Chain monitoring options

This explainer covers the basics of onchain and offchain monitoring options for your OP Stack chain. Onchain monitoring services allow chain operators to monitor the overall system and onchain events. Offchain monitoring lets chain operators to monitor the operation and behavior of nodes and other offchain components.

Onchain monitoring services

Onchain monitoring services provide insights into the overall system, helping chain operators track and monitor on-chain events. Some examples of onchain monitoring services includemonitorism anddispute-mon .

monitorism

Monitorism is a tooling suite that supports monitoring and active remediation actions for the OP Stack chain. Monitorism uses monitors as passive security providing automated monitoring for the OP Stack. They are used to monitor the OP stack and alert on specific events that could be a sign of a security incident.

Currently, the list of monitors includes:

Security integrity monitors: These are monitors necessary for making sure Bridges between L2 and L1 are safe and work as expected. These monitors are divided in two subgroups:

- Pre-Faultproof Chain Monitors:* Fault Monitor: checks for changes in output roots posted to the L2OutputOracle contract. When a change is detected, it reconstructs the output root from a trusted L2 source and looks for a match.
-
    - Withdrawals Monitor: checks for new withdrawals that have been proven to the OptimismPortal contract. Each withdrawal is checked against theL2ToL1MessagePasser
-
    - contract.
- Faultproof chain monitors:* Faultproof Withdrawal: The Faultproof Withdrawal component monitorsProvenWithdrawals
-
    - events on theOptimismPortal
-
    - contract and performs checks to detect any violations of invariant conditions on the chain. If a violation is detected, the issue is logged, and a Prometheus metric is set for the event. This component is designed to work exclusively with chains that are already utilizing the Fault Proofs system. This is a new version of the deprecatedchain-mon
-
    - ,faultproof-wd-mon
-
    - . For detailed information on how the component works and the algorithms used, please refer to the component README.

Security monitors: Those tools monitor other aspects of several contracts used in optimism:

- Global Events Monitor: made for taking YAML rules as configuration and monitoring the events that are emitted on the chain.
- Liveness Expiration Monitor: monitors the liveness expiration on Safes.
- Balances Monitor: emits a metric reporting the balances for the configured accounts.
- Multisig Monitor: The multisig monitor reports the paused status of the OptimismPortal contract. If set, reports the latest nonce of the configured Safe address and the latest presigned nonce stored in One Password.. The latest presigned nonce is identified by looking for items in the configured vault that follow aready-.json
- name. The highest nonce of this item name format is reported.
- Drippie Monitor: tracks the execution and executability of drips within a Drippie contract.
- Secrets Monitor: takes a Drippie contract as a parameter and monitors for any drips within that contract that use theCheckSecrets
- dripcheck contract.CheckSecrets
- is a dripcheck that allows a drip to begin once a specific secret has been revealed (after a delay period) and cancels the drip if a second secret is revealed. Monitoring these secrets is important, as their revelation may indicate that the secret storage platform has been compromised and someone is attempting to exfiltrate the ETH controlled by the drip.

For more information on these monitors and how to use themcheck out the repo(opens in a new tab).

dispute-mon

Chain operators should consider runningop-dispute-mon . It's an essential security monitoring service that tracks game statuses, providing visibility over the last 28 days.

dispute-mon is set up and built the same way asop-challenger . This means that you can run it the same way (runmake op-dispute-mon in the directory).

A basic configuration option would look like this:

OP_DISPUTE_MON_LOG_FORMAT=logfmt OP_DISPUTE_MON_METRICS_ENABLED=true
OP_DISPUTE_MON_METRICS_ADDR=0.0.0.0 OP_DISPUTE_MON_METRICS_PORT=7300

OP_DISPUTE_MON_L1_ETH_RPC=.. OP_DISPUTE_MON_ROLLUP_RPC=..
OP_DISPUTE_MON_GAME_FACTORY_ADDRESS=..

OP_DISPUTE_MON_HONEST_ACTORS=.. OP_DISPUTE_MON_HONEST_ACTORS is a CSV (no spaces) list of addresses that are used for the honestop-challenger instances.

Additional flags:

- OP_DISPUTE_MON_GAME_WINDOW
- : This is the window of time to report on games. It should leave a buffer beyond the max game duration for bond claiming. If Fault Proof game parameters are not changes (e.g. MAX_CLOCK_DURATION), it is recommended to leave this as the default.
- OP_DISPUTE_MON_MONITOR_INTERVAL
- : The interval at which to check for new games. Defaults to 30 seconds currently.
- OP_DISPUTE_MON_MAX_CONCURRENCY
- : The max thread count. Defaults to 5 currently.

You can find more info onop-dispute-mon onthe repo(opens in a new tab) .

Chain operators can easily create their grafana dashboard for Dispute Monitor using the following json file.Download the Dispute Monitor JSON .

Offchain component monitoring

Offchain monitoring allows chain operators to monitor the operation and behavior of nodes and other offchain components. Some of the more common components that you'll likely want to monitor includeop-node ,op-geth ,op-proposer ,op-batcher , andop-challenger . The general steps for enabling offchain monitoring are pretty consistent for all the OP components:

1. Expose the monitoring port by enabling the--metrics.enabled
2. flag
3. Customize the metrics port and address via the--metrics.port
4. and--metrics.addr
5. flags, respectively
6. UsePrometheus(opens in a new tab)
7. to scrape data from the metrics port
8. Save the data ininfluxdb
9. Share the data withGrafana(opens in a new tab)
10. to build your custom dashboard

op-node

op-node metrics and monitoring is detailed in theNode Metrics and Monitoring guide. To enable metrics, pass the--metrics.enabled flag toop-node and follow the steps above for customization options. Seethis curated list for important metrics to track specifically forop-node .

op-geth

To enable metrics, pass the--metrics.enabled flag to the op-geth. You can customize the metrics port and address via the--metrics.port and--metrics.addr flags, respectively.

op-proposer

To enable metrics, pass the--metrics.enabled flag to the op-proposer. You can customize the metrics port and address via the--metrics.port and--metrics.addr flags, respectively.

You can find more information about these flags in ourProposer configuration doc(opens in a new tab) .

op-batcher

To enable metrics, pass the--metrics.enabled flag to the op-batcher. You can customize the metrics port and address via the--metrics.port and--metrics.addr flags, respectively.

You can find more information about these flags in ouBatcher configuration doc(opens in a new tab) .

op-challenger

Theop-challenger operates as thehonest actor in the fault dispute system and defends the chain by securing theOptimismPortal and ensuring the game always resolves to the correct state of the chain. For verifying the legitimacy of claims,op-challenger relies on a synced, trusted rollup node as well as a trace provider (e.g.,Cannon). See theOP-Challenger Explainer for more information on this service.

To enable metrics, pass the--metrics.enabled flag toop-challenger and follow the steps above for customization options.

--metrics.addr value (default: "0.0.0.0") (OP_CHALLENGER_METRICS_ADDR) Metrics listening address

--metrics.enabled (default: false) (OP_CHALLENGER_METRICS_ENABLED) Enable the metrics server

--metrics.port value (default: 7300) (OP_CHALLENGER_METRICS_PORT) Metrics listening port

Next steps

- If you encounter difficulties at any stage of this process, please reach out todeveloper support(opens in a new tab)
- .

Chain monitoring op-challenger