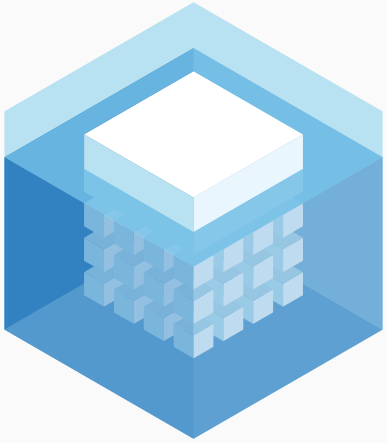
Icechunk

A database for large N-dim arrays



Sebastian Galkin

Earthmover

https://icechunk.io

What is Icechunk

https://icechunk.io

Tensor storage engine

Rust library to store and query large **binary arrays** to cloud object storage like **S3**

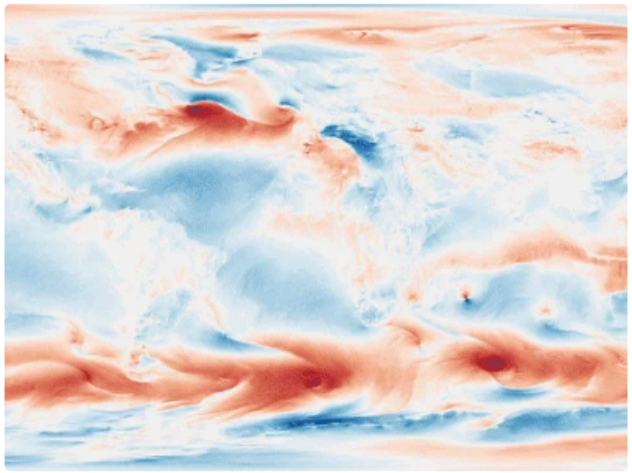
Dataset — Logical Size: 3.5 TB

DIMENSIONS (4)

latitude : 1801 longitude : 3600 time : 62 step : 145

COORDINATES (4)

Name	Dimensions	Dtype	Shape	Chunks	Size
latitude	latitude	float64	1801	1801	14.4 kB
longitude	longitude	float64	3600	3600	28.8 kB
step	step	uint64	145	145	1.16 kB
time	time	int64	62	1000	496 B



Details

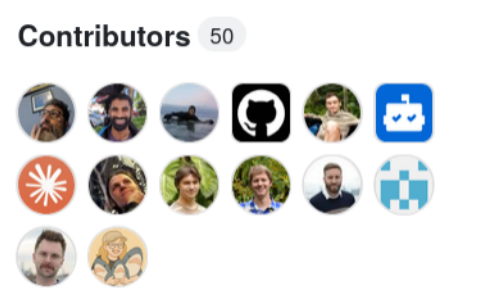
Cloud	Cloudflare R2
Region	ENAM

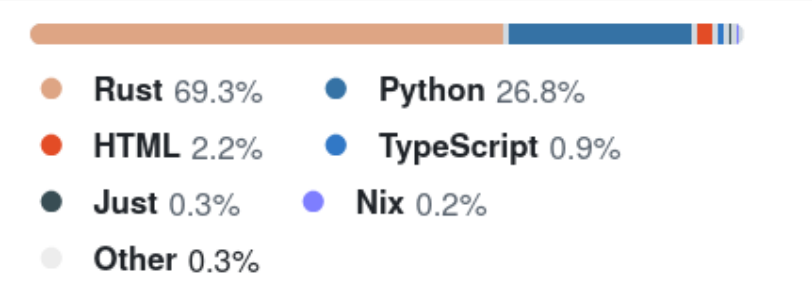
Tensor storage engine

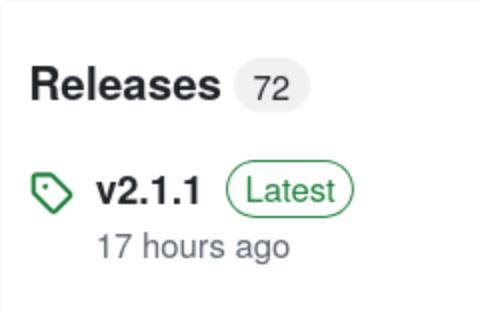
DATA VARIABLES (15)

Name	Dimensions	Dtype	Shape	Chunks	Size
100u	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI
100v	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI
10u	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI
10v	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI
2d	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI
2t	time step latitude longitude	float32	62,145,1801,3600	1,1,900,900	233 GI

