Validator Tasks

This guide focuses on running the Sui node software as a validator.

To run a Sui Validator a machine with the following is required:

You can deploy Sui node in a number of ways.

There are pre-built container images available in Docker Hub .

And pre built linux/amd64 binaries available in S3 that you can fetch using one of the following methods:

To build directly from source:

Configuration and guides are available for the following deployment options:

Sui node runs with a single configuration file provided as an argument, example:

./sui-node --config-path /opt/sui/config/validator.yaml .

See Validator for configuration templates.

Sui node uses the following ports by default:

To run a validator successfully, it is critical that ports 8080-8084 are open as outlined, including the specific protocol (TCP/UDP).

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Requirements

To run a Sui Validator a machine with the following is required:

You can deploy Sui node in a number of ways.

There are pre-built container images available in Docker Hub .

And pre built linux/amd64 binaries available in S3 that you can fetch using one of the following methods:

To build directly from source:

Configuration and guides are available for the following deployment options:

Sui node runs with a single configuration file provided as an argument, example:

./sui-node --config-path /opt/sui/config/validator.yaml .

See Validator for configuration templates.

Sui node uses the following ports by default:

To run a validator successfully, it is critical that ports 8080-8084 are open as outlined, including the specific protocol (TCP/UDP).

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Deployment

You can deploy Sui node in a number of ways.

There are pre-built container images available in Docker Hub .

And pre built linux/amd64 binaries available in S3 that you can fetch using one of the following methods:

To build directly from source:

Configuration and guides are available for the following deployment options:

Sui node runs with a single configuration file provided as an argument, example:

./sui-node --config-path /opt/sui/config/validator.yaml .

See Validator for configuration templates.

Sui node uses the following ports by default:

To run a validator successfully, it is critical that ports 8080-8084 are open as outlined, including the specific protocol (TCP/UDP).

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Configuration

Sui node runs with a single configuration file provided as an argument, example:

./sui-node --config-path /opt/sui/config/validator.yaml .

See Validator for configuration templates.

Sui node uses the following ports by default:

To run a validator successfully, it is critical that ports 8080-8084 are open as outlined, including the specific protocol (TCP/UDP).

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Connectivity

Sui node uses the following ports by default:

To run a validator successfully, it is critical that ports 8080-8084 are open as outlined, including the specific protocol (TCP/UDP).

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Network Buffer

From load testing Sui validator networks, it has been determined that the default Linux network buffer sizes are too small. We recommend increasing them using one of the following two methods:

These settings can be added to a new sysctl file specifically for the sui-node, or appended to an existing file. Modifications made in this way will persist across system restarts.

Create a new sysctl file for the sui-node:

Add these lines to the file, overwriting existing settings if necessary.

Apply the settings immediately, before the next restart

These modifications do not persist across system restarts. Therefore, the commands should be run each time the host restarts.

To verify that the system settings have indeed been updated, check the output of the following command:

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Storage

All Sui node related data is stored by default under /opt/sui/db/ . This is controlled in the Sui node configuration file.

Ensure that you have an appropriately sized disk mounted for the database to write to.

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Key management

The following keys are used by Sui node:

These are configured in the Sui node configuration file .

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Monitoring

Sui node exposes metrics via a local HTTP interface. These can be scraped for use in a central monitoring system as well as viewed directly from the node.

Sui node also pushes metrics to a central Sui metrics proxy.

Logs are controlled using the RUST_LOG environment variable.

The RUST_LOG_JSON=1 environment variable can optionally be set to enable logging in JSON structured format.

Depending on your deployment method, these are configured in the following places:

To view and follow the Sui node logs:

To search for a particular match

It is possible to change the logging configuration while a node is running using the admin interface.

To view the currently configured logging values:

To change the currently configured logging values:

Public dashboard for network wide visibility:

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Software updates

When an update is required to the Sui node software the following process can be used. Follow the relevant Systemd or Docker Compose runbook depending on your deployment type. It is highly unlikely that you will want to restart with a clean database.

