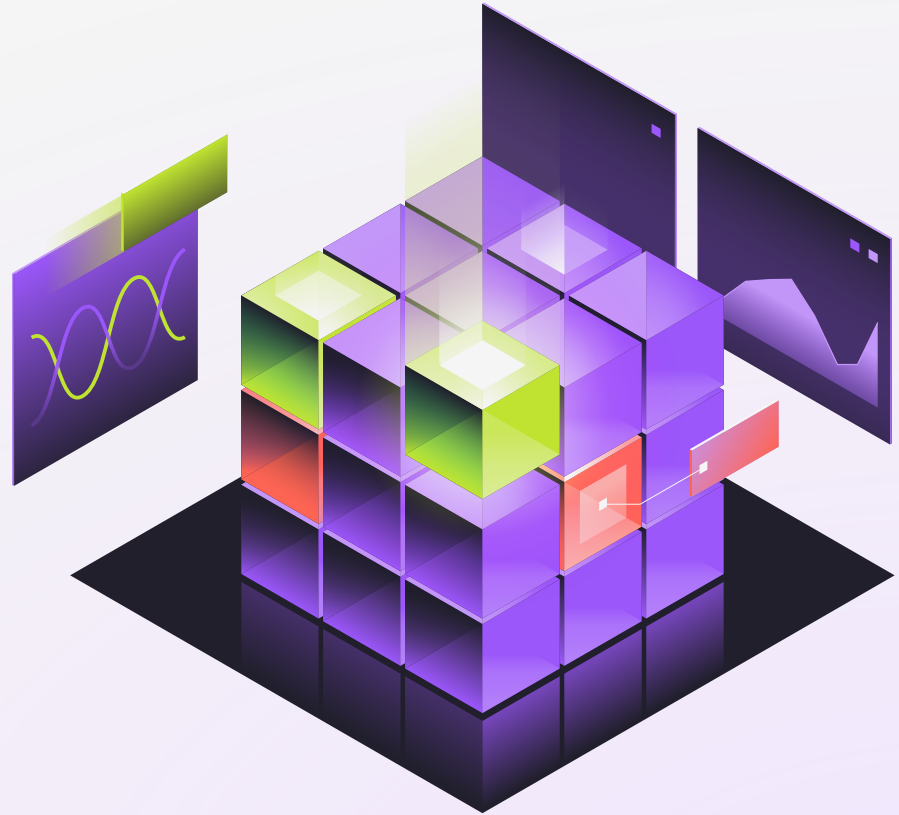
earthmover

virtual chunks at earthmover: from theory to production



ESIP Cloud Computing Cluster

Tom Nicholas - 05/15/2026

OUTLINE

what we'll cover

- Why?
- Definitions
  - What is a **virtual chunk** / **chunk manifest** / **virtual chunk container**?
- Implementations
  - Icechunk's manifest format
  - VirtualiZarr's manifest data structure
  - A comparison of manifest formats
- Usage
  - Virtual chunks as a platform primitive
  - Security considerations

VIRTUAL CHUNK

but why virtual chunks?

Object storage makes it easy to access data!

- Can fetch data from anywhere, not just the same system

Object storage makes it hard to change data!

- No edit-in-place operation, only full object overwrites
- `move` is really `create` then `delete` (expensive)

Object storage is a write-once, read-many system.

> *Primitive for referring to data at rest is a natural requirement in the cloud.*

DEFINITIONS

what abstractions are we talking about?

VIRTUAL CHUNK

a single reference

A **virtual chunk reference** is a lightweight reference to a single object living in arbitrary storage.

Always a single contiguous byte range, typically inside a HDF5, NetCDF, or GRIB file in object storage or on disk.

Tuple: `(url: str, byte_offset: int, byte_length: int)`

- Points to bytes that already exist somewhere else
- A storage optimization — no duplication of source data
- Lives at the **I/O layer**, not the data model

As an I/O concern, virtual chunks are **orthogonal to the Zarr spec**!

CHUNK MANIFEST

map logical space to physical

A **chunk manifest** maps the *logical data model* (e.g. a Zarr array's chunk grid) to *chunk references* — wherever the actual bytes live.

- Data model says: "I want chunk `[2, 0, 5]` of `temperature`"
- Manifest says: "those bytes are at `s3://archive/...` offset `48201`, length `8192`"

The manifest is an **abstraction layer:** you can change the physical layout or the logical layout independently of one another.