Open-source

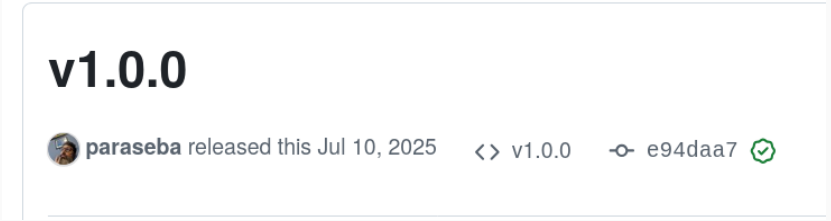






THE ASF

Apache 2.0 License



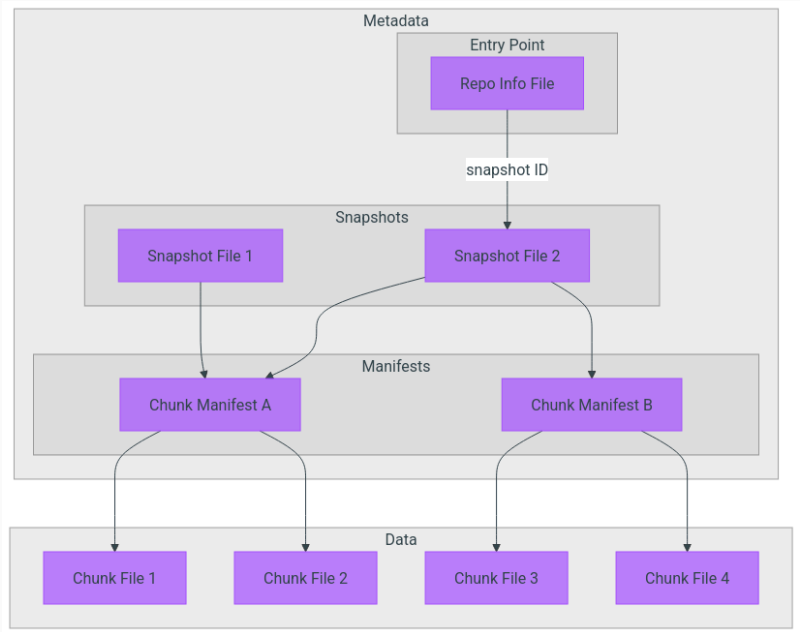
Open-source & open-format

Home / reference / spec-v2-1

Icechunk Specification, version 2.1

Note

This is the current Icechunk specification (spec version 2.1), used by Icechunk 2.1.0 and higher. For the previous on-disk formats, see spec version 2, and spec version 1. See Changes from spec version 2 to 2.1 for a summary of what changed.




```
// The repo info file — the single entry point and only mutable object
// in an Icechunk V2 repository. Stored at `$ROOT/repo`.
table Repo {

  // The spec version of this repository (1 or 2).
  spec_version: uint8;

  // All tags. MUST be sorted in ascending UTF-8 byte order of Ref.name.
  tags: [Ref] (required);

  // All branches. MUST be sorted in ascending UTF-8 byte order of Ref.name.
  branches: [Ref] (required);

  // Names of deleted tags. MUST be sorted in ascending UTF-8 byte order.
  deleted_tags: [string] (required);

  // All snapshots in this repository.
  // MUST be sorted in ascending byte order of SnapshotInfo.id.
  snapshots: [SnapshotInfo] (required);

  // Current repository availability status.
  status: RepoStatus (required);

  // Repository-level metadata (user attributes).
  metadata: [MetadataItem];

  // The ops log — a list of recent operations performed on this repository.
  latest_updates: [Update] (required);

  // Path to the previous repo info file (in the overwritten/ prefix),
  // forming a linked list for ops log entries beyond this file.
  repo_before_updates: string;

  // Repository configuration, serialized as FlexBuffers.
  config: [uint8] (flexbuffer);

  // Feature flag ids explicitly enabled by the user.
  // MUST be sorted in ascending numeric order.
  enabled_feature_flags: [uint16];

  // Feature flag ids explicitly disabled by the user.
  // MUST be sorted in ascending numeric order.
  // Flags not in either list are at their default value.
  disabled_feature_flags: [uint16];

  // Reserved for future use. Opaque bytes for future extensions.
  // Introduced in spec version 2.
  extra: [uint8];
}
```

