Custom Indexer

Refer to Access Sui Data for an overview of options to access Sui network data.

You can build custom indexers using the Sui micro-data ingestion framework. To create an indexer, you subscribe to a checkpoint stream with full checkpoint content. This stream can be one of the publicly available streams from Mysten Labs, one that you set up in your local environment, or a combination of the two.

Establishing a custom indexer helps improve latency, allows pruning the data of your Sui Full node, and provides efficient assemblage of checkpoint data.

To use the framework, implement a basic interface:

In this example, the CheckpointData struct represents full checkpoint content. The struct contains checkpoint summary and contents, as well as detailed information about each individual transaction. The individual transaction data includes events and input/output objects. The full definition for this content is in the full_checkpoint_content.rs file of the sui-types crate.

See the Source code for an implementation section for a complete code example.

Data ingestion for your indexer supports several checkpoint stream sources.

The most straightforward stream source is to subscribe to a remote store of checkpoint contents. Mysten Labs provides the following buckets:

The checkpoint files are stored in the following format: https://checkpoints.testnet.sui.io/.chk . You can download the checkpoint file by sending an HTTP GET request to the relevant URL. Try it yourself for checkpoint 1 at https://checkpoints.testnet.sui.io/1.chk .

The Sui data ingestion framework provides a helper function to quickly bootstrap an indexer workflow.

This is suitable for setups with a single ingestion pipeline where progress tracking is managed outside of the framework.

Colocate the data ingestion daemon with a Full node and enable checkpoint dumping on the latter to set up a local stream source. After enabling, the Full node starts dumping executed checkpoints as files to a local directory, and the data ingestion daemon subscribes to changes in the directory through an inotify-like mechanism. This approach allows minimizing ingestion latency (checkpoint are processed immediately after a checkpoint executor on a Full node) and getting rid of dependency on an externally managed bucket.

To enable, add the following to your Full node configuration file:

Let's highlight a couple lines of code:

The data ingestion executor can run multiple workflows simultaneously. For each workflow, you need to create a separate worker pool and register it in the executor. The WorkerPool requires an instance of the Worker trait, the name of the workflow (which is used for tracking the progress of the flow in the progress store and metrics), and concurrency.

The concurrency parameter specifies how many threads the workflow uses. Having a concurrency value greater than 1 is helpful when tasks are idempotent and can be processed in parallel and out of order. The executor only updates the progress/watermark to a certain checkpoint when all preceding checkpoints are processed.

Specify both a local and remote store as a fallback to ensure constant data flow. The framework always prioritizes locally available checkpoint data over remote data. It's useful when you want to start utilizing your own Full node for data ingestion but need to partially backfill historical data or just have a failover.

Code for the cargo.toml manifest file for the custom indexer.

Find the following source code in the Sui repo .

Interface and data format

To use the framework, implement a basic interface:

In this example, the CheckpointData struct represents full checkpoint content. The struct contains checkpoint summary and contents, as well as detailed information about each individual transaction. The individual transaction data includes events and input/output

objects. The full definition for this content is in the full_checkpoint_content.rs file of the sui-types crate.

See the Source code for an implementation section for a complete code example.

Data ingestion for your indexer supports several checkpoint stream sources.

The most straightforward stream source is to subscribe to a remote store of checkpoint contents. Mysten Labs provides the following buckets:

The checkpoint files are stored in the following format: https://checkpoints.testnet.sui.io/.chk . You can download the checkpoint file by sending an HTTP GET request to the relevant URL. Try it yourself for checkpoint 1 at https://checkpoints.testnet.sui.io/1.chk .

The Sui data ingestion framework provides a helper function to quickly bootstrap an indexer workflow.

This is suitable for setups with a single ingestion pipeline where progress tracking is managed outside of the framework.

Colocate the data ingestion daemon with a Full node and enable checkpoint dumping on the latter to set up a local stream source. After enabling, the Full node starts dumping executed checkpoints as files to a local directory, and the data ingestion daemon subscribes to changes in the directory through an inotify-like mechanism. This approach allows minimizing ingestion latency (checkpoint are processed immediately after a checkpoint executor on a Full node) and getting rid of dependency on an externally managed bucket.

To enable, add the following to your Full node configuration file:

Let's highlight a couple lines of code:

The data ingestion executor can run multiple workflows simultaneously. For each workflow, you need to create a separate worker pool and register it in the executor. The WorkerPool requires an instance of the Worker trait, the name of the workflow (which is used for tracking the progress of the flow in the progress store and metrics), and concurrency.

The concurrency parameter specifies how many threads the workflow uses. Having a concurrency value greater than 1 is helpful when tasks are idempotent and can be processed in parallel and out of order. The executor only updates the progress/watermark to a certain checkpoint when all preceding checkpoints are processed.

Specify both a local and remote store as a fallback to ensure constant data flow. The framework always prioritizes locally available checkpoint data over remote data. It's useful when you want to start utilizing your own Full node for data ingestion but need to partially backfill historical data or just have a failover.

Code for the cargo.toml manifest file for the custom indexer.

Find the following source code in the Sui repo .

Checkpoint stream sources

Data ingestion for your indexer supports several checkpoint stream sources.

The most straightforward stream source is to subscribe to a remote store of checkpoint contents. Mysten Labs provides the following buckets:

The checkpoint files are stored in the following format: https://checkpoints.testnet.sui.io/.chk . You can download the checkpoint file by sending an HTTP GET request to the relevant URL. Try it yourself for checkpoint 1 at https://checkpoints.testnet.sui.io/1.chk .

The Sui data ingestion framework provides a helper function to quickly bootstrap an indexer workflow.

This is suitable for setups with a single ingestion pipeline where progress tracking is managed outside of the framework.

Colocate the data ingestion daemon with a Full node and enable checkpoint dumping on the latter to set up a local stream source. After enabling, the Full node starts dumping executed checkpoints as files to a local directory, and the data ingestion daemon subscribes to changes in the directory through an inotify-like mechanism. This approach allows minimizing ingestion latency (checkpoint are processed immediately after a checkpoint executor on a Full node) and getting rid of dependency on an externally managed bucket.

To enable, add the following to your Full node configuration file:

Let's highlight a couple lines of code:

The data ingestion executor can run multiple workflows simultaneously. For each workflow, you need to create a separate worker pool and register it in the executor. The WorkerPool requires an instance of the Worker trait, the name of the workflow (which is used for tracking the progress of the flow in the progress store and metrics), and concurrency.

The concurrency parameter specifies how many threads the workflow uses. Having a concurrency value greater than 1 is helpful when tasks are idempotent and can be processed in parallel and out of order. The executor only updates the progress/watermark to a certain checkpoint when all preceding checkpoints are processed.

Specify both a local and remote store as a fallback to ensure constant data flow. The framework always prioritizes locally available checkpoint data over remote data. It's useful when you want to start utilizing your own Full node for data ingestion but need to partially backfill historical data or just have a failover.

Code for the cargo.toml manifest file for the custom indexer.

Find the following source code in the Sui repo .

Source code for an implementation

Find the following source code in the Sui repo .

Related links