- Can swap native and virtual chunks out for each other - indistinguishable for the Zarr store consumer
- Move Zarr groups around without moving actual bytes on disk

CHUNK TYPES

manifests hold various physical representations

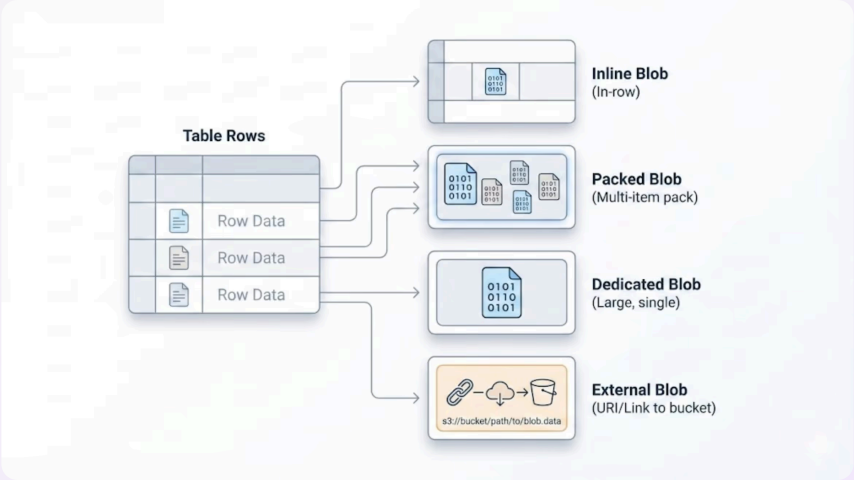
A single manifest can mix types of chunk references freely:

- **Virtual** — byte range in an external file
- **Native** — a standalone object in object storage
- **Sharded** — chunks packed inside a larger object
- **Inline** — bytes stored *inside the manifest itself* (for tiny chunks)

Choice is purely about storage optimization.

not unique to zarr!

Other cloud-native storage systems use same distinction, e.g. LanceDB



LanceDB's breakdown of blob types — from lancedb.com/blog/lance-blob-v2

VIRTUAL CHUNK CONTAINER

group refs by physical storage location

A **virtual chunk container (VCC)** is a grouping of co-located virtual chunks — typically those living in the same bucket, prefix, or permission boundary.

Generally use prefixes, for example `{"s3://archive/public/data/": <cloud provider and auth details>}`.

- Reason about storage **per-bucket**, not per-chunk
- Credentials, endpoint, region, and policy attach to the **container**
- A single dataset can have **many VCCs** spanning multiple clouds
- Permissions, credentials, and authentication become tractable

(Lance calls this a "location base", Delta Tables and Iceberg have something similar too.)

IMPLEMENTATIONS

examples: icechunk and virtualizarr

ICECHUNK MANIFESTS

a binary, paged, indexable format

Icechunk stores chunk manifests on-disk as flatbuffers files, paged into manifest files that are tracked by snapshot files.

- **Snapshot-scoped** — every commit has its own manifest set
- **Paged** — manifests can be split arbitrarily
- **Indexable** — only the manifest pages covering a query region are read
- **Dedup-friendly** — unchanged pages are shared across snapshots
- Supports **virtual**, **native**, and **inline** references
- Stores **last-modified time** as optional etag/checksum

ICECHUNK MANIFESTS

Rust enum defining chunk types, from `icechunk-format/src/manifest.rs#L409`:

```rust
#[derive(Clone, Debug, Hash, PartialEq, Eq, PartialOrd, Ord, Serialize, Deserialize)]
#[non_exhaustive]
pub enum ChunkPayload {
    Inline(Bytes),
    Virtual(VirtualChunkRef),
    Ref(ChunkRef),
}
```

ICECHUNK MANIFESTS

scalable, manageable, modular...

Beautiful design with powerful properties:

Scalable: At query time fetching manifests incurs a cost, but read-side cost is set by the **query**, not by the dataset size.

Efficient: Chunk-level tracking means changing one chunk in a PB array

Manageable: Storage prefixes are tracking by Virtual Chunk Containers, stored in global repo config.

Abstracted: Clear separation of physical and logical layer allows cheap manipulations of logical layer, e.g. `.shift_array(offset, axis)` or `.move(old_node_path, new_node_path)`

Modular: Nothing about this is tied to Zarr! Icechunk files don't really know they represent a Zarr store - Zarr interface is just a thin logical layer on top.

VIRTUALIZARR MANIFESTS

n-dimensional array of references

VirtualiZarr represents a manifest as a **n-dimensional array of chunk references** — one entry per chunk in the logical grid.

Each entry carries:

- `path` — URL of the source object
- `offset` — byte offset into that object
- `length` — number of bytes

(v2.6.1 now also supports inlined references.)

VirtualiZarr manifests are **in-memory**, but can be serialized to (or deserialized from) Icechunk, Kerchunk, or other hypothetical on-disk formats.

