How to deploy a mirror Data Availability Server (DAS)

PUBLIC PREVIEW DOCUMENT This document is currently inpublic preview and may change significantly as feedback is captured from readers like you. Click theRequest an update button at the top of this document orjoin the Arbitrum Discord to share your feedback. Running a regular DAS vs running a mirror DAS The main use-case for running a mirror DAS is to complement your setup as a Data Availability Committee (DAC) member. That means that you should run your main DAS first, and then configure the mirror DAS. Refer toHow to deploy a DAS if needed. AnyTrust chains rely on an external Data Availability Committee (DAC) to store data and provide it on-demand instead of using itsparent chain as the Data Availability (DA) layer. The members of the DAC run a Data Availability Server (DAS) to handle these operations.

In this how-to, you'll learn how to configure a mirror DAS that servesGET requests for stored batches of information through a REST HTTP interface. For a refresher on DACs, refer to theIntroduction .

This how-to assumes that you're familiar with:

- How a regular DAS works and what configuration options are available. Refer toHow to deploy a DAS
- for a refresher.
- Kubernetes
- . The examples in this guide use Kubernetes to containerize your DAS.

What is a mirror DAS?

To avoid exposing the REST interface of your main DAS to the public in order to prevent spamming attacks (as explained inHow to deploy a DAS), you can choose to run a mirror DAS to complement your setup. The mirror DAS will handle all public REST requests, while reading information from the main DAS via its (now private) REST interface.

In general, mirror DA servers serve two main purposes:

1. Prevent the main DAS from having to serve requests for data, allowing it to focus only on storing the data received.
2. Provide resiliency to the network in the case of a DAS going down.

Configuration options

A mirror DAS will use the same tool and, thus, the same configuration options as your main DAS. You can find an explanation of those options inHow to deploy a DAS .

How to deploy a mirror DAS

Step 0: Prerequisites

Gather the following information:

- The latest Nitro docker image:offchainlabs/nitro-node:v2.3.2-064fa11
- An RPC endpoint for theparent chain
- . It is recommended to use athird-party provider RPC
- orrun your own node
- to prevent being rate limited.
- The SequencerInbox contract address in the parent chain.
- URL of the list of REST endpoints of other DA servers to configure the REST aggregator.

Step 1: Set up a persistent volume

First, we'll set up a volume to store the DAS database. In k8s, we can use a configuration like this:

apiVersion : v1 kind : PersistentVolumeClaim metadata : name : das - mirror spec : accessModes : - ReadWriteOnce resources : requests : storage : 200Gi storageClassName : gp2

Step 2: Deploy the mirror DAS

To run the mirror DAS, we'll use thedaserver tool and we'll configure the following parameters:

Parameter Description --data-availability.parent-chain-node-url RPC endpoint of a parent chain node --data-availability.sequencer-inbox-address Address of the SequencerInbox in the parent chain --enable-rest Enables the REST server listening on --rest-addr and --rest-port --rest-addr REST server listening interface (default "localhost") --rest-port (Optional) REST server listening port (default 9877) --log-level Log level: 1 - ERROR, 2 - WARN, 3 - INFO, 4 - DEBUG, 5 - TRACE (default 3) --data-availability.rest-aggregator.enable Enables retrieval of sequencer batch data from a list of remote REST endpoints --data-availability.rest-aggregator.online-url-list A URL to a list of URLs of REST DAS endpoints that is

checked at startup. This option is additive with the urls option --data-availability.rest-aggregator.urls List of URLs including 'http://' or 'https://' prefixes and port numbers to REST DAS endpoints. This option is additive with the online-url-list option --data-availability.rest-aggregator.sync-to-storage.check-already-exists When using a REST aggregator, checks if the data already exists in this DAS's storage. Must be disabled for fast sync with an IPFS backend (default true) --data-availability.rest-aggregator.sync-to-storage.eager When using a REST aggregator, eagerly syncs batch data to this DAS's storage from the REST endpoints, using the parent chain as the index of batch data hashes; otherwise only syncs lazily --data-availability.rest-aggregator.sync-to-storage.eager-lower-bound-block When using a REST aggregator that's eagerly syncing, starts indexing forward from this block from the parent chain. Only used if there is no sync state. --data-availability.rest-aggregator.sync-to-storage.retention-period When using a REST aggregator, period to retain the synced data (defaults to forever) --data-availability.rest-aggregator.sync-to-storage.state-dir When using a REST aggregator, directory to store the sync state in, i.e. the block number currently synced up to, so that it doesn't sync from scratch each time To enable caching, you can use the following parameters:

Parameter Description --data-availability.local-cache.enable Enables local in-memory caching of sequencer batch data --data-availability.local-cache.capacity Maximum number of entries (up to 64KB each) to store in the cache. (default 20000) Finally, for the storage backends you wish to configure, use the following parameters. Toggle between the different options to see all available parameters.