Cloud-native

Support for all major object stores + S3-compatible stores + HTTP + local file + in-memory

```
new_s3_storage
new_gcs_storage
new_r2_storage
new_azure_blob_storage
new_tigris_storage
new_http_storage
new_in_memory_storage
new_local_filesystem_storage
...
```

Cloud-native

Read/Write only the relevant parts of the datasets

```
$ aws s3 ls --human-readable s3://bucket/example/chunks/ | head

2026-03-11 23:04:15    1.9 MiB 0005ZWR358MTDCNYNHWG
2026-03-14 05:53:43    4.5 KiB 000B0MWDWCSKQGYX8JGG
2026-03-14 05:19:42    1.9 MiB 000J5SY6P3C55ARX6B0G
2026-03-12 11:26:55    1.7 MiB 000KT0AM86ZEDSM732Q0
2026-03-10 23:07:28    1.1 MiB 000NF3NX6TB4AW58XT00
2026-03-14 17:19:15    1.9 MiB 000PJYJNBFBF2D6J7REG
2026-03-11 05:29:55    1.1 MiB 000T3MP3TTVHQBGT7SNG
2026-03-11 11:13:03    1.1 MiB 000VK770ES0QERTDZW8G
2026-03-10 23:04:14  786.4 KiB 000WP9QWG6RBFSQ0736G
2026-03-12 17:36:39    1.1 MiB 000XAS7SXP5ANQD7EVPG
```

Transactional

- Icechunk library behaves like an **embedded database**
- Readers and writers are isolated through **ACID**, serializable **transactions**
- Full history and **time-travel**
- **Branches** and immutable **tags**
- Version expiration and garbage collection

Rust

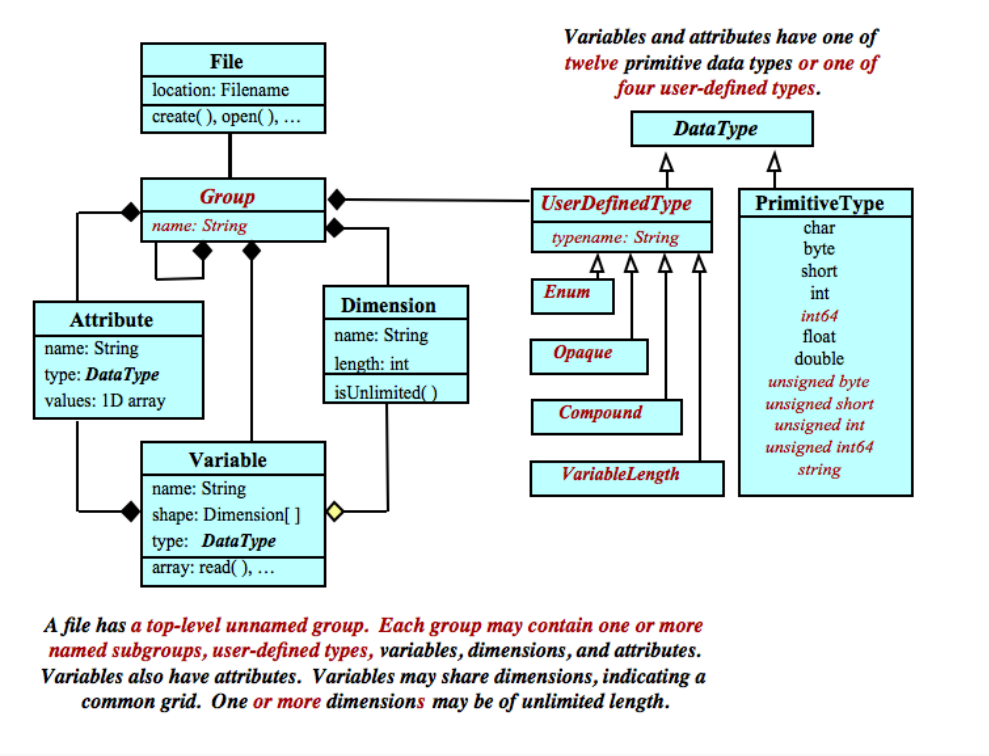
```rust
let repo = Repository::create( ... ).await?;
let mut session = repo.writable_session("main").await?;

// Write, write, write...

session.commit("My first commit", None).await?;
```

Icechunk data model

- NetCDF-like: hierarchy of folders & arrays
- Folders = Directories = Groups
- Many variables, many dimensions
- Arrays can have different data types
- Groups and arrays can have attributes (metadata)



Sample code

Example using the Rust API; the same can be achieved using the Python bindings

Rust

```rust
// 1. Tell Icechunk where to write
let storage = new_s3_storage(
    S3Options::default(),
    "my-bucket".into(), Some("weather.icechunk".into()),
    Some(S3Credentials::FromEnv), vec![], vec![], None,
)?;

// 2. Create the repository
let repo = Repository::open_or_create(
    Some(RepositoryConfig::default()), storage, Default::default(), None, true,
).await?;

// 3. Create the write session
let session = repo.writable_session("main").await?;
```

Sample code

Example using the `zarrs` crate.

