content hash defends against duplicate-row races even across two JVMs writing the same file.

4. Cost-aware routing

Cost-aware routing is ForgeData's core differentiator. `find_optimal_for_operation` lets an agent name an operation and an area of interest and receive a ranked, costed, deduplicated list of catalog entries that can serve it — ordered by coverage and resolution, with access cost as the deciding tiebreaker — with no hardcoded paths.

4.1 The five access tiers

Every dataset is scored at request time into one of five tiers, ordered cheapest to most expensive:

Tier	Meaning	When a dataset gets it
`LOCAL_HOT`	Recently loaded; warm in ForgeGIS CPU/GPU memory.	Loaded within the last cache.hot_ttl_seconds (default 600 s).
`LOCAL_COLD`	On-disk local file; fast but needs I/O.	Local files not recently touched.
`REMOTE_DB`	PostGIS. Bimodal: ~10 ms point lookup, 100 + mb·60 ms windowed.	postgis:// scheme.
`REMOTE_COG`	HTTP/S3 range-queryable raster.	http(s):// or s3:// — scheme-based, any such path.
`REMOTE_WCS`	OGC WCS and other on-demand HTTP services.	wcs://, arcgis://, wfs:// schemes.

> **Self-calibrating remote cost**
> After the first read of a remote dataset, ForgeData stores the measured wall-clock latency and uses it on subsequent calls instead of the static formula. The calibration is persisted in the catalog's `observed_load_ms` column, so it survives restarts, and `estimate_access_cost` reports whether a given estimate is `observed` or `heuristic`. The cost signal sharpens the longer the suite runs.

4.2 How candidates are gathered, then ranked

`find_optimal_for_operation` runs two phases, both fed through one comparator.

1. **Phase 1 — local derived products.** Locally-available derived products whose derived operation is the exact operation requested are surfaced first, ahead of raw source data. Remote derived products are not injected.
2. **Phase 2 — raw source datasets.** Non-derived rows in the operation's required category whose bbox intersects the AOI, after filtering on resolution ceiling, optional hard fidelity floor, minimum band count, and an optional vector feature-class hint.

Both phases sort by the same ranking comparator, in this precise order:

3. **AOI coverage — descending.** The dataset covering more of your bbox wins.
4. **Resolution — ascending.** When coverage ties, finer pixels win.

5. **Extent-fit — descending.** When coverage and resolution tie, the footprint closest to the AOI size wins.
6. **Access tier — final tiebreaker only.** Locality breaks an otherwise-perfect tie; it never overrides coverage or resolution.

> **A deliberate non-obvious choice**
>
> `preferWindowedRead` is ***not*** used to filter or rank — it only feeds the reported load-time heuristic. So an `SRTM_HGT` tile, which cannot do windowed reads, does *not* drop out for a windowed-preferring operation such as VIEWSHED. It ranks by coverage and resolution like anything else and simply reads the full tile. This avoids silently excluding otherwise-ideal data on a property that is a performance hint, not a correctness gate.

After sorting, a multi-resolution coverage **dedup** drops any raster tile fully covered by higher-ranked tiles, so four redundant tiles collapse to the minimal covering set. The dedup is skipped for `VECTOR`, where layers sharing an extent are distinct content, not redundant tiles.

4.3 Worked example

AOI: a 1° × 1° bbox that one SRTM tile fully covers. The catalog holds 36 Arizona SRTM tiles (one intersecting), an Arizona-wide Copernicus DEM COG (REMOTE_COG, 30 m), and a locally-computed slope COG for that exact bbox (derived `SLOPE`).

Query	Result order	Why
`…, "SLOPE")`	slope COG → SRTM tile → Copernicus DEM	Phase 1 surfaces the local derived SLOPE first. Phase 2 raw survivors tie on coverage (1.0) and resolution (30 m), so extent-fit breaks it: the 1°×1° SRTM tile fits the AOI better than the Arizona-wide DEM. Tier never consulted.
`…, "VIEWSHED")`	SRTM tile → Copernicus DEM	Phase 1 empty (no derived VIEWSHED). The slope COG, being derived, is also excluded from Phase 2. The SRTM tile stays valid despite VIEWSHED preferring windowed reads — it just reads the full tile.

Coverage fraction is always *your query bbox covered by the dataset*, not the reverse. A fraction below 1.0 means you need multiple datasets to cover the AOI; the ranker handles that, and the caller unions the top N. `estimate_access_cost(dataset_id)` returns the full cost profile for a single entry without running a search.

5. The operation registry

Routing is driven by a registry of 47 spatial operations across nine groups. Each operation declares its data requirements: required category, maximum recommended resolution, whether windowed

reads are preferred, and any minimum band count. Adding an operation is a two-line change in one file; no other site in the codebase needs updating.