- AWS S3 bucket
- Local Badger database
- Local files
- IPFS

Parameter Description --data-availability.s3-storage.enable Enables storage/retrieval of sequencer batch data from an AWS S3 bucket --data-availability.s3-storage.access-key S3 access key --data-availability.s3-storage.bucket S3 bucket --data-availability.s3-storage.region S3 region --data-availability.s3-storage.secret-key S3 secret key --data-availability.s3-storage.object-prefix Prefix to add to S3 objects --data-availability.s3-storage.discard-after-timeout Whether to discard data after its expiry timeout (setting it to false, activates the “archive” mode) Parameter Description --data-availability.local-db-storage.enable Enables storage/retrieval of sequencer batch data from a database on the local filesystem --data-availability.local-db-storage.data-dir Absolute path of the directory inside the volume in which to store the database (it must exist) --data-availability.local-db-storage.discard-after-timeout Whether to discard data after its expiry timeout (setting it to false, activates the “archive” mode) Parameter Description --data-availability.local-file-storage.enable Enables storage/retrieval of sequencer batch data from a directory of files, one per batch --data-availability.local-file-storage.data-dir Absolute path of the directory inside the volume in which to store the data (it must exist) Parameter Description --data-availability.ipfs-storage.enable Enables storage/retrieval of sequencer batch data from IPFS --data-availability.ipfs-storage.profiles Comma separated list of IPFS profiles to use --data-availability.ipfs-storage.read-timeout Timeout for IPFS reads, since by default it will wait forever. Treat timeout as not found (default 1m0s) Here's an exampledaserver command for a mirror DAS that:

- Enables local cache
- Enables AWS S3 bucket storage that doesn't discard data after expiring (archive
-)
- Enables local Badger database storage that doesn't discard data after expiring (archive
-)
- Uses a local main DAS as part of the REST aggregator

daserver --data-availability.parent-chain-node-url "" --data-availability.sequencer-inbox-address "

" --enable-rest --rest-addr '0.0.0.0' --log-level 3 --data-availability.local-cache.enable --data-availability.rest-aggregator.enable --data-availability.rest-aggregator.urls "http://your-main-das.svc.cluster.local:9877" --data-availability.rest-aggregator.online-url-list "" --data-availability.rest-aggregator.sync-to-storage.eager --data-availability.rest-aggregator.sync-to-storage.eager-lower-bound-block "BLOCK NUMBER" --data-availability.rest-aggregator.sync-to-storage.state-dir /home/user/data/syncState --data-availability.s3-storage.enable --data-availability.s3-storage.access-key "" --data-availability.s3-storage.bucket "" --data-availability.s3-storage.region "" --data-availability.s3-storage.secret-key "" --data-availability.s3-storage.object-prefix "/" --data-availability.s3-storage.discard-after-timeout false --data-availability.local-db-storage.enable --data-availability.local-db-storage.data-dir /home/user/data/badgerdb --data-availability.local-db-storage.discard-after-timeout false And here's an example of how to use a k8s deployment to run that command:

apiVersion : apps/v1 kind : Deployment metadata : name : das - mirror spec : replicas :

1 selector : matchLabels : app : das - mirror strategy : rollingUpdate : maxSurge :

0 maxUnavailable : 50% type : RollingUpdate template : metadata : labels : app : das - mirror spec : containers : -

command : - bash -

- c -

| mkdir -p /home/user/data/badgerdb mkdir -p /home/user/data/syncState /usr/local/bin/daserver --data-availability.parent-chain-node-url "" --data-availability.sequencer-inbox-address "

" --enable-rest --rest-addr '0.0.0.0' --log-level 3 --data-availability.local-cache.enable --data-availability.rest-aggregator.enable --data-availability.rest-aggregator.urls "http://your-main-das.svc.cluster.local:9877" --data-availability.rest-aggregator.online-url-list "" --data-availability.rest-aggregator.sync-to-storage.eager --data-availability.rest-aggregator.sync-to-storage.eager-lower-bound-block "BLOCK NUMBER" --data-availability.rest-aggregator.sync-to-storage.state-dir /home/user/data/syncState --data-availability.s3-storage.enable --data-availability.s3-storage.access-key "" --data-availability.s3-storage.bucket "" --data-availability.s3-storage.region "" --data-availability.s3-storage.secret-key "" --data-availability.s3-storage.object-prefix "/" --data-availability.local-db-storage.enable --data-availability.local-db-storage.data-dir /home/user/data/badgerdb image : offchainlabs/nitro - node : v2.3.2 - 064fa11 imagePullPolicy : Always resources : limits : cpu :