Rust

```rust
// 4. Build a chunked, compressed array
let store = Arc::new(AsyncIcechunkStore::new(session));
GroupBuilder::new().build(store.clone(), "/")?.async_store_metadata().await?;
let array = ArrayBuilder::new(
    vec![720, 1440], vec![128, 128],       // shape, chunk shape
    data_type::float32(), ZARR_NAN_F32,    // dtype, fill value
).dimension_names(["lat", "lon"].into())
 .build_arc(store.clone(), "/temperature")?;
array.async_store_metadata().await?;

// 5. Write a chunk straight from a vector (transparently compressed)
array.async_store_chunk(&[0, 0], vec![300.0f32; 128 * 128]).await?;
// 6. Commit: one atomic, isolated snapshot
let snap = store.session().write().await.commit("Add temperature").execute().await?;
// 7. Tag it
repo.create_tag("v1", &snap).await?;
```

Sample code

Rust

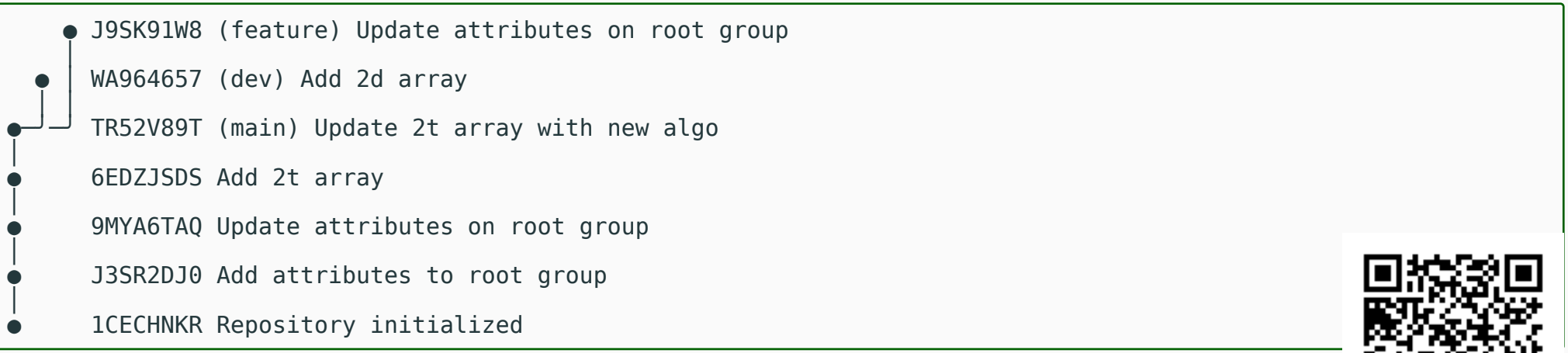
```rust
// 8. Open the repository on a tag
let past = repo.readonly_session(&VersionInfo::TagRef("v1".into())).await?;
let store = Arc::new(AsyncIcechunkStore::new(past));

// 9. Compute with the array
let array = Array::async_open(store, "/temperature").await?;
let data: ArrayD<f32> = array.async_retrieve_array_subset(&array.subset_all()).await?;
println!("mean = {:?}", data.mean());
```

Why you should try Icechunk

Consistency and error recovery

- Transactions/concurrency/isolation
- History
- Branches and tags



Why you should try Icechunk

Production ready

Rust and Python unit, integration and stateful tests

```
Summary [  23.174s] 393 tests run: 393 passed, 27 skipped
```

```
====== 709 passed, 26 skipped, 3 xfailed in 117.72s (0:01:57) ======
```

Some blog posts

- Property and stateful testing in Icechunk
- Concurrency & fault injection testing
- Cross-version stateful compatibility testing