Group	Count	Required category	Notable requirements
Elevation	9	ELEVATION	SLOPE / ASPECT / VIEWSHED / ROUGHNESS / TPI cap at 30 m; point, region, profile and hillshade uncapped.
Hydrology	9	ELEVATION	Prefer windowed reads; most cap at 30 m; SINK_FILL uncapped.
Spectral indices	11	IMAGERY	Minimum band count 2–4; most cap at 30 m; BSI requires 10 m and 4 bands.
Focal statistics	4	ELEVATION	Windowed-preferred; no resolution cap.
Interpolation	3	VECTOR	Input is point data (IDW, KRIGING, SPLINE).
Bathymetry	3	BATHYMETRY	Profile, water depth, dredge.
Vector	4	VECTOR	Includes BUFFER, ForgeMind's inline-geometry buffer with a VECTOR output override.
Imagery	1	IMAGERY	IMAGERY_CLASSIFY.
LiDAR	3	LIDAR	Windowed preference stays false until COPC range reads land.

> **Output category vs input category**
> An operation's **output** category is what the product *is*, not what it consumes. SLOPE consumes ELEVATION but its product is `ANALYTICAL`; HILLSHADE is `VISUALIZATION`; SINK_FILL returns a corrected DEM and stays `ELEVATION`. This output mapping is what `register_derived_product` auto-stamps, so downstream routing re-finds products correctly.

6. Heterogeneous sources, one interface

Thirteen source-type implementations cover the common public and private geospatial sources. Adding a new type is one Java class implementing one interface, a validator, and CLI wiring. The set is fixed at compile time — intentionally not runtime-configurable.

Source type	What it ingests	Auth
`LOCAL_DIR`	A directory of tiles / vectors / point clouds; optional filesystem-watch auto-pickup.	None
`STAC`	SpatioTemporal Asset Catalog collections (e.g. Sentinel-2 on S3).	Per-source
`WCS`	OGC Web Coverage Service — on-demand windowed raster.	Optional

WFS	OGC Web Feature Service — GeoJSON output required at read time.	Optional token
POSTGIS	PostGIS vector + raster; server-side spatial filtering.	Password env var
ARCGIS_REST	Esri FeatureServer / ImageServer layers.	Optional token
OVERPASS	OpenStreetMap features via the Overpass API (tile-grid).	None
OPENTOPO_API	OpenTopography elevation (tile-grid; daily rate limits).	API key env var
NOAA_API	NOAA datasets (tile-grid).	Optional
COPERNICUS_CDS	Copernicus Climate Data Store (single-request, hash-named output).	API key env var
TIGER_ROADS	TIGER/Line All Roads — named-road lookup and bbox queries.	None
TIGER_ADDRFEAT	TIGER/Line address-range edges — address geocoding.	None
TERRARIUM	Mapzen Terrarium terrain tiles (no-API-key elevation bootstrap).	None

6.1 Preview before you sync

`preview_data_source` returns the exact `DatasetEntry` rows a sync would insert, **without** touching the catalog or downloading bytes — each preview item carries a fully-populated entry plus an `alreadyCataloged` flag. Semantics adapt to the ingester: catalog-style sources enumerate with a true count; tile-grid sources compute the deterministic tile list with zero HTTP; single-request sources synthesize the hash-derived filename the sync would produce.

6.2 Sync, auto-sync, and the bootstrap hint

Sync runs manually via CLI or MCP, or on a schedule via `AutoSyncScheduler`. A `LOCAL_DIR` source with an interval set also gets a filesystem `WatchService` for near-real-time pickup of dropped files and immediate pruning of deleted ones. Only one instance per data root owns the scheduler, enforced by an advisory lock.

When routing finds nothing for an AOI, the empty result is not a dead end: it carries an `empty_reason` and, for coverage gaps, `acquisition_suggestions` — enabled registered sources whose sync could acquire the area, each naming the next tool call — plus a `bootstrap_hint` naming the no-API-key source type (TERRARIUM for elevation, OVERPASS for vector) when nothing registered can help.

7. The MCP tool surface

ForgeData exposes **49 MCP tools** over stdio JSON-RPC. All tools return JSON; on the wire, local and `managed://` file paths are externalized to absolute filesystem paths so external readers such as ForgeGIS can open them, while remote URIs are emitted unchanged. The same surface backs ForgeMind, Studio, and any third-party MCP client.