"4" memory : 10Gi requests : cpu :

"4" memory : 10Gi ports : -

containerPort :

9877 hostPort :

9877 protocol : TCP volumeMounts : -

mountPath : /home/user/data/ name : data readinessProbe : failureThreshold :

3 httpGet : path : /health/ port :

9877 scheme : HTTP initialDelaySeconds :

5 periodSeconds :

5 successThreshold :

1 timeoutSeconds :

1 volumes : -

name : data persistentVolumeClaim : claimName : das - mirror

Archive DA servers

Archive DA servers are servers that don't discard any data after expiring. Each DAC should have at the very least one archive DAS to ensure all historical data is available.

To activate the "archive mode" in your DAS, set the parameterdiscard-after-timeout tofalse in your storage method. For example:

--data-availability.s3-storage.discard-after-timeout

false --data-availability.local-db-storage.discard-after-timeout = false Note thatlocal-file-storage andipfs-storage don't discard data after expiring, so the optiondiscard-after-timeout is not available.

Archive servers should make use of the--data-availability.rest-aggregator.sync-to-storage options described above to pull in any data that they don't have.

Helm charts

A helm chart is available atArtifactHUB . It supports running a mirror DAS by providing the parameters for your server. Find more information in theOCL community Helm charts repository .

Testing the DAS

Once the DAS is running, we can test if everything is working correctly using the following methods.

Test 1: REST health check

The REST interface enabled in the mirror DAS has a health check on the path/health which will return200 if the underlying storage is working, otherwise503 .

Example:

curl -I < YOUR REST ENDPOINT

/health

Security considerations

Keep in mind the following information when running the mirror DAS.

For a mirror DAS, using a load balancer is recommended to manage incoming traffic effectively. Additionally, as the REST interface is cacheable, consider deploying a Content Delivery Network (CDN) or caching proxy in front of your REST endpoint. The URL for the REST interface will be publicly known; ensure that it is sufficiently distinct from the RPC endpoint to prevent the latter from being easily discovered.

What to do next?

Once the DAS is deployed and tested, you'll have to communicate the following information to the chain owner, so they can update the chain parameters and configure the sequencer:

- *The https URL for the REST endpoint (e.g. das.your-chain.io/rest)*

Optional parameters

Besides the parameters described in this guide, there are some more options that can be useful when running the DAS. For a comprehensive list of configuration parameters, you can rundaserver --help .

Parameter Description --conf.dump Prints out the current configuration --conf.file Absolute path to the configuration file inside the volume to use instead of specifying all parameters in the command

Metrics

The DAS comes with the option of producing Prometheus metrics. This option can be activated by using the following parameters:

Parameter Description --metrics Enables the metrics server --metrics-server.addr Metrics server address (default "127.0.0.1") --metrics-server.port Metrics server port (default 6070) --metrics-server.update-interval Metrics server update interval (default 3s) When metrics are enabled, several useful metrics are available at the configured port, at pathdebug/metrics ordebug/metrics/prometheus .

RPC metrics

Metric Description arb_das_rpc_store_requests Count of RPC Store calls arb_das_rpc_store_success Successful RPC Store calls arb_das_rpc_store_failure Failed RPC Store calls arb_das_rpc_store_bytes Bytes retrieved with RPC Store calls arb_das_rpc_store_duration (p50, p75, p95, p99, p999, p9999) Duration of RPC Store calls (ns)

REST metrics

Metric Description arb_das_rest_getbyhash_requests Count of REST GetByHash calls arb_das_rest_getbyhash_success Successful REST GetByHash calls arb_das_rest_getbyhash_failure Failed REST GetByHash calls arb_das_rest_getbyhash_bytes Bytes retrieved with REST GetByHash calls arb_das_rest_getbyhash_duration (p50, p75, p95, p99, p999, p9999) Duration of REST GetByHash calls (ns) Edit this page Last updatedonMar 14, 2024 Previous How to deploy a Data Availability Server (DAS) Next How to configure the Data Availability Committee (DAC) in your chain