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

State sync

Checkpoints in Sui contain the permanent history of the network. They are comparable to blocks in other blockchains with one big difference being that they are lagging instead of leading. All transactions are final and executed prior to being included in a checkpoint.

These checkpoints are synchronized between validators and fullnodes via a dedicated peer to peer state sync interface.

Inter-validator state sync is always permitted however there are controls available to limit what fullnodes are allowed to sync from a specific validator.

The default and recommended max-concurrent-connections: 0 configuration does not affect inter-validator state sync, but will restrict all fullnodes from syncing. The Sui node configuration can be modified to allow a known fullnode to sync from a validator:

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Chain operations

The following chain operations are executed using the sui CLI. This binary is built and provided as a release similar to sui-node , examples:

It is recommended and often required that the sui binary release/version matches that of the deployed network.

You can leverage Validator Tool to perform majority of the following tasks.

An active/pending validator can update its on-chain metadata by submitting a transaction. Some metadata changes take effect immediately, including:

Other metadata (keys, addresses, and so on) only come into effect at the next epoch.

To update metadata, a validator makes a MoveCall transaction that interacts with the System Object. For example:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

See the full list of metadata update functions here .

To avoid touching account keys too often and allowing them to be stored off-line, validators can delegate the operation ability to another address. This address can then update the reference gas price and tallying rule on behalf of the validator.

Upon creating a Validator , an UnverifiedValidatorOperationCap is created as well and transferred to the validator address. The holder of this Cap object (short for "Capability") therefore could perform operational actions for this validator. To authorize another address to conduct these operations, a validator transfers the object to another address that they control. The transfer can be done by using Sui Client CLI: sui client transfer .

To rotate the delegatee address or revoke the authorization, the current holder of Cap transfers it to another address. In the event of compromised or lost keys, the validator could create a new Cap object to invalidate the incumbent one. This is done by calling sui_system::rotate_operation_cap :

By default the new Cap object is transferred to the validator address, which then could be transferred to the new delegatee address. At this point, the old Cap becomes invalidated and no longer represents eligibility.

To get the current valid Cap object's ID of a validator, use the Sui Client CLI sui client objects command after setting the holder as the active address.

To update the gas price survey quote of a validator, which is used to calculate the reference gas price at the end of the epoch, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::request_set_gas_price :

To report a validator or undo an existing reporting, the sender needs to hold a valid UnverifiedValidatorOperationCap . The sender could be the validator itself, or a trusted delegatee. To do so, call sui_system::report_validator/undo_report_validator :

After a validator is reported by 2f + 1 other validators by voting power, their staking rewards will be slashed.

In order for a Sui address to join the validator set, they need to first sign up as a validator candidate by calling sui_system::request_add_validator_candidate with their metadata and initial configs:

After an address becomes a validator candidate, any address (including the candidate address itself) can start staking with the candidate's staking pool. Refer to our dedicated staking FAQ on how staking works. Once a candidate's staking pool has accumulated at least sui_system::MIN_VALIDATOR_JOINING_STAKE amount of stake, the candidate can call sui_system::request_add_validator to officially add themselves to next epoch's active validator set:

To leave the validator set starting next epoch, the sender needs to be an active validator in the current epoch and should call sui_system::request_remove_validator :

After the validator is removed at the next epoch change, the staking pool will become inactive and stakes can only be withdrawn from an inactive pool.

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh

Private security fixes

There might be instances where urgent security fixes need to be rolled out before publicly announcing it's presence (issues affecting liveliness, invariants such as SUI supply, governance, and so on). To not be actively exploited, Mysten Labs will release signed security binaries incorporating such fixes with a delay in publishing the source code until a large percentage of our validators have patched the vulnerability.

This release process is different and we expect us to announce the directory for such binaries out of band. Our public key to verify these binaries would be stored here

There is also a script available that downloads all the necessary signed binaries and docker artifacts incorporating the security fixes.

Usage ./download_private.sh

You can also download and verify specific binaries that may not be included by the above script using the download_and_verify_private_binary.sh script.

Usage: ./download_and_verify_private_binary.sh