Group	Count	Representative tools
Catalog queries	9	list_datasets, get_dataset, get_status, search_by_bbox / _category / _tag / _source
Catalog management	4	insert_dataset, patch_dataset, patch_dataset_category, delete_dataset
Elevation	3	elevation_at_point, elevation_in_region, elevation_profile
Raster window extraction	1	read_raster_window
LiDAR	2	lidar_elevation_at_point, lidar_density_in_region
Vector	5	vector_features_in_bbox, lookup_road_by_name, roads_in_region, roads_near_point, geocode_address
Compute-aware routing	2	find_optimal_for_operation, estimate_access_cost
Data source management	7	list / add / update / remove / preview / sync_data_source, sync_all_sources
Source-type discovery	3	list_source_types, get_source_type_schema, check_env_var
Async sync	5	submit_sync_*, get_sync_status, list_sync_jobs, cancel_sync
Derived products	2	register_derived_product, find_derived_products
User publish flow	5	probe_file, publish_file, probe_vector, publish_vector, find_by_content_hash
Catalog presence	1	verify_dataset_presence
Total	**49**	

7.1 Structured, classified errors

Every tool failure returns a structured envelope — `{ "error_code": "…", "message": "…" }` — routed through a shared classifier. Common codes include `source_file_missing`, `rate_limited`, `timeout`, `bbox_invalid`, `aoi_out_of_coverage`, and `operation_not_supported`. Message text is inspected first, so an exception can be upgraded by its message. Tools layer their own codes on top — `read_raster_window` adds six, including `windowed_read_unsupported` and `bbox_outside_dataset`; `elevation_in_region` adds `aoi_too_large` with the requested and maximum areas attached.

8. Reading data: the query paths

Once routing has chosen a dataset, ForgeData reads it. The read paths are summarized below; each is specified in full, with exact parameters, response shapes, and error codes, in the suite reference documentation. This section gives the shape and one design decision worth calling out, not the field-level spec.

Path	Tools	What it does	Reference
Elevation	`elevation_at_point / _in_region / _profile`	Point, region stats, and straight-line profile via the cost model; a separate windowed extract bypasses ranking.	querying-elevation
Raster windows	`read_raster_window`	AOI sub-window to a local GeoTIFF; extractability gated by category + format/scheme; on-disk LRU extract cache.	raster-windows
Vector	`vector_features_in_bbox + road / geocode tools`	One RFC 7946 FeatureCollection, dispatched by scheme (PostGIS, ArcGIS, WFS, local GeoJSON / shapefile); plus named-road, roads-in-region / near-point, and TIGER address geocoding.	querying-vector
LiDAR	`lidar_elevation_at_point / _density_in_region`	Ground elevation and point density over .las / .laz via laszip4j; COPC / EPT / remote pending upstream range reads.	querying-lidar

The one design decision worth surfacing here, because it changes results rather than just performance, is how elevation handles water:

> **Honest water, from one seam**
> DEMs encode open water as a value, not as absence: SRTM flattens ocean to exactly 0 m — a real stored value that reads as genuine ground and cascades into slope, viewshed, and profile. ForgeData fixes this at the shared elevation read seam: when the catalog holds a `water-mask`-tagged VECTOR layer, over-water cells are recoded to the tile's NoData sentinel before sampling. The decision is geometry-driven, so it is source-agnostic — SRTM, Copernicus, 3DEP, Terrarium, remote WCS all behave identically — and it is a strict no-op until a water layer is present. Genuine sea-level land is left untouched.

9. Memory: derived products and publishing

9.1 Derived-product memory

A derived product is a dataset computed from another — slope from elevation, NDVI from imagery, a watershed delineation. Registering one sets `is_derived`, a `parent_id` pointing at the source, a `derived_op`, and an opaque `derived_params` JSON string. The payoff is that future routing prefers the pre-computed product over recomputing from source — and over a long-running project this compounds into real speedups.

An *unrecognized* derived operation still registers — stamped ANALYTICAL with a warning — so a new ForgeMind compiler op never blocks the layer from loading. Adding the op to the registry later enables input routing for it; registration works regardless. The three reproducibility anchors are `parent_id` (combine with the source content hash for byte-level reproducibility), `derived_op`, and `derived_params`.

9.2 The publish flow

`publish_file` lets a user or agent contribute a local raster into a shared catalog in a single call: it probes the file's georeferencing internally, atomically copies it into hash-sharded managed storage, computes SHA-256, and inserts a catalog row with provenance. It is idempotent — a duplicate content hash returns the existing entry as a silent dedup, with no error. `publish_vector` is the GeoJSON sibling, storing bytes unchanged. `probe_file` is an optional, side-effect-free pre-flight, never required.

> **Atomicity in seven steps**
>
> Compute SHA-256 → acquire an in-process hash lock → check for an existing row (and stop if found) → probe and apply hints → copy to a temp file and atomically rename into `managed://ab/cd/{hash}.{ext}` → insert the catalog row → release the lock. The partial UNIQUE index on the content hash defends against races that slip past the in-process lock — even two JVMs publishing to the same file — and SQLite makes one insert lose cleanly.