VIRTUALIZARR MANIFESTS

manipulation as n-d arrays

Manifest *is* an array (duck-typed `vz.ManifestArray` class), so can be concatenated and indexed - composes naturally with Xarray.

Array-level wrapping allows normal numpy/dask arrays (effectively containing in-memory native chunks) to coexist with arrays of virtual refs.

Zarr-data-model-specific - aimed at organizing arbitrary chunk refs (e.g. from sets of netCDF/TIFF files) into Zarr datacubes.

VIRTUALIZARR MANIFESTS

parsers

Extensible system of parsers for different filetypes each map contents of single file to zarr-like N-D array manifests.

Limited only by Zarr data model and by object storage requirement that single chunk must be a contiguous byte range.

VIRTUALIZARR MANIFESTS

direct access

Can read data directly by using `obstore` to fetch - `vz.ManifestStore` abstracts this.

```
ms = TIFFParser("some-cog.tiff")
ds = xr.open_zarr(ms).load()
```

Tracks common prefixes through the `ObjectStoreRegistry` - similar to Virtual Chunk Containers.

Transient tool - none of this affects the on-disk format, so VirtualiZarr is ultimately just a tool for assembling refs from archival files. Not actually *required* to use virtual refs with Icechunk or any other system. At some point should probably be replaced with something more integrated (in rust??).

FORMAT COMPARISON

manifest formats side-by-side

	Icechunk	VirtualiZarr	Kerchunk	DMR++	LanceDB	Iceberg
Logical data model	Zarr (but flexible)	Zarr	Zarr (but flexible)	HDF5?	tabular	tabular
Mixed chunk types	✅	✅ (per-array)	✅ (not native)	❌	✅	?
Efficient at scale	✅	✅	❌ (JSON/dict), ✅ (Parquet)	❌	✅	✅
Prefix management	✅ (VCCs)	✅ (registry)	❌	❌	✅	?
Runtime I/O layer	✅	✅ (obstore)	✅ (fsspec)	✅ (Hyrax)	✅	✅
Serverless data access	✅	✅	✅	❌	✅	✅
Change detection	✅	n/a	❌	❌	✅	✅
Transactions / versioning	✅	n/a	❌	❌	✅	✅
Serialized format	IC flatbuffers	n/a	JSON / Parquet	XML	lance	parquet

THE BIG IDEA

virtual chunks as a platform primitive

ROLLING EMBARGO

time-windowed access

Embargoed datasets can be served virtually without ever copying the embargoed bytes.

- New data lands in a **restricted-access VCC**
- After the embargo expires, either modify the manifest or the access policy - not the data

Allows e.g. selling low-latency forecasts while keeping older high-latency data free, without duplicating data.

FILTERED SUBSCRIPTIONS

per-subscriber views

A single physical archive can expose **many logical datasets** by varying which manifest a subscriber sees.

- Subscriber A sees variables `x`, `y` over region `R1`
- Subscriber B sees variables `y`, `z` over region `R2`
- Both backed by the **same underlying objects**
- Allows differential pricing, geographic restrictions, license tiers

Works by syncing manifests/snapshots only to subscriber buckets, containing virtual chunks referring back to single copy of actual data.

earthmover.io/blog/filtered-subscriptions

CAVEATS

security considerations

THE RISKS

virtual chunks are dangerous

Virtual chunks reference **arbitrary locations** containing **arbitrary data.** Serious risks:

- **Exfiltration** — bad guy just has to
  1. Create manifest with virtual chunk ref `file:///toms-pc/bitcoin-wallet`
  2. Say "hey Tom load this Icechunk repo and send me the results"
- **Overexposure** — "oops that setting meant the whole internet could see our entire private bucket..."
- **Tampering** — "What do you mean you updated those NetCDFs, my paper's results assumed it was static!"

IC default of requiring explicit opt-in to every VCC is safe, but a pain for users...

THE FIX

manage permissions via trusted server

Virtual chunks done right require a **trusted server** between the client and the storage:

- Server tracks and validates virtual chunk storage locations
- Server knows who users are, and can dispense creds for buckets only if user authorized
- Server logic adds multiple layers of protection against accidental overexposure

Blog post on Arraylake's security model for virtual chunks — coming soon

TAKEAWAYS

what did we learn about virtual chunks?

TAKEAWAYS

what did we learn about virtual chunks?

- Natural requirement in object storage
- IO-layer concern, orthogonal to Zarr spec
- Icechunk's implementation is really neat
- But core ideas not unique to Icechunk
- Powerful primitive for platform features
- Serious security considerations - only fully solvable with a trusted server

earthmover

earthmover.io