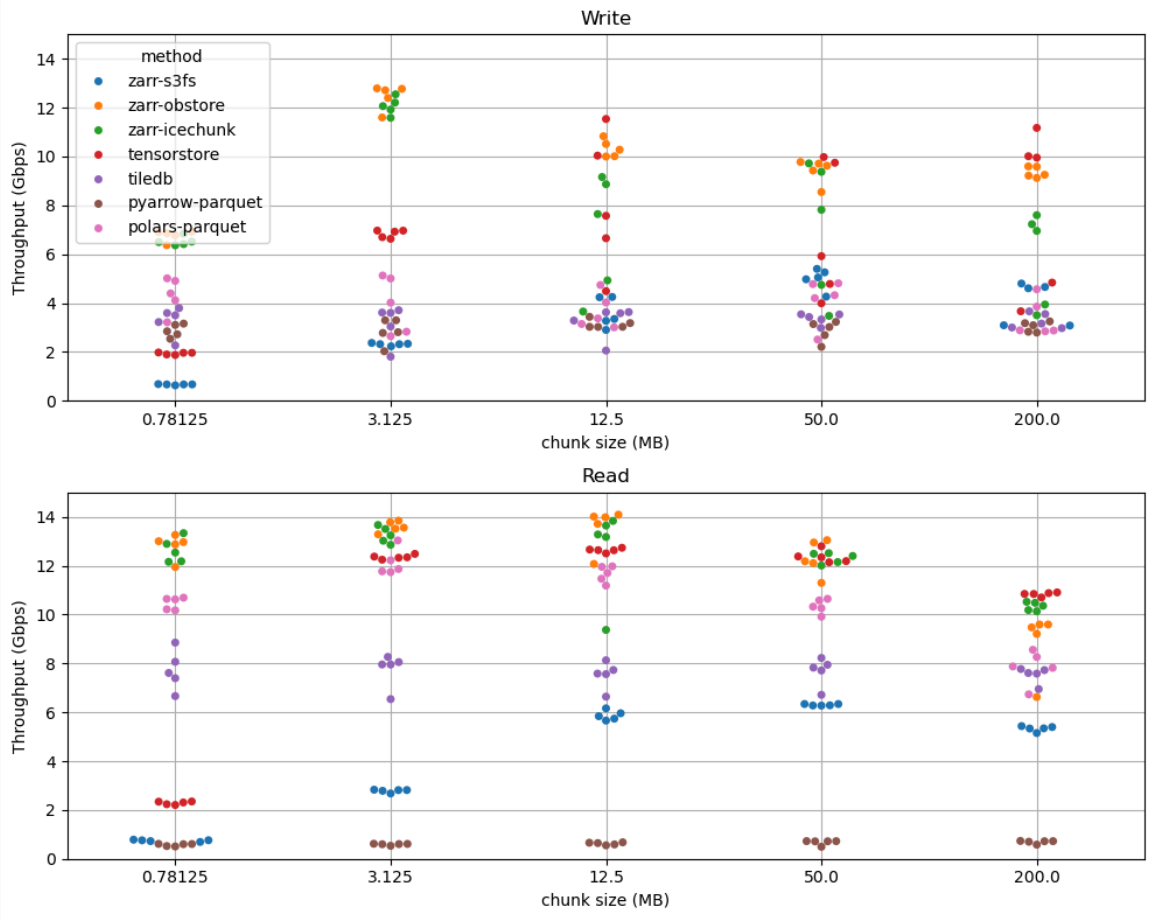
Why you should try Icechunk

Space Efficiency

- Powerful compression
- Write only what changed
  - Keep full or partial history
  - No data duplication
- Virtual datasets
  - Don’t rewrite data existing in other formats
  - Make Icechunk “point” to existing files
  - NetCDF, HDF5, TIFF, GRIB, and more

Why you should try Icechunk

It's FAST: Consistently saturates the network cards



Why you should try Icechunk

Adoption: Many PB of data in the public and private sectors

- NOAA - National Weather Service - CIRRUS
- NASA - Virtualized IMERG and others
- Advanced Research + Invention Agency (ARIA) - Forecasting Tipping Points programme
- MeteoSwiss - Operational Weather Forecasting
- Earthmover Marketplace
- Many companies across earth observation, climate, weather, and finance
- Rapidly growing usage in microscopy, transcriptomics, and genomics
- Probably a lot more we don't know about

Summary

- If you have **array data** you want to store in the **cloud**
- Icechunk is:
  - safe and consistent
  - powerful
  - fast
  - open source & open format
- Go to https://icechunk.io and give it a try
- Join our Slack community or submit an issue if you need help