Managed-storage paths are stored as opaque, portable `managed://` URIs and externalized to absolute paths on the wire. The managed root is reported by `get_status` so a consumer driving ForgeGIS over a published path knows the directory to add to its read roots.

10. Running ForgeData

10.1 Runtime and configuration

ForgeData is a single fat JAR on Java 17+ — no native install step (the JAR self-contains its native dependencies, such as the bundled per-platform SQLite driver), no daemon to install, no schema migrations to run by hand. The ForgeGIS GPU path is optional and falls back to CPU automatically. It logs to stderr so the MCP JSON-RPC stream on stdout stays clean. Configuration is optional: a `forgedata.yml` file (global and per-catalog-bundle) tunes caches, timeouts, storage roots, and extract limits, and every key has an environment-variable override. Precedence, highest first: env var, then catalog-bundle yml, then global yml, then built-in default.

> **The path-resolution gotcha worth knowing up front**
>
> A relative `download.dir` resolves against the JVM working directory, not the JAR. When ForgeData is spawned as an MCP subprocess (Claude Desktop, ForgeMind, Studio), that CWD is inherited from the parent — so for any non-repo-root deployment, set `FORGEDATA_DOWNLOAD_DIR` to an absolute path. A catalog-bundle's relative `data.root`, by contrast, is anchored to the bundle directory itself — the load-bearing trick that makes self-contained, relocatable data packs work regardless of launch directory.

10.2 Catalog bundles and federation

A bundle is a `catalog.gpkg` plus the `data/` it references and an optional `forgedata.yml`; bundle-relative `data.root` keeps it self-contained, so it ships as a tarball a recipient drops in and queries. For larger deployments, a federated `catalog set` makes several bucket catalogs invisible

to consumers: reads fan out across every bucket and merge with namespaced IDs, while writes target the primary. An agent asks for data by intent and ForgeData routes across everything it holds.

10.3 Health checks

Two tools keep a catalog honest. `verify_dataset_presence` is a bulk stat-check that stamps a verification timestamp on local rows and, optionally, re-hashes present files to detect corruption or out-of-band replacement. The CLI `doctor` command is diagnostic-only — it never moves or deletes anything — and runs two scans: orphan subdirectories (bytes on disk no catalog row references) and orphan rows (catalog rows whose files are missing), exiting non-zero when any orphan rows are found.

11. Where a routing layer earns its place

Each adjacent tool solves part of the data-access problem well and leaves the rest to the integrator. The gap each one leaves is, in aggregate, the four mechanisms named in §1.1.

Tool	Solves	Leaves to the integrator
PostGIS	Stores and processes spatial data.	Routing across heterogeneous, non-PostGIS sources.
STAC catalogs	Index assets and metadata.	Modeling what is warm in memory or what an operation actually requires.
GeoServer	Hosts OGC services.	Decision-making for an agent; it serves maps to clients.
Raw cloud COGs	Fast, cheap storage at rest.	Any discovery or ranking layer at all.

Adopting any of these alone means hand-rolling the catalog, the operation-to-requirement binding, the fidelity-first ranking, and the derived-product memory — the four things §1.1 opened with. ForgeData is that layer, built once. The procurement-level case for the suite as a whole is made in the companion sales brief; this document's claim is narrower and verifiable: the mechanism in §§3–9 is the layer you would otherwise build and maintain yourself.

12. Specifications at a glance

Dimension	Initial release
Catalog format	Single GeoPackage (.gpkg) — SQLite; readable in QGIS / DB Browser.
Catalog tables	datasets (queryable inventory) + data_sources (refresh recipes).
MCP tools	49, over stdio JSON-RPC.
CLI subcommands	sources, source-types, add-source, sync, lookup-road, roads-in-region, roads-near-point, geocode-address, doctor.
Spatial operations	47, across 9 groups, in the operation registry.

Ingester / source types	13 (compile-time set).
Access tiers	5 — LOCAL_HOT, LOCAL_COLD, REMOTE_DB, REMOTE_COG, REMOTE_WCS.
Remote cost calibration	Self-calibrating on measured latency; persisted in observed_load_ms.
Concurrency model	WAL mode; concurrent readers, single writer, busy-timeout waits.
Runtime	One fat JAR, Java 17+, no native install step (self-contained native deps), no daemon, stderr-only logging.
Integration	ForgeMind + Studio over MCP; ForgeGIS as an in-process Java library.

Further reading. The full reference set — MCP tool specs, the `forgedata.yml` schema, the environment-variable table, catalog DDL, ingester schemas, the operation registry, dataset-field semantics, and the structured error taxonomy — ships with the suite documentation, alongside architecture notes and end-to-end tutorials. Per-source data-attribution requirements for redistributing bundled data are documented per catalog bundle. Licensing is summarized on the cover.